GTK+ Reference Manual

for GTK+ 2.5.4

Table of Contents

GtkTreeViewColumn - A visible column in a GtkTreeView widget
GtkTreeView - A widget for displaying both trees and lists
GtkTreeView drag-and-drop - Interfaces for drag-and-drop support in GtkTreeView
GtkCellView - A widget displaying a single row of a GtkTreeModel
GtkIconView - A widget which displays a list of icons in a grid
GtkTreeSortable - The interface for sortable models used by GtkTreeView
GtkTreeModelSort - A GtkTreeModel which makes an underlying tree model sortable
GtkTreeModelFilter - A GtkTreeModel which hides parts of an underlying tree model
GtkCellLayout - An interface for packing cells
GtkCellRenderer - An object for rendering a single cell on a GdkDrawable
GtkCellEditable - Interface for widgets which can are used for editing cells
GtkCellRendererCombo -
GtkCellRendererPixbuf - Renders a pixbuf in a cell
GtkCellRendererProgress - Renders numbers as progress bars
GtkCellRendererText - Renders text in a cell
GtkCellRendererToggle - Renders a toggle button in a cell
GtkListStore - A list-like data structure that can be used with the GtkTreeView
GtkTreeStore - A tree-like data structure that can be used with the GtkTreeView

Menus, Combo Box, Toolbar

GtkComboBox - A widget used to choose from a list of items
GtkComboBoxEntry - A text entry field with a dropdown list
GtkMenu - A menu widget
GtkMenuBar - A subclass widget for GtkMenuShell which holds GtkMenuItem widgets
GtkMenuItem - The widget used for item in menus
GtkMenuShell - A base class for menu objects
GtkImageMenuItem - A menu item with an icon
GtkRadioMenuItem - A choice from multiple check menu items
GtkCheckMenuItem - A menu item with a check box
GtkSeparatorMenuItem - A separator used in menus
GtkTearoffMenuItem - A menu item used to tear off and reattach its menu
GtkToolbar - Create bars of buttons and other widgets
GtkToolItem - The base class of widgets that can be added to GtkToolbar
GtkSeparatorToolItem - A toolbar item that separates groups of other toolbar items
GtkToolButton - A GtkToolItem subclass that displays buttons
GtkMenuToolButton - A GtkToolItem containing a button with an additional dropdown menu
GtkToggleToolButton - A GtkToolItem containing a toggle button
GtkRadioToolButton - A toolbar item that contains a radio button

Part I. GTK+ Overview >>

GTK+ Overview

GTK+ is a library for creating graphical user interfaces. It works on many UNIX-like platforms, Windows, and on framebuffer devices. GTK+ is released under the GNU Library General Public License (GNU LGPL), which allows for flexible licensing of client applications. GTK+ has a C-based object-oriented architecture that allows for maximum flexibility. Bindings for other languages have been written, including C++, Objective-C, Guile/Scheme, Perl, Python, TOM, Ada95, Free Pascal, and Eiffel.

GTK+ depends on the following libraries:

GLib	A general-purpose utility library, not specific to graphical user interfaces. GLib provides many useful data types, macros, type conversions, string utilities, file utilities, a main loop abstraction, and so on.
Pango	Pango is a library for internationalized text handling. It centers around the PangoLayout object, representing a paragraph of text. Pango provides the engine for GtkTextView, GtkLabel, GtkEntry, and other widgets that display text.
ATK	ATK is the Accessibility Toolkit. It provides a set of generic interfaces allowing accessibility technologies to interact with a graphical user interface. For example, a screen reader uses ATK to discover the text in an interface and read it to blind users. GTK+ widgets have built-in support for accessibility using the ATK framework.
GdkPixbuf	This is a small library which allows you to create GdkPixbuf ("pixel buffer") objects from image data or image files. Use a GdkPixbuf in combination with GtkImage to display images.
GDK	GDK is the abstraction layer that allows GTK+ to support multiple windowing systems. GDK provides drawing and window system facilities on X11, Windows, and the Linux framebuffer device.
GTK+	The GTK+ library itself contains *widgets*, that is, GUI components such as GtkButton or GtkTextView.

Compiling the GTK+ libraries

Compiling the GTK+ Libraries — How to compile GTK+ itself

Building GTK+ on UNIX-like systems

This chapter covers building and installing GTK+ on UNIX and UNIX-like systems such as Linux. Compiling GTK+ on Microsoft Windows is different in detail and somewhat more difficult to get going since the necessary tools aren't included with the operating system.

Before we get into the details of how to compile GTK+, we should mention that in many cases, binary packages of GTK+ prebuilt for your operating system will be available, either from your operating system vendor or from independent sources. If such a set of packages is available, installing it will get you programming wih GTK+ much faster than building it yourself. In fact, you may well already have GTK+ installed on your system already.

On UNIX-like systems GTK+ uses the standard GNU build system, using autoconf for package configuration and resolving portability issues, automake for building makefiles that comply with the GNU Coding Standards, and libtool for building shared libraries on multiple platforms.

If you are building GTK+ from the distributed source packages, then won't need these tools installed; the necessary pieces of the tools are already included in the source packages. But it's useful to know a bit about how packages that use these tools work. A source package is distributed as a `tar.gz` file which you unpack into a directory full of the source files as follows:

```
tar xvfz gtk+-2.0.0.tar.gz
```

In the toplevel of the directory that is created, there will be a shell script called `configure` which you then run to take the template makefiles called `Makefile.in` in the package and create makefiles customized for your operating system. The `configure` script can be passed various command line arguments to determine how the package is built and installed. The most commonly useful argument is the `--prefix` argument which determines where the package is installed. To install a package in `/opt/gtk` you would run configure as:

```
./configure --prefix=/opt/gtk
```

A full list of options can be found by running `configure` with the `--help` argument. In general, the defaults are right and should be trusted. After you've run `configure`, you then run the **make** command to build the package and install it.

```
make
make install
```

If you don't have permission to write to the directory you are installing in, you may have to change to root temporarily before running `make install`. Also, if you are installing in a system directory, on some systems (such as Linux), you will need to run **ldconfig** after `make install` so that the newly installed libraries will be found.

Several environment variables are useful to pass to set before running configure. `CPPFLAGS` contains options to pass to the C compiler, and is used to tell the compiler where to look for include files. The `LDFLAGS` variable is used in a similar fashion for the linker. Finally the `PKG_CONFIG_PATH` environment variable contains a search path that **pkg-config** (see below) uses when looking for for file describing how to compile programs using different libraries. If you were installing GTK+ and it's dependencies into `/opt/gtk`, you might want to set these variables as:

```
CPPFLAGS="-I/opt/gtk/include"
LDFLAGS="-L/opt/gtk/lib"
PKG_CONFIG_PATH="/opt/gtk/lib/pkgconfig"
export CPPFLAGS LDFLAGS PKG_CONFIG_PATH
```

You may also need to set the `LD_LIBRARY_PATH` environment variable so the systems dynamic linker can find the newly installed libraries, and the `PATH` environment program so that utility binaries installed by the various libraries will be found.

```
LD_LIBRARY_PATH="/opt/gtk/lib"
PATH="/opt/gtk/bin:$PATH"
export LD_LIBRARY_PATH PATH
```

Dependencies

Before you can compile the GTK+ widget toolkit, you need to have various other tools and libraries installed on your system. The two tools needed during the build process (as differentiated from the tools used in when creating GTK+ mentioned above such as autoconf) are **pkg-config** and GNU make.

- pkg-config is a tool for tracking the compilation flags needed for libraries that are used by the GTK+ libraries. (For each library, a small `.pc` text file is installed in a standard location that contains the compilation flags needed for that library along with version number information.) The version of **pkg-config** needed to build GTK+ is mirrored in the `dependencies` directory on the GTK+ FTP site.
- The GTK+ makefiles will mostly work with different versions of **make**, however, there tends to be a few incompatibilities, so the GTK+ team recommends installing GNU make if you don't already have it on your system and using it. (It may be called **gmake** rather than **make**.)

Three of the libraries that GTK+ depends on are maintained by by the GTK+ team: GLib, Pango, and ATK. Other libraries are maintained separately.

- The GLib library provides core non-graphical functionality such as high level data types, Unicode manipulation, and an object and type system to C programs. It is available from the GTK+ FTP site.
- Pango is a library for internationalized text handling. It is available from the GTK+ FTP site.. Either Pango-1.0 or Pango-1.2 can be used with GTK+-2.2, though Pango-1.2 is recommended.
- ATK is the Accessibility Toolkit. It provides a set of generic interfaces allowing accessibility technologies such as screen readers to interact with a graphical user interface. It is available from the GTK+ FTP site.
- The GNU libiconv library is needed to build GLib if your system doesn't have the `iconv()` function for doing conversion between character encodings. Most modern systems should have `iconv()`.
- The libintl library from the GNU gettext package is needed if your system doesn't have the `gettext()` functionality for handling message translation databases.
- The JPEG, PNG, and TIFF image loading libraries are needed to compile GTK+. You probably already have these libraries installed, but if not, the versions you need are available in the `dependencies` directory on the the GTK+ FTP site.. (Before installing these libraries from source, you should check if your operating system vendor has prebuilt packages of these libraries that you don't have installed.)
- The libraries from the X window system are needed to build Pango and GTK+. You should already have these installed on your system, but it's possible that you'll need to install the development environment for these libraries that your operating system vendor provides.
- The fontconfig library provides Pango with a standard way of locating fonts and matching them

against font names. The Xft2 library, distributed with fontconfig, provides support for scalable and antialiased fonts on X. Pango includes two backends that work on top of fontconfig: an Xft2 backend and a backend that uses fontconfig and the underlying FreeType library directly. Neither backend is mandatory, but the Xft2 backend is the preferred backend for X and the FreeType backend is needed by many applications.

Building and testing GTK+

First make sure that you have the necessary external dependencies installed: **pkg-config**, GNU make, the JPEG, PNG, and TIFF libraries, FreeType, and, if necessary, libiconv and libintl. To get detailed information about building these packages, see the documentation provided with the individual packages. On a Linux system, it's quite likely you'll have all of these installed already except for **pkg-config**.

Then build and install the GTK+ libraries in the order: GLib, Pango, ATK, then GTK+. For each library, follow the steps of `configure`, `make`, `make install` mentioned above. If you're lucky, this will all go smoothly, and you'll be ready to start compiling your own GTK+ applications. You can test your GTK+ installation by running the **gtk-demo** program that GTK+ installs.

If one of the `configure` scripts fails or running **make** fails, look closely at the error messages printed; these will often provide useful information as to what went wrong. When `configure` fails, extra information, such as errors that a test compilation ran into, is found in the file `config.log`. Looking at the last couple of hundred lines in this file will frequently make clear what went wrong. If all else fails, you can ask for help on the gtk-list mailing list. See Mailing lists and bug reports(3) for more information.

Extra Configuration Options

In addition to the normal options, the **configure** script for the GTK+ library supports a number of additional arguments. (Command line arguments for the other GTK+ libraries are described in the documentation distributed with the those libraries.)

`configure` [[--disable-modules] | [--enable-modules]] [[--with-included-loaders==LOADER1,LOADER2,...]] [[--enable-debug=[no|minimum|yes]]] [[--disable-visibility] | [--enable-visibility]] [[--disable-shm] | [--enable-shm]] [[--disable-xim] | [--enable-xim]] [[--disable-xim-inst] | [--enable-xim-inst]] [[--disable-xkb] | [--enable-xkb]] [[--disable-gtk-doc] | [--enable-gtk-doc]] [[--with-xinput=[no|yes]]] [[--with-gdktarget=[x11|linux-fb|win32]]] [[--disable-shadowfb] | [--enable-shadowfb]]

`--disable-modules` and `--enable-modules`. Normally GTK+ will try to build the GdkPixbuf image file format loaders as little shared libraries that are loaded on demand. The `--disable-modules` argument indicates that they should all be built statically into the GTK+ library instead. This is

useful for people who need to produce statically-linked binaries. If neither `--disable-modules` nor `--enable-modules` is specified, then the **configure** script will try to auto-detect whether shared modules work on your system.

`--with-included-loaders`. This option allows you to specify which image loaders you want to include; for example, you might include only the PNG loader to create a smaller GdkPixbuf binary.

`--enable-debug`. Turns on various amounts of debugging support. Setting this to 'no' disables g_assert(), g_return_if_fail(), g_return_val_if_fail() and all cast checks between different object types. Setting it to 'minimum' disables only cast checks. Setting it to 'yes' enables runtime debugging. The default is 'minimum'. Note that 'no' is fast, but dangerous as it tends to destabilize even mostly bug-free software by changing the effect of many bugs from simple warnings into fatal crashes. Thus `--enable-debug=no` should *not* be used for stable releases of GTK+.

`--disable-visibility` and `--enable-visibility`. The option `--disable-visibility` turns off the use of ELF visibility attributes for linking optimizations. This makes sense while changing GTK+ itself, since the way in which GTK+ uses visibility attributes forces a full rebuild of all source files for any header modification.

`--enable-explicit-deps` and `--disable-explicit-deps`. If `--enable-explicit-deps` is specified then GTK+ will write the full set of libraries that GTK+ depends upon into its `.pc` files to be used when programs depending on GTK+ are linked. Otherwise, GTK+ only will include the GTK+ libraries themselves, and will depend on system library dependency facilities to bring in the other libraries. By default GTK+ will disable explicit dependencies unless it detects that they are needed on the system. (If you specify `--enable-static` to force building of static libraries, then explicit dependencies will be written since library dependencies don't work for static libraries.) Specifying `--enable-explicit-deps` or `--enable-static` can cause compatibility problems when libraries that GTK+ depends upon change their versions, and should be avoided if possible.

`--disable-shm` and `--enable-shm`. These options can be used to control whether GTK+ will use shared memory to communicate with the X server when possible. The default is 'yes'.

`--disable-xim` and `--enable-xim`. These options can be used to control whether GTK+ will be compiled with support for XIM. (The X Input Method extension, used for Japanese input.) The default is yes.

`--disable-xim-inst` and `--enable-xim-inst`. These options determine whether GTK+ will use the XIM instantiate callback. The default is 'yes', unless the host system is Solaris, where `XRegisterIMInstantiateCallback()` seems to cause a segfault.

`--disable-xkb` and `--enable-xkb`. By default the **configure** script will try to auto-detect whether the XKB extension is supported by the X libraries GTK+ is linked with. These options can be

used to explicitly control whether GTK+ will support the XKB extension.

`--disable-gtk-doc` and `--enable-gtk-doc`. The gtk-doc package is used to generate the reference documentation included with GTK+. By default support for gtk-doc is disabled because it requires various extra dependencies to be installed. If you have gtk-doc installed and are modifying GTK+, you may want to enable gtk-doc support by passing in `--enable-gtk-doc`. If not enabled, pre-generated HTML files distributed with GTK+ will be installed.

`--with-xinput`. Controls whether GTK+ is built with support for the XInput extension. The XInput extension provides an interface to extended input devices such as graphics tablets. When this support is compiled in, specially written GTK+ programs can get access to subpixel positions, multiple simultaneous input devices, and extra "axes" provided by the device such as pressure and tilt information. This is only known to work well on XFree86 systems, though other systems do have this extension.

`--with-gdktarget`. Toggles between the supported backends for GDK. The default is x11, unless the platform is Windows, in which case the default is win32.

`--disable-shadowfb` and `--enable-shadowfb`. Toggles shadow framebuffer support for the linux-fb target, if selected.

<< Part I. GTK+ Overview **Compiling GTK+ Applications >>**

Compiling GTK+ Applications

Compiling GTK+ Applications — How to compile your GTK+ application

Compiling GTK+ Applications on UNIX

To compile a GTK+ application, you need to tell the compiler where to find the GTK+ header files and libraries. This is done with the `pkg-config` utility.

The following interactive shell session demonstrates how `pkg-config` is used:

```
$ pkg-config --cflags gtk+-2.0
 -I/usr/include/gtk-2.0 -I/usr/lib/gtk-2.0/include -I/usr/include/glib-2.0 -
I/usr/lib/glib-2.0/include -I/usr/include/pango-1.0 -I/usr/X11R6/include -
I/usr/include/freetype2 -I/usr/include/atk-1.0
$ pkg-config --libs gtk+-2.0
 -L/usr/lib -L/usr/X11R6/lib -lgtk-x11-2.0 -lgdk-x11-2.0 -lXi -lgdk_pixbuf-2.0 -lm -
lpangox -lpangoxft -lXft -lXrender -lXext -lX11 -lfreetype -lpango -latk -lgobject-
2.0 -lgmodule-2.0 -ldl -lglib-2.0
```

The simplest way to compile a program is to use the "backticks" feature of the shell. If you enclose a command in backticks (*not single quotes*), then its output will be substituted into the command line before execution. So to compile a GTK+ Hello, World, you would type the following:

```
$ cc `pkg-config --cflags --libs gtk+-2.0` hello.c -o hello
```

To compile a GTK+ program for the framebuffer, use the "gtk+-linux-fb-2.0" package name instead of "gtk+-2.0":

```
$ cc `pkg-config --cflags --libs gtk+-linux-fb-2.0` hello.c -o hello
```

If you want to make sure that your program doesn't use any deprecated functions, you can define the preprocessor symbol GTK_DISABLE_DEPRECATED by using the command line option `-DGTK_DISABLE_DEPRECATED=1`. There are similar symbols GDK_DISABLE_DEPRECATED, GDK_PIXBUF_DISABLE_DEPRECATED and G_DISABLE_DEPRECATED for GDK, GdkPixbuf and GLib.

If you want to make sure that your program doesn't use any functions which may be problematic in a multihead setting, you can define the preprocessor symbol GDK_MULTIHEAD_SAFE by using the command line option `-DGTK_MULTIHEAD_SAFE=1`.

Running GTK+ Applications

Running GTK+ Applications — How to run and debug your GTK+ application

Running and debugging GTK+ Applications

Common commandline options

All GTK+ applications support a number of standard commandline options. These are removed from `argv` by gtk_init(). Modules may parse and remove further options. The X11 and Windows GDK backends parse some additional commandline options.

`--gtk-module` *`module`*. A list of modules to load in addition to those specified in the `GTK_MODULES` environment variable and the `gtk-modules` setting.

`--g-fatal-warnings`. Make GTK+ abort on all warnings. This is useful to stop on the first warning in a debugger, if your application is printing multiple warnings. It's almost always best to start debugging with the first warning that occurs.

`--gtk-debug` *`options`*. A list of debug options to turn on in addition to those specified in the `GTK_DEBUG` environment variable. This option is only available if GTK+ has been configured with `--enable-debug=yes`.

`--gtk-no-debug` *`options`*. A list of debug options to turn off. This option is only available if GTK+ has been configured with `--enable-debug=yes`.

The following options are really used by GDK, not by GTK+, but we list them here for completeness nevertheless.

`--class` *`class`*. Sets the program class; see `gdk_set_program_class()`.

`--name` *`name`*. Sets the program name.

`--gdk-debug` *`options`*. A list of debug options to turn on in addition to those specified in the

GDK_DEBUG environment variable. This option is only available if GTK+ has been configured with --enable-debug=yes.

--gdk-no-debug *options*. A list of debug options to turn off. This option is only available if GTK+ has been configured with --enable-debug=yes.

Environment variables

GTK+ inspects a number of environment variables in addition to standard variables like LANG, PATH, HOME or DISPLAY; mostly to determine paths to look for certain files. The X11, Windows and Framebuffer GDK backends use some additional environment variables.

GTK_DEBUG. If GTK+ has been configured with --enable-debug=yes, this variable can be set to a list of debug options, which cause GTK+ to print out different types of debugging information.

misc	Miscellaneous information
plugsocket	Cross-process embedding
text	Text widget internals
tree	Tree widget internals
updates	Visual feedback about window updates
keybindings	Keybindings
multihead	Working on multiple displays
modules	Loading of modules
geometry	Size allocation

The special value all can be used to turn on all debug options.

GTK_MODULES. A list of modules to load.

GTK_PATH. Specifies a list of directories to search when GTK+ is looking for dynamically loaded objects such as the modules specified by GTK_MODULES, theme engines, and input method modules. If the path to the dynamically loaded object is given as an absolute path name, then GTK+ loads it directly. Otherwise, GTK+ goes in turn through the directories in GTK_PATH, followed by the directory .gtk-2.0 in the user's home directory, followed by the system default directory, which is *libdir*/gtk-2.0/modules. (If GTK_EXE_PREFIX is defined, *libdir* is $GTK_EXE_PREFIX/lib. Otherwise it is the libdir specified when GTK+ was configured, usually /usr/lib, or /usr/local/lib.) For each directory in this list, GTK+ actually looks in a subdirectory *directory/version/host/type*

Where *version* is derived from the version of GTK+ (use `pkg-config --variable=gtk_binary_version gtk+-2.0` to determine this from a script), *host* is the architecture on which GTK+ was built. (use `pkg-config --variable=gtk_host gtk+-2.0` to determine this from a script), and *type* is a directory specific to the type of modules; currently it can be `modules`, `engines` or `immodules` corresponding to the three types of modules above. Either *version*, *host*, or both may be omitted. GTK+ looks first in the most specific directory, then in directories with fewer components. The components of GTK_PATH are separated by the ':' character on Linux and Unix, and the ';' character on Windows.

GTK_IM_MODULE. Specifies an IM module to use in preference to the one determined from the locale.

GTK_IM_MODULE_FILE. Specifies the file listing the IM modules to load. This environment variable overrides the `im_module_file` specified in the RC files, which in turn overrides the default value *sysconfdir*`/gtk-2.0/gtk.immodules` (*sysconfdir* is the sysconfdir specified when GTK+ was configured, usually `/usr/local/etc`.)

GTK2_RC_FILES. Specifies a list of RC files to parse instead of the default ones; see Resource Files.

GTK_EXE_PREFIX. If set, GTK+ uses `$GTK_EXE_PREFIX/lib` instead of the libdir configured when GTK+ was compiled.

GTK_DATA_PREFIX. If set, makes GTK+ use `$GTK_DATA_PREFIX` instead of the prefix configured when GTK+ was compiled.

The following environment variables are used by GdkPixbuf, GDK or Pango, not by GTK+ itself, but we list them here for completeness nevertheless.

GDK_PIXBUF_MODULE_FILE. Specifies the file listing the GdkPixbuf loader modules to load. This environment variable overrides the default value *sysconfdir*`/gtk-2.0/gdk-pixbuf.loaders` (*sysconfdir* is the sysconfdir specified when GTK+ was configured, usually `/usr/local/etc`.)

GDK_DEBUG. If GTK+ has been configured with `--enable-debug=yes`, this variable can be set to a list of debug options, which cause GDK to print out different types of debugging information.

misc	Miscellaneous information
events	Show all events received by GDK
dnd	Information about drag-and-drop
xim	Information about XIM support

The special value `all` can be used to turn on all debug options.

XDG_DATA_HOME, XDG_DATA_DIRS. GTK+ uses these environment variables to locate icon themes and MIME information. For more information, see Icon Theme Specification, the Shared MIME-info Database and the Base Directory Specification.

<< Compiling GTK+ Applications **Using GTK+ on Windows >>**

Using GTK+ on Windows

Using GTK+ on Windows — Windows-specific aspects of using GTK+

Using GTK+ on Windows

The Windows port of GTK+ is an implementation of GDK (and therefore GTK+) on top of the Win32 API.

Windows-specific commandline options

The Windows GDK backend can be influenced with some additional command line arguments.

`--sync`. Don't batch GDI requests. This is a useful option for debugging, but it will slow down the performance considerably.

`--no-wintab`, `--ignore-wintab`. Don't use the Wintab API for tablet support. This is the default.

`--use-wintab`. Use the Wintab API for tablet support, if GTK+ has been compiled with the `--with-wintab` option.

`--max-colors` *`number`*. In 256 color mode, restrict the size of the color palette to the specified number of colors.

Windows-specific environment variables

The Win32 GDK backend can be influenced with some additional environment variables.

`GDK_IGNORE_WINTAB`. If this variable is set, GTK+ doesn't use the Wintab API for tablet support. This is the default.

`GDK_USE_WINTAB`. If this variable is set, GTK+ uses the Wintab API for tablet support, if GTK+ has been compiled with the `--with-wintab` option.

GDK_WIN32_MAX_COLORS. Specifies the size of the color palette used in 256 color mode.

PRETEND_WIN9X. Only use the Windows 9x API.

<< Running GTK+ Applications **Using GTK+ on the Framebuffer >>**

Using GTK+ on the Framebuffer

Using GTK+ on the Framebuffer — Linux framebuffer aspects of using GTK+

GTK+ for the Linux Framebuffer

The linux-fb port of GTK+, also known as GtkFB is an implementation of GDK (and therefore GTK+) that runs on the Linux framebuffer. It runs in a single process that doesn't need X. It should run most GTK+ programs without any changes to the source.

Build requirements

You need GTK+ 2.0; the 1.2.x series does not have framebuffer support. To compile GTK+ with framebuffer support you will need FreeType 2; we recommend FreeType 2.0.1 or later, as there was some problems with **freetype-config** in 2.0. Make sure that you install FreeType before Pango, since Pango also needs it. FreeType can be found at ftp://ftp.freetype.org. You also need fontconfig in order to properly use the FreeType 2 backend in Pango. Fontconfig depends on FreeType as well. Fontconfig can be found at http://fontconfig.org.

Hardware requirements

You need a graphics card with an available framebuffer driver that can run in 8, 16, 24 or 32 bpp, such as matroxfb or vesafb. You also need a supported mouse. GTK+ currently supports the ps2 mouse, ms serial mouse and fidmour touchscreen. Additional hardware support should be simple to add.

Building and installing

First build and install GLib, ATK and Pango as usual, in that order. Then configure GTK+ by running **configure** (or **autogen.sh** if running from CVS) with `--with-gdktarget=linux-fb`.

Then compile as usual: **make; make install**

Fonts

Since GtkFB uses FreeType 2 to render fonts it can render TrueType and Postscript type 1 antialiased fonts.

GtkFB uses fontconfig for configuration of font information, including directories and aliases. Make sure that you have your fonts.conf file configured with where your TrueType and Type1 fonts are. Please refer to the fontconfig documentation for more information.

Running

To run a program you should only need to start it, but there are some things that can cause problems, and some things that can be controlled by environment variables. Try gtk-demo distributed with GTK+ to test if things work.

If you use a ps2 mouse, make sure that `/dev/psaux` is readable and writable.

Make sure **gpm** is not running.

If you don't specify anything GtkFB will start up in the current virtual console in the current resolution and bit-depth. This can be changed by specifying environment variables:

```
GDK_VT:
 unset means open on the current VT.
 0-9: open on the specified VT. Make sure you have read/write rights
      there.
 new: Allocate a new VT after the last currently used one.

GDK_DISPLAY_MODE:
 Specifies the name of a mode in /etc/fb.modes that you
 want to use.

GDK_DISPLAY_DEPTH:
 Specify the desired bit depth of the framebuffer.
```

```
GDK_DISPLAY_WIDTH:
 Specify the desired width of the framebuffer.

GDK_DISPLAY_HEIGHT:
 Specify the desired height of the framebuffer.

GDK_DISPLAY:
 Specify the framebuffer device to use. Default is /dev/fb0.

GDK_MOUSE_TYPE:
 Specify mouse type. Currently supported is:
  ps2 - PS/2 mouse
  imps2 - PS/2 intellimouse (wheelmouse)
  ms - Microsoft serial mouse
  fidmour - touch screen
 Default is ps2.

GDK_KEYBOARD_TYPE:
 Specify keyboard type. Currently supported is
  xlate - normal tty mode keyboard.
    Quite limited, cannot detect key up/key down events. Doesn't
    handle ctrl/alt/shift for all keys. This is the default driver,
    but should not be used in "production" use.
  raw - read from the tty in RAW mode.
    Sets the keyboard in RAW mode and handles all the keycodes. This
    gives correct handling of modifiers and key up/down events. You
    must be root to use this. If you use this for development or
    debugging it is recommended to enable magic sysrq handling in the
    kernel. Then you can use ALT-SysRQ-r to turn the keyboard back to
    normal mode.
 Default is xlate.
```

Debug features

Pressing Ctrl-Alt-Return repaints the whole screen. Unfortunately this cannot be pressed when using the xlate keyboard driver, so instead you can use shift-F1 instead when using this driver.

Pressing Ctrl-Alt-BackSpace kills the GtkFB program. (This can't be pressed in the xlate driver, so instead you can use shift-F8.)

<< Using GTK+ on Windows **Using GTK+ on the X Window System >>**

Using GTK+ on the X Window System

Using GTK+ on the X Window System — X11 aspects of using GTK+

GTK+ for the X Window System

On UNIX, the X backend is the default build for GTK+. So you don't need to do anything special when compiling it, and everything should "just work."

To mix low-level Xlib routines into a GTK program, see GDK X Window System interaction in the GDK manual.

X11-specific commandline options

The X backend understands some additional command line arguments.

`--display` *`display`*. The name of the X display to open instead of the one specified in the `DISPLAY` environment variable.

`--screen` *`screen_number`*. The number of the screen within the default display. This overrides any screen number specified in the display name specified by by he `--display` command line option or the `DISPLAY` environment variable. If this screen cannot be opened, then GTK+ will fall back to the screen specified in the display name. This option is not useful interactively; the intended purposes is that when a program registers its command line with a *session manager* for later restarting, it can save the screen it is on, without having to worry if it might be restarted on a different display.

`--sync`. Makes all X requests synchronously. This is a useful option for debugging, but it will slow down the performance considerably.

`--gxid-host` *`host`*. The host to contact the gxid daemon on; overrides the `GXID_HOST` environment variable.

`--gxid-port` *`port`*. The port for the connection to gxid; overrides the `GXID_PORT` environment variable. This option is only available if GTK+ has been configured with `--gdk-target=x11`.

X11-specific environment variables

The X backend can be influenced with some additional environment variables.

`GXID_HOST`, `GXID_PORT`. The host and port to contact the gxid daemon on. gxid is only necessary on X servers which don't support using the pointer and extension devices at once, and is only built if GTK+ is configured with `--with-xinput=gxi`. The XFree86 and Xorg X servers don't have this restriction.

`GDK_USE_XFT`. If this variable is set to 1, GTK+ will use the Pango Xft backend instead of the X backend when possible (i.e. when the X server supports the XRender extension and Pango has been built with Xft support).

Understanding the X11 architecture

People coming from a Windows or MacOS background often find certain aspects of the X Window System surprising. This section introduces some basic X concepts at a high level. For more details, the book most people use is called the *Xlib Programming Manual* by Adrian Nye; this book is volume one in the O'Reilly X Window System series.

Standards are another important resource if you're poking in low-level X11 details, in particular the ICCCM and the Extended Window Manager Hints specifications. freedesktop.org has links to many relevant specifications.

The GDK manual covers using Xlib in a GTK program.

Server, client, window manager

Other window systems typically put all their functionality in the application itself. With X, each application involves three different programs: the *X server*, the application (called a *client* because it's a client of the X server), and a special client called the *window manager*.

The X server is in charge of managing resources, processing drawing requests, and dispatching events such as keyboard and mouse events to interested applications. So client applications can ask the X server to create a window, draw a circle, or move windows around.

The window manager is in charge of rendering the frame or borders around windows; it also has final say on the size of each window, and window states such as minimized, maximized, and so forth. On Windows and MacOS the application handles most of this. On X11, if you wish to modify the window's

state, or change its frame, you must ask the window manager to do so on your behalf, using an established convention.

GTK+ has functions for asking the window manager to do various things; see for example gtk_window_iconify() or gtk_window_maximize() or gtk_window_set_decorated(). Keep in mind that gtk_window_move() and window sizing are ultimately controlled by the window manager as well and most window managers *will* ignore certain requests from time to time, in the interests of good user interface.

<< Using GTK+ on the Framebuffer **Mailing lists and bug reports >>**

Mailing lists and bug reports

Mailing lists and bug reports — Getting help with GTK+

Filing a bug report or feature request

If you encounter a bug, misfeature, or missing feature in GTK+, please file a bug report on http://bugzilla.gnome.org. We'd also appreciate reports of incomplete or misleading information in the GTK+ documentation; file those against the "docs" component of the "gtk+" product in Bugzilla.

Don't hesitate to file a bug report, even if you think we may know about it already, or aren't sure of the details. Just give us as much information as you have, and if it's already fixed or has already been discussed, we'll add a note to that effect in the report.

The bug tracker should definitely be used for feature requests, it's not only for bugs. We track all GTK+ development in Bugzilla, so it's the way to be sure the GTK+ developers won't forget about an issue.

Submitting Patches

If you develop a bugfix or enhancement for GTK+, please file that in Bugzilla as well. Bugzilla allows you to attach files; please attach a patch generated by the **diff** utility, using the `-u` option to make the patch more readable. All patches must be offered under the terms of the GNU LGPL license, so be sure you are authorized to give us the patch under those terms.

If you want to discuss your patch before or after developing it, mail gtk-devel-list@gnome.org. But be sure to file the Bugzilla report as well; if the patch is only on the list and not in Bugzilla, it's likely to slip through the cracks.

Mailing lists

There are several mailing lists dedicated to GTK+ and related libraries. Discussion of GLib, Pango, and ATK in addition to GTK+ proper is welcome on these lists. You can subscribe or view the archives of these lists on http://mail.gnome.org. If you aren't subscribed to the list, any message you post to the list will be held for manual moderation, which might take some days to happen.

gtk-list@gnome.org	gtk-list covers general GTK+ topics; questions about using GTK+ in programs, GTK+ from a user standpoint, announcements of GTK+-related projects such as themes or GTK+ modules would all be on-topic. The bulk of the traffic consists of GTK+ programming questions.
gtk-app-devel-list@gnome.org	gtk-app-devel-list covers writing applications in GTK+. It's narrower in scope than gtk-list, but the two lists overlap quite a bit. gtk-app-devel-list is a good place to ask questions about GTK+ programming.
gtk-devel-list@gnome.org	gtk-devel-list is for discussion of work on GTK+ itself, it is *not* for asking questions about how to use GTK+ in applications. gtk-devel-list is appropriate for discussion of patches, bugs, proposed features, and so on.
gtk-i18n-list@gnome.org	gtk-i18n-list is for discussion of internationalization in GTK+; Pango is the main focus of the list. Questions about the details of using Pango, and discussion of proposed Pango patches or features, are all on topic.
gtk-doc-list@gnome.org	gtk-doc-list is for discussion of the gtk-doc documentation system (used to document GTK+), and for work on the GTK+ documentation.

<< Using GTK+ on the X Window System **Common Questions >>**

Common Questions

Common Questions — Find answers to common questions in the GTK+ manual

Questions and Answers

This is an "index" of the reference manual organized by common "How do I..." questions. If you aren't sure which documentation to read for the question you have, this list is a good place to start.

1. General

1.1. How do I get started with GTK+?

The GTK+ website offers a tutorial and a FAQ. More documentation ranging from whitepapers to online books can be found at the GNOME developer's site. After studying these materials you should be well prepared to come back to this reference manual for details.

1.2. Where can I get help with GTK+, submit a bug report, or make a feature request?

See the documentation on this topic.

1.3. How do I port from one GTK+ version to another?

See the list of incompatible changes from 1.2 to 2.0. Also, the GNOME 2.0 porting guide on http://developer.gnome.org has some more detailed discussion of porting from 1.2 to 2.0. You may also find useful information in the documentation for specific widgets and functions.

If you have a question not covered in the manual, feel free to ask on the mailing lists and please file a bug report against the documentation.

1.4. How does memory management work in GTK+? Should I free data returned from functions?

See the documentation for GObject and GtkObject. For GObject note specifically g_object_ref() and g_object_unref(). GtkObject is a subclass of GObject so the same points apply, except that it has a "floating" state (explained in its documentation).

For strings returned from functions, they will be declared "const" (using G_CONST_RETURN) if they should not be freed. Non-const strings should be freed with g_free(). Arrays follow the same rule. (If you find an exception to the rules, please report a bug to http://bugzilla.gnome.org.)

1.5. How do I use GTK+ with threads?

This is covered in the GDK threads documentation. See also the GThread documentation for portable threading primitives.

1.6. How do I internationalize a GTK+ program?

Most people use GNU gettext, already required in order to install GLib. On a UNIX or Linux system with gettext installed, type `info gettext` to read the documentation.

The short checklist on how to use gettext is: call `bindtextdomain()` so gettext can find the files containing your translations, call `textdomain()` to set the default translation domain, then call `gettext()` to look up each string to be translated in the default domain. Conventionally, people define macros as follows for convenience:

```
  #define  _(x)  gettext (x)
  #define N_(x)  x
```

You use `N_()` (N stands for no-op) to mark a string for translation in a context where a function call to `gettext()` is not allowed, such as in an array initializer. You eventually have to call `gettext()` on the string to actually fetch the translation. `_()` both marks the string for translation and actually translates it.

Code using these macros ends up looking like this:

```
 #include <libintl.h>

 #define  _(x)  gettext (x)
 #define N_(x)  x

 static const char *global_variable = N_("Translate this string");

 static void
 make_widgets (void)
 {
    GtkWidget *label1;
    GtkWidget *label2;

    label1 = gtk_label_new (_("Another string to translate"));
    label2 = gtk_label_new (_(global_variable));
...
```

Libraries using gettext should use `dgettext()` instead of `gettext()`, which allows them to specify the translation domain each time they ask for a translation. Libraries should also avoid calling `textdomain()`, since they'll be specifying the domain instead of using the default. For `dgettext()` the `_()` macro can be defined as:

```
  #define _(x) dgettext ("MyDomain", x)
```

1.7. How do I use non-ASCII characters in GTK+ programs ?

GTK+ uses Unicode (more exactly UTF-8) for all text. UTF-8 encodes each Unicode codepoint as a sequence of one to six bytes and has a number of nice properties which make it a good choice for working with Unicode text in C programs:

- ASCII characters are encoded by their familiar ASCII codepoints.
- ASCII characters never appear as part of any other character.
- The zero byte doesn't occur as part of a character, so that UTF-8 strings can be manipulated with the usual C library functions for handling zero-terminated strings.

More information about Unicode and UTF-8 can be found in the UTF-8 and Unicode FAQ for Unix/Linux. GLib provides functions for converting strings between UTF-8 and other encodings, see g_locale_to_utf8() and g_convert().

Text coming from external sources (e.g. files or user input), has to be converted to UTF-8 before being handed over to GTK+. The following example writes the content of a IS0-8859-1 encoded text file to `stdout`:

```
gchar *text, *utf8_text;
gsize length;
GError *error = NULL;

if (g_file_get_contents (filename, &text, &length, NULL))
  {
     utf8_text = g_convert (text, length, "UTF-8", "ISO-8859-1",
                            NULL, NULL, &error);
     if (error != NULL)
       {
         fprintf ("Couldn't convert file %s to UTF-8\n", filename);
         g_error_free (error);
       }
     else
       g_print (utf8_text);
  }
else
  fprintf (stderr, "Unable to read file %s\n", filename);
```

For string literals in the source code, there are several alternatives for handling non-ASCII content:

direct UTF-8	If your editor and compiler are capable of handling UTF-8 encoded sources, it is very convenient to simply use UTF-8 for string literals, since it allows you to edit the strings in "wysiwyg". Note that choosing this option may reduce the portability of your code.
escaped UTF-8	Even if your toolchain can't handle UTF-8 directly, you can still encode string literals in UTF-8 by using octal or hexadecimal escapes like `\212` or `\xa8` to encode each byte. This is portable, but modifying the escaped strings is not very convenient. Be careful when mixing hexadecimal escapes with ordinary text; `"\xa8abcd"` is a string of length 1 !
runtime conversion	If the string literals can be represented in an encoding which your toolchain can handle (e.g. IS0-8859-1), you can write your source files in that encoding and use g_convert() to convert the strings to UTF-8 at runtime. Note that this has some runtime overhead, so you may want to move the conversion out of inner loops.

Here is an example showing the three approaches using the copyright sign © which has Unicode and ISO-8859-1 codepoint 169 and is represented in UTF-8 by the two bytes 194, 169:

```
g_print ("direct UTF-8: ©");
g_print ("escaped UTF-8: \302\251");
text = g_convert ("runtime conversion: ©", -1, "ISO-8859-1", "UTF-8", NULL, NULL,
NULL);
g_print(text);
g_free (text);
```

1.8. How do I use GTK+ with C++?

There are two ways to approach this. The GTK+ header files use the subset of C that's also valid C++, so you can simply use the normal GTK+ API in a C++ program. Alternatively, you can use a "C++ binding" such as gtkmm which provides a C++-native API.

When using GTK+ directly, keep in mind that only functions can be connected to signals, not methods. So you will need to use global functions or "static" class functions for signal connections.

Another common issue when using GTK+ directly is that C++ will not implicitly convert an integer to an enumeration. This comes up when using bitfields; in C you can write the following code:

```
  gdk_window_set_events (gdk_window,
                         GDK_BUTTON_PRESS_MASK | GDK_BUTTON_RELEASE_MASK);
```

while in C++ you must write:

```
  gdk_window_set_events (gdk_window,
                         (GdkEventMask) GDK_BUTTON_PRESS_MASK |
GDK_BUTTON_RELEASE_MASK);
```

There are very few functions that require this cast, however.

1.9. How do I use GTK+ with other non-C languages?

See the list of language bindings on http://www.gtk.org.

1.10. How do I load an image or animation from a file?

To load an image file straight into a display widget, use gtk_image_new_from_file() [1]. To load an image for another purpose, use gdk_pixbuf_new_from_file(). To load an animation, use gdk_pixbuf_animation_new_from_file(). gdk_pixbuf_animation_new_from_file() can also load non-animated images, so use it in combination with gdk_pixbuf_animation_is_static_image() to load a file of unknown type.

To load an image or animation file asynchronously (without blocking), use GdkPixbufLoader.

1.11. How do I draw text ?

To draw a piece of text, use a Pango layout and gdk_draw_layout(), using code like the following:

```
layout = gtk_widget_create_pango_layout (widget, text);
fontdesc = pango_font_description_from_string ("Luxi Mono 12");
pango_layout_set_font_description (layout, fontdesc);
gdk_draw_layout (..., layout);
pango_font_description_free (fontdesc);
g_object_unref (layout);
```

Do not use the deprecated GdkFont and gdk_draw_text().

See also the "Text Handling in GTK 2" section of Porting applications to the GNOME 2.0 platform.

1.12. How do I measure the size of a piece of text ?

To obtain the size of a piece of text, use a Pango layout and pango_layout_get_pixel_size(), using code like the following:

```
layout = gtk_widget_create_pango_layout (widget, text);
fontdesc = pango_font_description_from_string ("Luxi Mono 12");
pango_layout_set_font_description (layout, fontdesc);
pango_layout_get_pixel_size (layout, &width, &height);
pango_font_description_free (fontdesc);
g_object_unref (layout);
```

Do not use the deprecated function gdk_text_width().

See also the "Text Handling in GTK 2" section of Porting applications to the GNOME 2.0 platform.

1.13. How do I make a text view scroll to the end of the buffer automatically ?

The "insert" mark marks the insertion point where gtk_text_buffer_insert() inserts new text into the buffer. The text is inserted *before* the "insert" mark, so that it generally stays at the end of the buffer. If it gets explicitly moved to some other position, e.g. when the user selects some text, use gtk_text_buffer_move_mark() to set it to the desired location before inserting more text. The "insert" mark of a buffer can be obtained with gtk_text_buffer_get_insert().

To ensure that the end of the buffer remains visible, use gtk_text_view_scroll_to_mark() to scroll to the "insert" mark after inserting new text.

2. Which widget should I use...

2.1. ...for lists and trees?

See tree widget overview — you should use the GtkTreeView widget. (A list is just a tree with no branches, so the tree widget is used for lists as well.) Do not use the deprecated widgets GtkTree or GtkCList/GtkCTree in newly-written code, they are less flexible and result in an inferior user interface.

2.2. ...for multi-line text display or editing?

See text widget overview — you should use the GtkTextView widget. Do not use the deprecated widget GtkText in newly-written code, it has a number of problems that are best avoided.

If you only have a small amount of text, GtkLabel may also be appropriate of course. It can be made selectable with gtk_label_set_selectable(). For a single-line text entry, see GtkEntry.

2.3. ...to display an image or animation?

GtkImage can display images in just about any format GTK+ understands. You can also use GtkDrawingArea if you need to do something more complex, such as draw text or graphics over the top of the image.

2.4. ...for presenting a set of mutually-exclusive choices, where Windows would use a combo box?

With GTK+, a GtkOptionMenu is recommended instead of a combo box, if the user is selecting from a fixed set of options. That is, non-editable combo boxes are not encouraged. GtkOptionMenu is much easier to use than GtkCombo as well. Use GtkCombo only when you need the editable text entry.

(As a future enhancement to GTK+, a new widget to replace GtkOptionMenu and GtkCombo is planned. This widget will be themeable to look like either a combo box or the current option menu, and will address some shortcomings in the GtkCombo API. Bug 50554 tracks this issue, if you want to check status or post comments.)

3. GtkWidget

3.1. How do I change the color of a widget?

See gtk_widget_modify_fg(), gtk_widget_modify_bg(), gtk_widget_modify_base(), and gtk_widget_modify_text(). See GTK+ resource files for more discussion. You can also change widget color by installing a resource file and parsing it with gtk_rc_add_default_file(). The advantage of a resource file is that users can then override the color you've chosen.

To change the background color for widgets such as GtkLabel that have no background, place them in a GtkEventBox and set the background of the event box.

3.2. How do I change the font of a widget?

This has several possible answers, depending on what exactly you want to achieve. One option is gtk_widget_modify_font(). Note that this function can be used to change only the font size, as in the following example:

```
PangoFontDesc *font_desc = pango_font_description_new ();
pango_font_description_set_size (font_desc, 40);
gtk_widget_modify_font (widget, font);
pango_font_description_free (font_desc);
```

If you want to make the text of a label larger, you can use gtk_label_set_markup():

```
gtk_label_set_markup (label, "<big>big text</big>");
```

This is preferred for many apps because it's a relative size to the user's chosen font size. See g_markup_escape_text() if you are constructing such strings on the fly.

You can also change the font of a widget by putting

```
gtk-font-name = "Sans 30"
```

in a resource file and parsing it with gtk_rc_add_default_file(). The advantage of a resource file is that users can then override the font you've chosen. See GTK+ resource files for more discussion.

3.3. How do I disable/ghost/desensitize a widget?

In GTK+ a disabled widget is termed "insensitive." See gtk_widget_set_sensitive().

4. GtkTextView

4.1. How do I get the contents of the entire text widget as a string?

See gtk_text_buffer_get_bounds() and gtk_text_buffer_get_text() or gtk_text_iter_get_text().

```
GtkTextIter start, end;
GtkTextBuffer *buffer;
char *text;

buffer = gtk_text_view_get_buffer (GTK_TEXT_VIEW (text_view));
gtk_text_buffer_get_bounds (buffer, &start, &end);
text = gtk_text_iter_get_text (&start, &end);
/* use text */
g_free (text);
```

4.2. How do I make a text widget display its complete contents in a specific font?

If you use gtk_text_buffer_insert_with_tags() with appropriate tags to select the font, the inserted text will have the desired appearance, but text typed in by the user before or after the tagged block will appear in the default style.

To ensure that all text has the desired appearance, use gtk_widget_modify_font() to change the default font for the widget.

5. GtkTreeView

5.1. How do I associate some data with a row in the tree?

Remember that the GtkTreeModel columns don't necessarily have to be displayed. So you can put non-user-visible data in your model just like any other data, and retrieve it with gtk_tree_model_get(). See the tree widget overview.

5.2. What's the GtkTreeView equivalent of gtk_clist_find_row_from_data()?

As there is no separate data column in the GtkTreeModel, there's no built in function to find the iter from data. You can write a custom searching function to walk the tree and find the data, or use gtk_tree_model_foreach().

5.3. How do I put an image and some text in the same column?

You can pack more than one GtkCellRenderer into a single GtkTreeViewColumn using gtk_tree_view_column_pack_start() or gtk_tree_view_column_pack_end(). So pack both a GtkCellRendererPixbuf and a GtkCellRendererText into the column.

5.4. I can set data easily on my GtkTreeStore/GtkListStore models using gtk_list_store_set() and gtk_tree_store_set(), but can't read it back?

Both the GtkTreeStore and the GtkListStore implement the GtkTreeModel interface. Consequentially, the can use any function this interface implements. The easiest way to read a set of data back is to use gtk_tree_model_get().

5.5. How do I change the way that numbers are formatted by GtkTreeView?

Use gtk_tree_view_insert_column_with_data_func() or gtk_tree_view_column_set_cell_data_func() and do the conversion from number to string yourself (with, say, g_strdup_printf()).

The following example demonstrates this:

```
enum
{
  DOUBLE_COLUMN,
  N_COLUMNS
};

GtkListStore *mycolumns;
GtkTreeView *treeview;

void
my_cell_double_to_text (GtkTreeViewColumn *tree_column,
                        GtkCellRenderer   *cell,
                        GtkTreeModel      *tree_model,
                        GtkTreeIter       *iter,
                        gpointer           data)
{
  GtkCellRendererText *cell_text = (GtkCellRendererText *)cell;
  gdouble d;
  gchar *text;

  /* Get the double value from the model. */
  gtk_tree_model_get (tree_model, iter, (gint)data, &d, -1);
  /* Now we can format the value ourselves. */
  text = g_strdup_printf ("%.2f", d);
  g_object_set (cell, "text", text, NULL);
  g_free (text);
}

void
set_up_new_columns (GtkTreeView *myview)
{
  GtkCellRendererText *renderer;
  GtkTreeViewColumn *column;
  GtkListStore *mycolumns;

  /* Create the data model and associate it with the given TreeView */
  mycolumns = gtk_list_store_new (N_COLUMNS, G_TYPE_DOUBLE);
  gtk_tree_view_set_model (myview, GTK_TREE_MODEL (mycolumns));

  /* Create a GtkCellRendererText */
  renderer = gtk_cell_renderer_text_new ();

  /* Create a new column that has a title ("Example column"),
   * uses the above created renderer that will render the double
   * value into text from the associated model's rows.
   */
```

```
  column = gtk_tree_view_column_new ();
  gtk_tree_view_column_set_title  (column, "Example column");
  renderer = gtk_cell_renderer_text_new ();
  gtk_tree_view_column_pack_start (column, renderer, TRUE);

  /* Append the new column after the GtkTreeView's previous columns. */
  gtk_tree_view_append_column (GTK_TREE_VIEW (myview), column);
  /* Since we created the column by hand, we can set it up for our
   * needs, e.g. set its minimum and maximum width, etc.
   */
  /* Set up a custom function that will be called when the column content
   * is rendered. We use the func_data pointer as an index into our
   * model. This is convenient when using multi column lists.
   */
  gtk_tree_view_column_set_cell_data_func (column, renderer,
                                           my_cell_double_to_text,
                                           (gpointer)DOUBLE_COLUMN, NULL);
}
```

---

[1] If the file load fails, gtk_image_new_from_file() will display a "broken image" graphic — to detect a failed load yourself, use gdk_pixbuf_new_from_file() directly then gtk_image_new_from_pixbuf().

<< Mailing lists and bug reports **Part II. GTK+ Core Reference >>**

GTK+ Core Reference

<< Common Questions Main loop and Events >>

Main loop and Events

Main loop and Events — Library initialization, main event loop, and events

Synopsis

```
#include <gtk/gtk.h>

gchar*      gtk_set_locale                  (void);
void        gtk_disable_setlocale           (void);
PangoLanguage* gtk_get_default_language     (void);
gboolean    gtk_parse_args                  (int *argc,
                                             char ***argv);
void        gtk_init                        (int *argc,
                                             char ***argv);
gboolean    gtk_init_check                  (int *argc,
                                             char ***argv);
gboolean    gtk_init_with_args              (int *argc,
                                             char ***argv,
                                             char *parameter_string,
                                             GOptionEntry *entries,
                                             char *translation_domain,
                                             GError **error);
GOptionGroup* gtk_get_option_group          (gboolean open_default_display);
void        gtk_exit                        (gint error_code);
gboolean    gtk_events_pending              (void);
void        gtk_main                        (void);
guint       gtk_main_level                  (void);
void        gtk_main_quit                   (void);
gboolean    gtk_main_iteration              (void);
gboolean    gtk_main_iteration_do           (gboolean blocking);
void        gtk_main_do_event               (GdkEvent *event);
void        (*GtkModuleInitFunc)            (gint *argc,
                                             gchar ***argv);
void        (*GtkModuleDisplayInitFunc)     (GdkDisplay *display);
```

```
gboolean    gtk_true                    (void);
gboolean    gtk_false                   (void);

void        gtk_grab_add                (GtkWidget *widget);
GtkWidget*  gtk_grab_get_current        (void);
void        gtk_grab_remove             (GtkWidget *widget);

void        gtk_init_add                (GtkFunction function,
                                         gpointer data);
void        gtk_quit_add_destroy        (guint main_level,
                                         GtkObject *object);
guint       gtk_quit_add                (guint main_level,
                                         GtkFunction function,
                                         gpointer data);
guint       gtk_quit_add_full           (guint main_level,
                                         GtkFunction function,
                                         GtkCallbackMarshal marshal,
                                         gpointer data,
                                         GtkDestroyNotify destroy);
void        gtk_quit_remove             (guint quit_handler_id);
void        gtk_quit_remove_by_data     (gpointer data);

guint       gtk_timeout_add_full        (guint32 interval,
                                         GtkFunction function,
                                         GtkCallbackMarshal marshal,
                                         gpointer data,
                                         GtkDestroyNotify destroy);
guint       gtk_timeout_add             (guint32 interval,
                                         GtkFunction function,
                                         gpointer data);
void        gtk_timeout_remove          (guint timeout_handler_id);

guint       gtk_idle_add                (GtkFunction function,
                                         gpointer data);
guint       gtk_idle_add_priority       (gint priority,
                                         GtkFunction function,
                                         gpointer data);
guint       gtk_idle_add_full           (gint priority,
                                         GtkFunction function,
                                         GtkCallbackMarshal marshal,
                                         gpointer data,
                                         GtkDestroyNotify destroy);
void        gtk_idle_remove             (guint idle_handler_id);
```

```
void        gtk_idle_remove_by_data         (gpointer data);

guint       gtk_input_add_full              (gint source,
                                             GdkInputCondition condition,
                                             GdkInputFunction function,
                                             GtkCallbackMarshal marshal,
                                             gpointer data,
                                             GtkDestroyNotify destroy);
void        gtk_input_remove                (guint input_handler_id);

#define     GTK_PRIORITY_REDRAW
#define     GTK_PRIORITY_RESIZE
#define     GTK_PRIORITY_HIGH
#define     GTK_PRIORITY_INTERNAL
#define     GTK_PRIORITY_DEFAULT
#define     GTK_PRIORITY_LOW

guint       gtk_key_snooper_install         (GtkKeySnoopFunc snooper,
                                             gpointer func_data);
gint        (*GtkKeySnoopFunc)              (GtkWidget *grab_widget,
                                             GdkEventKey *event,
                                             gpointer func_data);
void        gtk_key_snooper_remove          (guint snooper_handler_id);

GdkEvent*   gtk_get_current_event           (void);
guint32     gtk_get_current_event_time      (void);
gboolean    gtk_get_current_event_state     (GdkModifierType *state);
GtkWidget*  gtk_get_event_widget            (GdkEvent *event);
void        gtk_propagate_event             (GtkWidget *widget,
                                             GdkEvent *event);
```

Description

Before using GTK+, you need to initialize it; initialization connects to the window system display, and parses some standard command line arguments. The `gtk_init()` function initializes GTK+. `gtk_init()` exits the application if errors occur; to avoid this, use `gtk_init_check()`. `gtk_init_check()` allows you to recover from a failed GTK+ initialization - you might start up your application in text mode instead.

Like all GUI toolkits, GTK+ uses an event-driven programming model. When the user is doing nothing, GTK+ sits in the *main loop* and waits for input. If the user performs some action - say, a mouse click - then the main loop "wakes up" and delivers an event to GTK+. GTK+ forwards the event to one or more widgets.

When widgets receive an event, they frequently emit one or more *signals*. Signals notify your program that "something interesting happened" by invoking functions you've connected to the signal with `g_signal_connect()`. Functions connected to a signal are often termed *callbacks*.

When your callbacks are invoked, you would typically take some action - for example, when an Open button is clicked you might display a GtkFileSelectionDialog. After a callback finishes, GTK+ will return to the main loop and await more user input.

Example 1. Typical `main` function for a GTK+ application

```
int
main (int argc, char **argv)
{
  /* Initialize i18n support */
  gtk_set_locale ();

  /* Initialize the widget set */
  gtk_init (&argc, &argv);

  /* Create the main window */
  mainwin = gtk_window_new (GTK_WINDOW_TOPLEVEL);

  /* Set up our GUI elements */
  ...

  /* Show the application window */
  gtk_widget_show_all (mainwin);

  /* Enter the main event loop, and wait for user interaction */
  gtk_main ();

  /* The user lost interest */
  return 0;
}
```

It's OK to use the GLib main loop directly instead of `gtk_main()`, though it involves slightly more typing. See GMainLoop in the GLib documentation.

Details

gtk_set_locale ()

```
gchar*      gtk_set_locale                  (void);
```

Initializes internationalization support for GTK+. `gtk_init()` automatically does this, so there is typically no point in calling this function.

If you are calling this function because you changed the locale after GTK+ is was initialized, then calling this function may help a bit. (Note, however, that changing the locale after GTK+ is initialized may produce inconsistent results and is not really supported.)

In detail - sets the current locale according to the program environment. This is the same as calling the C library function `setlocale (LC_ALL, "")` but also takes care of the locale specific setup of the windowing system used by GDK.

Returns : a string corresponding to the locale set, typically in the form lang_COUNTRY, where lang is an ISO-639 language code, and COUNTRY is an ISO-3166 country code. On Unix, this form matches the result of the `setlocale()`; it is also used on other machines, such as Windows, where the C library returns a different result. The string is owned by GTK+ and should not be modified or freed.

---

gtk_disable_setlocale ()

```
void        gtk_disable_setlocale           (void);
```

Prevents `gtk_init()` and `gtk_init_check()` from automatically calling `setlocale (LC_ALL, "")`. You would want to use this function if you wanted to set the locale for your program to something other than the user's locale, or if you wanted to set different values for different locale categories.

Most programs should not need to call this function.

---

gtk_get_default_language ()

```
PangoLanguage* gtk_get_default_language     (void);
```

Returns the PangoLanguage for the default language currently in effect. (Note that this can change over the life of an application.) The default language is derived from the current locale. It determines, for example, whether GTK+ uses the right-to-left or left-to-right text direction. See `_gtk_get_lc_ctype()` for notes on behaviour on Windows.

Returns : the default language as a PangoLanguage, must not be freed

gtk_parse_args ()

```
gboolean    gtk_parse_args                  (int *argc,
                                             char ***argv);
```

Parses command line arguments, and initializes global attributes of GTK+, but does not actually open a connection to a display. (See gdk_display_open(), gdk_get_display_arg_name())

Any arguments used by GTK+ or GDK are removed from the array and *argc* and *argv* are updated accordingly.

You shouldn't call this function explicitely if you are using gtk_init(), or gtk_init_check().

argc : a pointer to the number of command line arguments.
argv : a pointer to the array of command line arguments.
Returns : TRUE if initialization succeeded, otherwise FALSE.

gtk_init ()

```
void        gtk_init                        (int *argc,
                                             char ***argv);
```

Call this function before using any other GTK+ functions in your GUI applications. It will initialize everything needed to operate the toolkit and parses some standard command line options. *argc* and *argv* are adjusted accordingly so your own code will never see those standard arguments.

Note

This function will terminate your program if it was unable to initialize the GUI for some reason. If you want your program to fall back to a textual interface you want to call gtk_init_check() instead.

Note

argc : Address of the *argc* parameter of your `main()` function. Changed if any arguments were handled.
argv : Address of the *argv* parameter of `main()`. Any parameters understood by gtk_init() are stripped before return.

gtk_init_check ()

```
gboolean    gtk_init_check                  (int *argc,
                                             char ***argv);
```

This function does the same work as `gtk_init()` with only a single change: It does not terminate the program if the GUI can't be initialized. Instead it returns FALSE on failure.

This way the application can fall back to some other means of communication with the user - for example a curses or command line interface.

argc : Address of the *argc* parameter of your `main()` function. Changed if any arguments were handled.

argv : Address of the *argv* parameter of `main()`. Any parameters understood by `gtk_init()` are stripped before return.

Returns : TRUE if the GUI has been successfully initialized, FALSE otherwise.

---

gtk_init_with_args ()

```
gboolean    gtk_init_with_args              (int *argc,
                                             char ***argv,
                                             char *parameter_string,
                                             GOptionEntry *entries,
                                             char *translation_domain,
                                             GError **error);
```

argc :

argv :

parameter_string :

entries :

translation_domain :

error :

Returns :

---

gtk_get_option_group ()

```
GOptionGroup* gtk_get_option_group          (gboolean open_default_display);
```

open_default_display :

Returns :

---

gtk_exit ()

```
void        gtk_exit                        (gint error_code);
```

Warning

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

Terminates the program and returns the given exit code to the caller. This function will shut down the GUI and free all resources allocated for GTK+.

error_code : Return value to pass to the caller. This is dependend on the target system but at least on Unix systems `0` means success.

---

gtk_events_pending ()

```
gboolean    gtk_events_pending              (void);
```

Checks if any events are pending. This can be used to update the GUI and invoke timeouts etc. while doing some time intensive computation.

Example 2. Updating the GUI during a long computation.

```
        /* computation going on */
...
        while (gtk_events_pending ())
          gtk_main_iteration ();
...
        /* computation continued */
```

Returns : TRUE if any events are pending, FALSE otherwise.

---

gtk_main ()

```
void        gtk_main                            (void);
```

Runs the main loop until `gtk_main_quit()` is called. You can nest calls to `gtk_main()`. In that case `gtk_main_quit()` will make the innermost invocation of the main loop return.

---

gtk_main_level ()

```
guint       gtk_main_level                      (void);
```

Asks for the current nesting level of the main loop. This can be useful when calling `gtk_quit_add()`.

Returns : the nesting level of the current invocation of the main loop.

---

gtk_main_quit ()

```
void        gtk_main_quit                       (void);
```

Makes the innermost invocation of the main loop return when it regains control.

---

gtk_main_iteration ()

```
gboolean    gtk_main_iteration                  (void);
```

Runs a single iteration of the mainloop. If no events are waiting to be processed GTK+ will block until the next event is noticed. If you don't want to block look at `gtk_main_iteration_do()` or check if any events are pending with `gtk_events_pending()` first.

Returns : TRUE if `gtk_main_quit()` has been called for the innermost mainloop.

---

gtk_main_iteration_do ()

```
gboolean    gtk_main_iteration_do           (gboolean blocking);
```

Runs a single iteration of the mainloop. If no events are available either return or block dependent on the value of *`blocking`*.

`blocking` : TRUE if you want GTK+ to block if no events are pending.

Returns : TRUE if `gtk_main_quit()` has been called for the innermost mainloop.

---

gtk_main_do_event ()

```
void        gtk_main_do_event               (GdkEvent *event);
```

Processes a single GDK event. This is public only to allow filtering of events between GDK and GTK+. You will not usually need to call this function directly.

While you should not call this function directly, you might want to know how exactly events are handled. So here is what this function does with the event:

1. Compress enter/leave notify events. If the event passed build an enter/leave pair together with the next event (peeked from GDK) both events are thrown away. This is to avoid a backlog of (de-)highlighting widgets crossed by the pointer.
2. Find the widget which got the event. If the widget can't be determined the event is thrown away unless it belongs to a INCR transaction. In that case it is passed to `gtk_selection_incr_event()`.
3. Then the event is passed on a stack so you can query the currently handled event with `gtk_get_current_event()`.
4. The event is sent to a widget. If a grab is active all events for widgets that are not in the contained in the grab widget are sent to the latter with a few exceptions:
   - Deletion and destruction events are still sent to the event widget for obvious reasons.
   - Events which directly relate to the visual representation of the event widget.
   - Leave events are delivered to the event widget if there was an enter event delivered to it before without the paired leave event.
   - Drag events are not redirected because it is unclear what the semantics of that would be.

Another point of interest might be that all key events are first passed through the key snooper functions if there are any. Read the description of `gtk_key_snooper_install()` if you need this feature.

5. After finishing the delivery the event is popped from the event stack.

`event` : An event to process (normally) passed by GDK.

GtkModuleInitFunc ()

```
void        (*GtkModuleInitFunc)            (gint *argc,
                                             gchar ***argv);
```

Each GTK+ module must have a function `gtk_module_init()` with this prototype. This function is called after loading the module with the *`argc`* and *`argv`* cleaned from any arguments that GTK+ handles itself.

`argc` : Pointer to the number of arguments remaining after `gtk_init()`.

`argv` : Points to the argument vector.

GtkModuleDisplayInitFunc ()

```
void        (*GtkModuleDisplayInitFunc)     (GdkDisplay *display);
```

`display` :

Since 2.2

---

gtk_true ()

```
gboolean    gtk_true                        (void);
```

All this function does it to return TRUE. This can be useful for example if you want to inhibit the deletion of a window. Of course you should not do this as the user expects a reaction from clicking the close icon of the window...

Example 3. A persistent window

```
#include <gtk/gtk.h>

int
main (int argc, char **argv)
{
  GtkWidget     *win, *but;

  gtk_init( &argc, &argv );

  win = gtk_window_new (GTK_WINDOW_TOPLEVEL);
  g_signal_connect (win, "delete-event",
                    G_CALLBACK (gtk_true), NULL);
  g_signal_connect (win, "destroy",
                    G_CALLBACK (gtk_main_quit), NULL);

  but = gtk_button_new_with_label ("Close yourself. I mean it!");
  g_signal_connect_swapped (but, "clicked",
                 G_CALLBACK (gtk_object_destroy), win);
  gtk_container_add (GTK_CONTAINER (win), but);

  gtk_widget_show_all (win);
  gtk_main ();
  return 0;
}
```

Returns : TRUE

---

gtk_false ()

```
gboolean    gtk_false                           (void);
```

Analogical to `gtk_true()` this function does nothing but always returns FALSE.

Returns : FALSE

---

gtk_grab_add ()

```
void        gtk_grab_add                        (GtkWidget *widget);
```

Makes *widget* the current grabbed widget. This means that interaction with other widgets in the same application is blocked and mouse as well as keyboard events are delivered to this widget.

widget : The widget that grabs keyboard and pointer events.

---

gtk_grab_get_current ()

```
GtkWidget*  gtk_grab_get_current            (void);
```

Queries the current grab.

Returns : The widget which currently has the grab or NULL if no grab is active.

---

gtk_grab_remove ()

```
void        gtk_grab_remove                     (GtkWidget *widget);
```

Removes the grab from the given widget. You have to pair calls to gtk_grab_add() and gtk_grab_remove().

widget : The widget which gives up the grab.

---

gtk_init_add ()

```
void        gtk_init_add                        (GtkFunction function,
                                                 gpointer data);
```

Registers a function to be called when the mainloop is started.

function : Function to invoke when gtk_main() is called next.

data : Data to pass to that function.

gtk_quit_add_destroy ()

```
void        gtk_quit_add_destroy            (guint main_level,
                                             GtkObject *object);
```

Trigger destruction of *object* in case the mainloop at level *main_level* is quit.

main_level : Level of the mainloop which shall trigger the destruction.
object : Object to be destroyed.

gtk_quit_add ()

```
guint       gtk_quit_add                    (guint main_level,
                                             GtkFunction function,
                                             gpointer data);
```

Registers a function to be called when an instance of the mainloop is left.

main_level : Level at which termination the function shall be called. You can pass 0 here to have the function run at the termination of the current mainloop.
function : The function to call. This should return 0 to be removed from the list of quit handlers. Otherwise the function might be called again.
data : Pointer to pass when calling *function*.
Returns : A handle for this quit handler (you need this for gtk_quit_remove()) or 0 if you passed a NULL pointer in *function*.

gtk_quit_add_full ()

```
guint       gtk_quit_add_full               (guint main_level,
                                             GtkFunction function,
                                             GtkCallbackMarshal marshal,
                                             gpointer data,
                                             GtkDestroyNotify destroy);
```

Registers a function to be called when an instance of the mainloop is left. In comparison to gtk_quit_add() this function adds the possibility to pass a marshaller and a function to be called when the quit handler is freed.

The former can be used to run interpreted code instead of a compiled function while the latter can be used to free the information stored in *data* (while you can do this in *function* as well)... So this function will mostly be used by GTK+ wrappers for languages other than C.

main_level :	Level at which termination the function shall be called. You can pass 0 here to have the function run at the termination of the current mainloop.
function :	The function to call. This should return 0 to be removed from the list of quit handlers. Otherwise the function might be called again.
marshal :	The marshaller to be used. If this is non-NULL, *function* is ignored.
data :	Pointer to pass when calling *function*.
destroy :	Function to call to destruct *data*. Gets *data* as argument.
Returns :	A handle for this quit handler (you need this for gtk_quit_remove()) or 0 if you passed a NULL pointer in *function*.

---

gtk_quit_remove ()

```
void        gtk_quit_remove                     (guint quit_handler_id);
```

Removes a quit handler by its identifier.

quit_handler_id : Identifier for the handler returned when installing it.

---

gtk_quit_remove_by_data ()

```
void        gtk_quit_remove_by_data             (gpointer data);
```

Removes a quit handler identified by its *data* field.

data : The pointer passed as *data* to gtk_quit_add() or gtk_quit_add_full().

---

gtk_timeout_add_full ()

```
guint       gtk_timeout_add_full            (guint32 interval,
                                             GtkFunction function,
                                             GtkCallbackMarshal marshal,
                                             gpointer data,
                                             GtkDestroyNotify destroy);
```

Warning

`gtk_timeout_add_full` is deprecated and should not be used in newly-written code. Use `g_timeout_add_full()` instead.

Registers a function to be called periodically. The function will be called repeatedly after *`interval`* milliseconds until it returns FALSE at which point the timeout is destroyed and will not be called again.

`interval` : The time between calls to the function, in milliseconds (1/1000ths of a second.)
`function` : The function to call periodically.
`marshal` : The marshaller to use instead of the function (if non-`NULL`).
`data` : The data to pass to the function.
`destroy` : Function to call when the timeout is destroyed or `NULL`.
Returns : A unique id for the event source.

---

gtk_timeout_add ()

```
guint       gtk_timeout_add                 (guint32 interval,
                                             GtkFunction function,
                                             gpointer data);
```

Warning

`gtk_timeout_add` is deprecated and should not be used in newly-written code. Use `g_timeout_add()` instead.

Registers a function to be called periodically. The function will be called repeatedly after *`interval`* milliseconds until it returns FALSE at which point the timeout is destroyed and will not be called again.

`interval` : The time between calls to the function, in milliseconds (1/1000ths of a second.)
`function` : The function to call periodically.

data : The data to pass to the function.

Returns : A unique id for the event source.

---

gtk_timeout_remove ()

```
void        gtk_timeout_remove              (guint timeout_handler_id);
```

Warning

`gtk_timeout_remove` is deprecated and should not be used in newly-written code. Use `g_source_remove()` instead.

Removes the given timeout destroying all information about it.

timeout_handler_id : The identifier returned when installing the timeout.

---

gtk_idle_add ()

```
guint       gtk_idle_add                    (GtkFunction function,
                                             gpointer data);
```

Warning

`gtk_idle_add` is deprecated and should not be used in newly-written code. Use `g_idle_add()` instead.

Causes the mainloop to call the given function whenever no events with higher priority are to be processed. The default priority is `GTK_PRIORITY_DEFAULT`, which is rather low.

function : The function to call.

data : The information to pass to the function.

Returns : a unique handle for this registration.

---

gtk_idle_add_priority ()

```
guint       gtk_idle_add_priority           (gint priority,
                                             GtkFunction function,
                                             gpointer data);
```

Warning

`gtk_idle_add_priority` is deprecated and should not be used in newly-written code. Use `g_idle_add_full()` instead.

Like `gtk_idle_add()` this function allows you to have a function called when the event loop is idle. The difference is that you can give a priority different from `GTK_PRIORITY_DEFAULT` to the idle function.

priority : The priority which should not be above `G_PRIORITY_HIGH_IDLE`. Note that you will interfere with GTK+ if you use a priority above `GTK_PRIORITY_RESIZE`.
function : The function to call.
data : Data to pass to that function.
Returns : A unique id for the event source.

---

gtk_idle_add_full ()

```
guint       gtk_idle_add_full               (gint priority,
                                             GtkFunction function,
                                             GtkCallbackMarshal marshal,
                                             gpointer data,
                                             GtkDestroyNotify destroy);
```

Warning

`gtk_idle_add_full` is deprecated and should not be used in newly-written code. Use `g_idle_add_full()` instead.

Like `gtk_idle_add()` this function allows you to have a function called when the event loop is idle. The difference is that you can give a priority different from `GTK_PRIORITY_DEFAULT` to the idle function.

priority : The priority which should not be above `G_PRIORITY_HIGH_IDLE`. Note that you will interfere with GTK+ if you use a priority above `GTK_PRIORITY_RESIZE`.
function : The function to call.
marshal : The marshaller to use instead of the function (if non-`NULL`).

data : Data to pass to that function.
destroy : Function to call when the timeout is destroyed or NULL.
Returns : A unique id for the event source.

---

gtk_idle_remove ()

```
void        gtk_idle_remove                 (guint idle_handler_id);
```

Warning

`gtk_idle_remove` is deprecated and should not be used in newly-written code. Use `g_source_remove()` instead.

Removes the idle function with the given id.

idle_handler_id : Identifies the idle function to remove.

---

gtk_idle_remove_by_data ()

```
void        gtk_idle_remove_by_data         (gpointer data);
```

Warning

`gtk_idle_remove_by_data` is deprecated and should not be used in newly-written code. Use `g_idle_remove_by_data()` instead.

Removes the idle function identified by the user data.

data : remove the idle function which was registered with this user data.

---

gtk_input_add_full ()

```
guint       gtk_input_add_full              (gint source,
                                             GdkInputCondition condition,
```

```
                                                GdkInputFunction function,
                                                GtkCallbackMarshal marshal,
                                                gpointer data,
                                                GtkDestroyNotify destroy);
```

Warning

`gtk_input_add_full` is deprecated and should not be used in newly-written code. Use `g_io_add_watch_full()` instead.

Registers a function to be called when a condition becomes true on a file descriptor.

source :	a file descriptor.
condition :	the condition.
function :	The function to call.
marshal :	The marshaller to use instead of the function (if non-`NULL`).
data :	callback data passed to *function*.
destroy :	callback function to call with *data* when the input handler is removed, or `NULL`.
Returns :	A unique id for the event source; to be used with `gtk_input_remove()`.

---

gtk_input_remove ()

```
void        gtk_input_remove                (guint input_handler_id);
```

Warning

`gtk_input_remove` is deprecated and should not be used in newly-written code. Use `g_source_remove()` instead.

Removes the function with the given id.

input_handler_id : Identifies the function to remove.

GTK_PRIORITY_REDRAW

```
#define GTK_PRIORITY_REDRAW     (G_PRIORITY_HIGH_IDLE + 20)
```

Warning

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

Use this priority for redrawing related stuff. It is used internally by GTK+ to do pending redraws. This priority is lower than `GTK_PRIORITY_RESIZE` to avoid redrawing a widget just before resizing (and therefore redrawing it again).

Note

This macro is deprecated. You should use `GDK_PRIORITY_REDRAW` instead.

GTK_PRIORITY_RESIZE

```
#define GTK_PRIORITY_RESIZE     (G_PRIORITY_HIGH_IDLE + 10)
```

Use this priority for resizing related stuff. It is used internally by GTK+ to compute the sizes of widgets. This priority is higher than `GTK_PRIORITY_REDRAW` to avoid resizing a widget which was just redrawn.

GTK_PRIORITY_HIGH

```
#define GTK_PRIORITY_HIGH       G_PRIORITY_HIGH
```

Warning

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

Use this for high priority timeouts. This priority is never used inside GTK+ so everything running at this priority will be running before anything inside the toolkit.

Note

This macro is deprecated. You should use `G_PRIORITY_HIGH` instead.

GTK_PRIORITY_INTERNAL

```
#define GTK_PRIORITY_INTERNAL   GTK_PRIORITY_REDRAW
```

Warning

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

This priority is for GTK+ internal stuff. Don't use it in your applications.

GTK_PRIORITY_DEFAULT

```
#define GTK_PRIORITY_DEFAULT    G_PRIORITY_DEFAULT_IDLE
```

Warning

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

Default priority for idle functions.

Note

This macro is deprecated. You should use `G_PRIORITY_DEFAULT_IDLE` instead.

GTK_PRIORITY_LOW

```
#define GTK_PRIORITY_LOW        G_PRIORITY_LOW
```

Warning

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

Priority for very unimportant background tasks.

Note

This macro is deprecated. You should use `G_PRIORITY_LOW` instead.

---

gtk_key_snooper_install ()

```
guint       gtk_key_snooper_install         (GtkKeySnoopFunc snooper,
                                             gpointer func_data);
```

Installs a key snooper function, which will get called on all key events before delivering them normally.

snooper : a GtkKeySnoopFunc.
func_data : data to pass to *snooper*.
Returns : a unique id for this key snooper for use with `gtk_key_snooper_remove()`.

GtkKeySnoopFunc ()

```
gint        (*GtkKeySnoopFunc)              (GtkWidget *grab_widget,
                                             GdkEventKey *event,
                                             gpointer func_data);
```

Key snooper functions are called before normal event delivery. They can be used to implement custom key event handling.

grab_widget : the widget to which the event will be delivered.
event : the key event.
func_data : the *func_data* supplied to `gtk_key_snooper_install()`.
Returns : TRUE to stop further processing of *event*, FALSE to continue.

---

gtk_key_snooper_remove ()

```
void        gtk_key_snooper_remove          (guint snooper_handler_id);
```

Removes the key snooper function with the given id.

snooper_handler_id : Identifies the key snooper to remove.

---

gtk_get_current_event ()

```
GdkEvent*   gtk_get_current_event           (void);
```

Obtains a copy of the event currently being processed by GTK+. For example, if you get a "clicked" signal from GtkButton, the current event will be the GdkEventButton that triggered the "clicked" signal. The returned event must be freed with `gdk_event_free()`. If there is no current event, the function returns `NULL`.

Returns : a copy of the current event, or `NULL` if no current event.

---

gtk_get_current_event_time ()

```
guint32     gtk_get_current_event_time      (void);
```

If there is a current event and it has a timestamp, return that timestamp, otherwise return `GDK_CURRENT_TIME`.

Returns : the timestamp from the current event, or `GDK_CURRENT_TIME`.

---

gtk_get_current_event_state ()

```
gboolean    gtk_get_current_event_state     (GdkModifierType *state);
```

If there is a current event and it has a state field, place that state field in *state* and return `TRUE`, otherwise return `FALSE`.

state : a location to store the state of the current event

Returns : `TRUE` if there was a current event and it had a state field

---

gtk_get_event_widget ()

```
GtkWidget*  gtk_get_event_widget            (GdkEvent *event);
```

If *event* is NULL or the event was not associated with any widget, returns NULL, otherwise returns the widget that received the event originally.

event : a GdkEvent

Returns : the widget that originally received *event*, or NULL

---

gtk_propagate_event ()

```
void        gtk_propagate_event             (GtkWidget *widget,
                                             GdkEvent *event);
```

Sends an event to a widget, propagating the event to parent widgets if the event remains unhandled. Events received by GTK+ from GDK normally begin in gtk_main_do_event(). Depending on the type of event, existence of modal dialogs, grabs, etc., the event may be propagated; if so, this function is used. gtk_propagate_event() calls gtk_widget_event() on each widget it decides to send the event to. So gtk_widget_event() is the lowest-level function; it simply emits the "event" and possibly an event-specific signal on a widget. gtk_propagate_event() is a bit higher-level, and gtk_main_do_event() is the highest level.

All that said, you most likely don't want to use any of these functions; synthesizing events is rarely needed. Consider asking on the mailing list for better ways to achieve your goals. For example, use gdk_window_invalidate_rect() or gtk_widget_queue_draw() instead of making up expose events.

widget : a GtkWidget

event : an event

See Also

See the GLib manual, especially GMainLoop and signal-related functions such as g_signal_connect().

Accelerator Groups

Accelerator Groups — Groups of global keyboard accelerators for an entire GtkWindow

Synopsis

```
#include <gtk/gtk.h>

            GtkAccelGroup;
GtkAccelGroup* gtk_accel_group_new          (void);
#define     gtk_accel_group_ref
#define     gtk_accel_group_unref
void        gtk_accel_group_connect         (GtkAccelGroup *accel_group,
                                             guint accel_key,
                                             GdkModifierType accel_mods,
                                             GtkAccelFlags accel_flags,
                                             GClosure *closure);
void        gtk_accel_group_connect_by_path (GtkAccelGroup *accel_group,
                                             const gchar *accel_path,
                                             GClosure *closure);
gboolean    (*GtkAccelGroupActivate)        (GtkAccelGroup *accel_group,
                                             GObject *acceleratable,
                                             guint keyval,
                                             GdkModifierType modifier);
gboolean    (*GtkAccelGroupFindFunc)        (GtkAccelKey *key,
                                             GClosure *closure,
                                             gpointer data);
gboolean    gtk_accel_group_disconnect      (GtkAccelGroup *accel_group,
                                             GClosure *closure);
gboolean    gtk_accel_group_disconnect_key  (GtkAccelGroup *accel_group,
                                             guint accel_key,
                                             GdkModifierType accel_mods);
GtkAccelGroupEntry* gtk_accel_group_query   (GtkAccelGroup *accel_group,
                                             guint accel_key,
                                             GdkModifierType accel_mods,
                                             guint *n_entries);
gboolean    gtk_accel_group_activate        (GtkAccelGroup *accel_group,
```

```
                                                GQuark accel_quark,
                                                GObject *acceleratable,
                                                guint accel_key,
                                                GdkModifierType accel_mods);
void        gtk_accel_group_lock                (GtkAccelGroup *accel_group);
void        gtk_accel_group_unlock              (GtkAccelGroup *accel_group);
GtkAccelGroup* gtk_accel_group_from_accel_closure
                                                (GClosure *closure);
gboolean    gtk_accel_groups_activate           (GObject *object,
                                                guint accel_key,
                                                GdkModifierType accel_mods);
GSList*     gtk_accel_groups_from_object        (GObject *object);
GtkAccelKey* gtk_accel_group_find               (GtkAccelGroup *accel_group,
                                                GtkAccelGroupFindFunc find_func,
                                                gpointer data);
            GtkAccelKey;
gboolean    gtk_accelerator_valid               (guint keyval,
                                                GdkModifierType modifiers);
void        gtk_accelerator_parse               (const gchar *accelerator,
                                                guint *accelerator_key,
                                                GdkModifierType *accelerator_mods);
gchar*      gtk_accelerator_name                (guint accelerator_key,
                                                GdkModifierType accelerator_mods);
gchar*      gtk_accelerator_get_label           (guint accelerator_key,
                                                GdkModifierType accelerator_mods);
void        gtk_accelerator_set_default_mod_mask
                                                (GdkModifierType default_mod_mask);
guint       gtk_accelerator_get_default_mod_mask
                                                (void);
```

Object Hierarchy

```
  GObject
   +----GtkAccelGroup
```

Signal Prototypes

```
"accel-activate"
```

```
            gboolean    user_function      (GtkAccelGroup *accelgroup,
                                            GObject *arg1,
                                            guint arg2,
                                            GdkModifierType arg3,
                                            gpointer user_data);
"accel-changed"
            void        user_function      (GtkAccelGroup *accelgroup,
                                            guint arg1,
                                            GdkModifierType arg2,
                                            GClosure *arg3,
                                            gpointer user_data);
```

Description

A GtkAccelGroup represents a group of keyboard accelerators, typically attached to a toplevel GtkWindow (with `gtk_window_add_accel_group()`). Usually you won't need to create a GtkAccelGroup directly; instead, when using GtkItemFactory, GTK+ automatically sets up the accelerators for your menus in the item factory's GtkAccelGroup.

Note that *accelerators* are different from *mnemonics*. Accelerators are shortcuts for activating a menu item; they appear alongside the menu item they're a shortcut for. For example "Ctrl+Q" might appear alongside the "Quit" menu item. Mnemonics are shortcuts for GUI elements such as text entries or buttons; they appear as underlined characters. See `gtk_label_new_with_mnemonic()`. Menu items can have both accelerators and mnemonics, of course.

Details

GtkAccelGroup

```
typedef struct _GtkAccelGroup GtkAccelGroup;
```

An object representing and maintaining a group of accelerators.

---

gtk_accel_group_new ()

```
GtkAccelGroup* gtk_accel_group_new          (void);
```

Creates a new GtkAccelGroup.

Returns : a new GtkAccelGroup object

gtk_accel_group_ref

```
#define gtk_accel_group_ref     g_object_ref
```

Warning

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

Deprecated equivalent of `g_object_ref()`.

Returns :

---

gtk_accel_group_unref

```
#define gtk_accel_group_unref   g_object_unref
```

Warning

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

Deprecated equivalent of `g_object_unref()`.

---

gtk_accel_group_connect ()

```
void        gtk_accel_group_connect         (GtkAccelGroup *accel_group,
                                             guint accel_key,
                                             GdkModifierType accel_mods,
                                             GtkAccelFlags accel_flags,
                                             GClosure *closure);
```

Installs an accelerator in this group. When *`accel_group`* is being activated in response to a call to `gtk_accel_groups_activate()`, *`closure`* will be invoked if the *`accel_key`* and *`accel_mods`* from `gtk_accel_groups_activate()` match those of this connection.

The signature used for the *`closure`* is that of GtkAccelGroupActivate.

Note that, due to implementation details, a single closure can only be connected to one accelerator group.

accel_group : the accelerator group to install an accelerator in
accel_key : key value of the accelerator
accel_mods : modifier combination of the accelerator
accel_flags : a flag mask to configure this accelerator
closure : closure to be executed upon accelerator activation

---

gtk_accel_group_connect_by_path ()

```
void        gtk_accel_group_connect_by_path (GtkAccelGroup *accel_group,
                                             const gchar *accel_path,
                                             GClosure *closure);
```

Installs an accelerator in this group, using an accelerator path to look up the appropriate key and modifiers (see gtk_accel_map_add_entry()). When *accel_group* is being activated in response to a call to gtk_accel_groups_activate(), *closure* will be invoked if the *accel_key* and *accel_mods* from gtk_accel_groups_activate() match the key and modifiers for the path.

The signature used for the *closure* is that of GtkAccelGroupActivate.

accel_group : the accelerator group to install an accelerator in
accel_path : path used for determining key and modifiers.
closure : closure to be executed upon accelerator activation

GtkAccelGroupActivate ()

```
gboolean    (*GtkAccelGroupActivate)        (GtkAccelGroup *accel_group,
                                             GObject *acceleratable,
                                             guint keyval,
                                             GdkModifierType modifier);
```

accel_group :
acceleratable :
keyval :
modifier :
Returns :

GtkAccelGroupFindFunc ()

```
gboolean    (*GtkAccelGroupFindFunc)        (GtkAccelKey *key,
                                             GClosure *closure,
                                             gpointer data);
```

key :
closure :
data :
Returns :

Since 2.2

---

gtk_accel_group_disconnect ()

```
gboolean    gtk_accel_group_disconnect      (GtkAccelGroup *accel_group,
                                             GClosure *closure);
```

Removes an accelerator previously installed through gtk_accel_group_connect().

accel_group : the accelerator group to remove an accelerator from
closure : the closure to remove from this accelerator group
Returns : TRUE if the closure was found and got disconnected

---

gtk_accel_group_disconnect_key ()

```
gboolean    gtk_accel_group_disconnect_key  (GtkAccelGroup *accel_group,
                                             guint accel_key,
                                             GdkModifierType accel_mods);
```

Removes an accelerator previously installed through gtk_accel_group_connect().

accel_group : the accelerator group to install an accelerator in
accel_key : key value of the accelerator

accel_mods : modifier combination of the accelerator

Returns : TRUE if there was an accelerator which could be removed, FALSE otherwise

---

gtk_accel_group_query ()

```
GtkAccelGroupEntry* gtk_accel_group_query   (GtkAccelGroup *accel_group,
                                             guint accel_key,
                                             GdkModifierType accel_mods,
                                             guint *n_entries);
```

Queries an accelerator group for all entries matching *accel_key* and *accel_mods*.

accel_group : the accelerator group to query

accel_key : key value of the accelerator

accel_mods : modifier combination of the accelerator

n_entries : location to return the number of entries found, or NULL

Returns : an array of *n_entries* GtkAccelGroupEntry elements, or NULL. The array is owned by GTK+ and must not be freed.

---

gtk_accel_group_activate ()

```
gboolean    gtk_accel_group_activate        (GtkAccelGroup *accel_group,
                                             GQuark accel_quark,
                                             GObject *acceleratable,
                                             guint accel_key,
                                             GdkModifierType accel_mods);
```

accel_group :

accel_quark :

acceleratable :

accel_key :

accel_mods :

Returns :

---

gtk_accel_group_lock ()

```
void        gtk_accel_group_lock            (GtkAccelGroup *accel_group);
```

Locks the given accelerator group.

Locking an acelerator group prevents the accelerators contained within it to be changed during runtime. Refer to `gtk_accel_map_change_entry()` about runtime accelerator changes.

If called more than once, *`accel_group`* remains locked until `gtk_accel_group_unlock()` has been called an equivalent number of times.

`accel_group` : a GtkAccelGroup

---

gtk_accel_group_unlock ()

```
void        gtk_accel_group_unlock          (GtkAccelGroup *accel_group);
```

Undoes the last call to `gtk_accel_group_lock()` on this *`accel_group`*.

`accel_group` : a GtkAccelGroup

---

gtk_accel_group_from_accel_closure ()

```
GtkAccelGroup* gtk_accel_group_from_accel_closure
                                            (GClosure *closure);
```

Finds the GtkAccelGroup to which *`closure`* is connected; see `gtk_accel_group_connect()`.

`closure` : a GClosure

Returns : the GtkAccelGroup to which *`closure`* is connected, or `NULL`.

---

gtk_accel_groups_activate ()

```
gboolean    gtk_accel_groups_activate       (GObject *object,
                                             guint accel_key,
                                             GdkModifierType accel_mods);
```

Finds the first accelerator in any GtkAccelGroup attached to *object* that matches *accel_key* and *accel_mods*, and activates that accelerator. If an accelerator was activated and handled this keypress, TRUE is returned.

object : the GObject, usually a GtkWindow, on which to activate the accelerator.
accel_key : accelerator keyval from a key event
accel_mods : keyboard state mask from a key event
Returns : TRUE if the accelerator was handled, FALSE otherwise

---

gtk_accel_groups_from_object ()

```
GSList*     gtk_accel_groups_from_object    (GObject *object);
```

Gets a list of all accel groups which are attached to *object*.

object : a GObject, usually a GtkWindow
Returns : a list of all accel groups which are attached to *object*

---

gtk_accel_group_find ()

```
GtkAccelKey* gtk_accel_group_find           (GtkAccelGroup *accel_group,
                                             GtkAccelGroupFindFunc find_func,
                                             gpointer data);
```

Finds the first entry in an accelerator group for which *find_func* returns TRUE and returns its GtkAccelKey.

accel_group : a GtkAccelGroup
find_func : a function to filter the entries of *accel_group* with
data : data to pass to *find_func*
Returns : the key of the first entry passing *find_func*. The key is owned by GTK+ and must not be freed.

GtkAccelKey

```
typedef struct {
  guint           accel_key;
```

```
  GdkModifierType accel_mods;
  guint           accel_flags : 16;
} GtkAccelKey;
```

---

gtk_accelerator_valid ()

```
gboolean    gtk_accelerator_valid           (guint keyval,
                                             GdkModifierType modifiers);
```

Determines whether a given keyval and modifier mask constitute a valid keyboard accelerator. For example, the GDK_a keyval plus GDK_CONTROL_MASK is valid - this is a "Ctrl+a" accelerator. But, you can't, for instance, use the GDK_Control_L keyval as an accelerator.

keyval : a GDK keyval
modifiers : modifier mask
Returns : TRUE if the accelerator is valid

---

gtk_accelerator_parse ()

```
void        gtk_accelerator_parse           (const gchar *accelerator,
                                             guint *accelerator_key,
                                             GdkModifierType *accelerator_mods);
```

Parses a string representing an accelerator. The format looks like "<Control>a" or "<Shift><Alt>F1" or "<Release>z" (the last one is for key release). The parser is fairly liberal and allows lower or upper case, and also abbreviations such as "<Ctl>" and "<Ctrl>".

If the parse fails, *accelerator_key* and *accelerator_mods* will be set to 0 (zero).

accelerator : string representing an accelerator
accelerator_key : return location for accelerator keyval
accelerator_mods : return location for accelerator modifier mask

---

gtk_accelerator_name ()

```
gchar*      gtk_accelerator_name            (guint accelerator_key,
```

```
                                                GdkModifierType accelerator_mods);
```

Converts an accelerator keyval and modifier mask into a string parseable by gtk_accelerator_parse(). For example, if you pass in GDK_q and GDK_CONTROL_MASK, this function returns "<Control>q".

If you need to display accelerators in the user interface, see gtk_accelerator_get_label().

accelerator_key : accelerator keyval
accelerator_mods : accelerator modifier mask
Returns : a newly-allocated accelerator name

---

gtk_accelerator_get_label ()

```
gchar*      gtk_accelerator_get_label       (guint accelerator_key,
                                             GdkModifierType accelerator_mods);
```

Converts an accelerator keyval and modifier mask into a string which can be used to represent the accelerator to the user.

accelerator_key : accelerator keyval
accelerator_mods : accelerator modifier mask
Returns : a newly-allocated string representing the accelerator.

Since 2.6

---

gtk_accelerator_set_default_mod_mask ()

```
void        gtk_accelerator_set_default_mod_mask
                                            (GdkModifierType default_mod_mask);
```

Sets the modifiers that will be considered significant for keyboard accelerators. The default mod mask is GDK_CONTROL_MASK | GDK_SHIFT_MASK | GDK_MOD1_MASK, that is, Control, Shift, and Alt. Other modifiers will by default be ignored by GtkAccelGroup. You must include at least the three default modifiers in any value you pass to this function.

The default mod mask should be changed on application startup, before using any accelerator groups.

default_mod_mask : accelerator modifier mask

gtk_accelerator_get_default_mod_mask ()

```
guint       gtk_accelerator_get_default_mod_mask
                                            (void);
```

Gets the value set by `gtk_accelerator_set_default_mod_mask()`.

Returns : the default accelerator modifier mask

Signals

The "accel-activate" signal

```
gboolean    user_function                   (GtkAccelGroup *accelgroup,
                                             GObject *arg1,
                                             guint arg2,
                                             GdkModifierType arg3,
                                             gpointer user_data);
```

accelgroup : the object which received the signal.
arg1 :
arg2 :
arg3 :
user_data : user data set when the signal handler was connected.
Returns :

The "accel-changed" signal

```
void        user_function                   (GtkAccelGroup *accelgroup,
                                             guint arg1,
                                             GdkModifierType arg2,
                                             GClosure *arg3,
                                             gpointer user_data);
```

accelgroup : the object which received the signal.

arg1 :
arg2 :
arg3 :
user_data : user data set when the signal handler was connected.

See Also

`gtk_window_add_accel_group()`, `gtk_accel_map_change_entry()`, `gtk_item_factory_new()`, `gtk_label_new_with_mnemonic()`

<< Main loop and Events **Accelerator Maps >>**

Accelerator Maps

Accelerator Maps — Loadable keyboard accelerator specifications

Synopsis

```
#include <gtk/gtk.h>

            GtkAccelMap;
void        (*GtkAccelMapForeach)           (gpointer data,
                                             const gchar *accel_path,
                                             guint accel_key,
                                             GdkModifierType accel_mods,
                                             gboolean changed);
void        gtk_accel_map_add_entry         (const gchar *accel_path,
                                             guint accel_key,
                                             GdkModifierType accel_mods);
gboolean    gtk_accel_map_lookup_entry      (const gchar *accel_path,
                                             GtkAccelKey *key);
gboolean    gtk_accel_map_change_entry      (const gchar *accel_path,
                                             guint accel_key,
                                             GdkModifierType accel_mods,
                                             gboolean replace);
void        gtk_accel_map_load              (const gchar *file_name);
void        gtk_accel_map_save              (const gchar *file_name);
void        gtk_accel_map_foreach           (gpointer data,
                                             GtkAccelMapForeach foreach_func);
void        gtk_accel_map_load_fd           (gint fd);
void        gtk_accel_map_save_fd           (gint fd);
void        gtk_accel_map_load_scanner      (GScanner *scanner);
void        gtk_accel_map_add_filter        (const gchar *filter_pattern);
void        gtk_accel_map_foreach_unfiltered
                                            (gpointer data,
                                             GtkAccelMapForeach foreach_func);
GtkAccelMap* gtk_accel_map_get              (void);
void        gtk_accel_map_lock_path         (const gchar *accel_path);
```

```
void        gtk_accel_map_unlock_path       (const gchar *accel_path);
```

Object Hierarchy

```
  GObject
   +----GtkAccelMap
```

Signal Prototypes

```
"changed"   void        user_function      (GtkAccelMap *object,
                                            gchar *accel_path,
                                            guint accel_key,
                                            GdkModifierType accel_mods,
                                            gpointer user_data);
```

Description

Details

GtkAccelMap

```
typedef struct _GtkAccelMap GtkAccelMap;
```

GtkAccelMapForeach ()

```
void        (*GtkAccelMapForeach)           (gpointer data,
                                             const gchar *accel_path,
                                             guint accel_key,
                                             GdkModifierType accel_mods,
                                             gboolean changed);
```

data :
accel_path :
accel_key :
accel_mods :
changed :

---

gtk_accel_map_add_entry ()

```
void        gtk_accel_map_add_entry         (const gchar *accel_path,
                                             guint accel_key,
                                             GdkModifierType accel_mods);
```

Registers a new accelerator with the global accelerator map. This function should only be called once per *accel_path* with the canonical *accel_key* and *accel_mods* for this path. To change the accelerator during runtime programatically, use gtk_accel_map_change_entry(). The accelerator path must consist of "<WINDOWTYPE>/Category1/Category2/.../Action", where <WINDOWTYPE> should be a unique application-specific identifier, that corresponds to the kind of window the accelerator is being used in, e.g. "Gimp-Image", "Abiword-Document" or "Gnumeric-Settings". The Category1/.../Action portion is most appropriately chosen by the action the accelerator triggers, i.e. for accelerators on menu items, choose the item's menu path, e.g. "File/Save As", "Image/View/Zoom" or "Edit/Select All". So a full valid accelerator path may look like: "<Gimp-Toolbox>/File/Dialogs/Tool Options...".

accel_path : valid accelerator path
accel_key : the accelerator key
accel_mods : the accelerator modifiers

---

gtk_accel_map_lookup_entry ()

```
gboolean    gtk_accel_map_lookup_entry      (const gchar *accel_path,
                                             GtkAccelKey *key);
```

Looks up the accelerator entry for *accel_path* and fills in *key*.

accel_path : a valid accelerator path
key : the accelerator key to be filled in (optional)
Returns : TRUE if *accel_path* is known, FALSE otherwise

---

gtk_accel_map_change_entry ()

```
gboolean    gtk_accel_map_change_entry      (const gchar *accel_path,
                                             guint accel_key,
                                             GdkModifierType accel_mods,
                                             gboolean replace);
```

Changes the *accel_key* and *accel_mods* currently associated with *accel_path*. Due to conflicts with other accelerators, a change may not always be possible, *replace* indicates whether other accelerators may be deleted to resolve such conflicts. A change will only occur if all conflicts could be resolved (which might not be the case if conflicting accelerators are locked). Successful changes are indicated by a TRUE return value.

accel_path : a valid accelerator path
accel_key : the new accelerator key
accel_mods : the new accelerator modifiers
replace : TRUE if other accelerators may be deleted upon conflicts
Returns : TRUE if the accelerator could be changed, FALSE otherwise

---

gtk_accel_map_load ()

```
void        gtk_accel_map_load              (const gchar *file_name);
```

Parses a file previously saved with gtk_accel_map_save() for accelerator specifications, and propagates them accordingly.

file_name : a file containing accelerator specifications

---

gtk_accel_map_save ()

```
void        gtk_accel_map_save              (const gchar *file_name);
```

Saves current accelerator specifications (accelerator path, key and modifiers) to *file_name*. The file is written in a format suitable to be read back in by gtk_accel_map_load().

file_name : the file to contain accelerator specifications

---

gtk_accel_map_foreach ()

```
void        gtk_accel_map_foreach           (gpointer data,
                                             GtkAccelMapForeach foreach_func);
```

Loops over the entries in the accelerator map whose accel path doesn't match any of the filters added with `gtk_accel_map_add_filter()`, and execute *`foreach_func`* on each. The signature of *`foreach_func`* is that of GtkAccelMapForeach, the *`changed`* parameter indicates whether this accelerator was changed during runtime (thus, would need saving during an accelerator map dump).

`data` : data to be passed into *`foreach_func`*

`foreach_func` : function to be executed for each accel map entry which is not filtered out

---

gtk_accel_map_load_fd ()

```
void        gtk_accel_map_load_fd           (gint fd);
```

Filedescriptor variant of `gtk_accel_map_load()`.

Note that the file descriptor will not be closed by this function.

`fd` : a valid readable file descriptor

---

gtk_accel_map_save_fd ()

```
void        gtk_accel_map_save_fd           (gint fd);
```

Filedescriptor variant of `gtk_accel_map_save()`.

Note that the file descriptor will not be closed by this function.

`fd` : a valid writable file descriptor

---

gtk_accel_map_load_scanner ()

```
void        gtk_accel_map_load_scanner      (GScanner *scanner);
```

GScanner variant of `gtk_accel_map_load()`.

scanner : a GScanner which has already been provided with an input file

---

gtk_accel_map_add_filter ()

```
void        gtk_accel_map_add_filter        (const gchar *filter_pattern);
```

Adds a filter to the global list of accel path filters.

Accel map entries whose accel path matches one of the filters are skipped by `gtk_accel_map_foreach()`.

This function is intended for GTK+ modules that create their own menus, but don't want them to be saved into the applications accelerator map dump.

filter_pattern : a pattern (see GPatternSpec)

---

gtk_accel_map_foreach_unfiltered ()

```
void        gtk_accel_map_foreach_unfiltered
                                            (gpointer data,
                                             GtkAccelMapForeach foreach_func);
```

Loops over all entries in the accelerator map, and execute *foreach_func* on each. The signature of *foreach_func* is that of GtkAccelMapForeach, the *changed* parameter indicates whether this accelerator was changed during runtime (thus, would need saving during an accelerator map dump).

data : data to be passed into *foreach_func*

foreach_func : function to be executed for each accel map entry

---

gtk_accel_map_get ()

```
GtkAccelMap* gtk_accel_map_get                (void);
```

Gets the singleton global GtkAccelMap object. This object is useful only for notification of changes to the accelerator map via the ::changed signal; it isn't a parameter to the other accelerator map functions.

Returns : the global GtkAccelMap object

Since 2.4

---

gtk_accel_map_lock_path ()

```
void        gtk_accel_map_lock_path         (const gchar *accel_path);
```

Locks the given accelerator path. If the accelerator map doesn't yet contain an entry for *accel_path*, a new one is created.

Locking an accelerator path prevents its accelerator from being changed during runtime. A locked accelerator path can be unlocked by gtk_accel_map_unlock_path(). Refer to gtk_accel_map_change_entry() for information about runtime accelerator changes.

If called more than once, *accel_path* remains locked until gtk_accel_map_unlock_path() has been called an equivalent number of times.

Note that locking of individual accelerator paths is independent from locking the GtkAccelGroup containing them. For runtime accelerator changes to be possible both the accelerator path and its GtkAccelGroup have to be unlocked.

accel_path : a valid accelerator path

Since 2.4

---

gtk_accel_map_unlock_path ()

```
void        gtk_accel_map_unlock_path       (const gchar *accel_path);
```

Undoes the last call to gtk_accel_map_lock_path() on this *accel_path*. Refer to

gtk_accel_map_lock_path() for information about accelerator path locking.

accel_path : a valid accelerator path

Since 2.4

Signals

The "changed" signal

```
void        user_function                   (GtkAccelMap *object,
                                             gchar *accel_path,
                                             guint accel_key,
                                             GdkModifierType accel_mods,
                                             gpointer user_data);
```

Notifies of a change in the global accelerator map. The path is also used as the detail for the signal, so it is possible to connect to changed::*accel_path*.

object : the global accel map object
accel_path : the path of the accelerator that changed
accel_key : the key value for the new accelerator
accel_mods : the modifier mask for the new accelerator
user_data : user data set when the signal handler was connected.

Since 2.4

<< Accelerator Groups **Clipboards >>**

Clipboards

Clipboards — Storing data on clipboards

Synopsis

```
#include <gtk/gtk.h>

            GtkClipboard;
void        (*GtkClipboardReceivedFunc)     (GtkClipboard *clipboard,
                                             GtkSelectionData *selection_data,
                                             gpointer data);
void        (*GtkClipboardTextReceivedFunc) (GtkClipboard *clipboard,
                                             const gchar *text,
                                             gpointer data);
void        (*GtkClipboardTargetsReceivedFunc)
                                            (GtkClipboard *clipboard,
                                             GdkAtom *atoms,
                                             gint n_atoms,
                                             gpointer data);
void        (*GtkClipboardGetFunc)          (GtkClipboard *clipboard,
                                             GtkSelectionData *selection_data,
                                             guint info,
                                             gpointer user_data_or_owner);
void        (*GtkClipboardClearFunc)        (GtkClipboard *clipboard,
                                             gpointer user_data_or_owner);
GtkClipboard* gtk_clipboard_get             (GdkAtom selection);
GtkClipboard* gtk_clipboard_get_for_display (GdkDisplay *display,
                                             GdkAtom selection);
GdkDisplay* gtk_clipboard_get_display       (GtkClipboard *clipboard);
gboolean    gtk_clipboard_set_with_data     (GtkClipboard *clipboard,
                                             const GtkTargetEntry *targets,
                                             guint n_targets,
                                             GtkClipboardGetFunc get_func,
                                             GtkClipboardClearFunc clear_func,
                                             gpointer user_data);
gboolean    gtk_clipboard_set_with_owner    (GtkClipboard *clipboard,
                                             const GtkTargetEntry *targets,
                                             guint n_targets,
```

```
                                                GtkClipboardGetFunc get_func,
                                                GtkClipboardClearFunc clear_func,
                                                GObject *owner);
GObject*    gtk_clipboard_get_owner         (GtkClipboard *clipboard);
void        gtk_clipboard_clear             (GtkClipboard *clipboard);
void        gtk_clipboard_set_text          (GtkClipboard *clipboard,
                                                const gchar *text,
                                                gint len);
void        gtk_clipboard_request_contents  (GtkClipboard *clipboard,
                                                GdkAtom target,
                                                GtkClipboardReceivedFunc callback,
                                                gpointer user_data);
void        gtk_clipboard_request_text      (GtkClipboard *clipboard,
                                                GtkClipboardTextReceivedFunc callback,
                                                gpointer user_data);
void        gtk_clipboard_request_targets   (GtkClipboard *clipboard,
                                                GtkClipboardTargetsReceivedFunc
callback,
                                                gpointer user_data);
GtkSelectionData* gtk_clipboard_wait_for_contents
                                            (GtkClipboard *clipboard,
                                                GdkAtom target);
gchar*      gtk_clipboard_wait_for_text     (GtkClipboard *clipboard);
gboolean    gtk_clipboard_wait_is_text_available
                                            (GtkClipboard *clipboard);
gboolean    gtk_clipboard_wait_for_targets  (GtkClipboard *clipboard,
                                                GdkAtom **targets,
                                                gint *n_targets);
gboolean    gtk_clipboard_wait_is_target_available
                                            (GtkClipboard *clipboard,
                                                GdkAtom target);
void        gtk_clipboard_set_can_store     (GtkClipboard *clipboard,
                                                GtkTargetEntry *targets,
                                                gint n_targets);
void        gtk_clipboard_store             (GtkClipboard *clipboard);
```

Description

The GtkClipboard object represents a clipboard of data shared between different processes or between different widgets in the same process. Each clipboard is identified by a name encoded as a GdkAtom. (Conversion to and from strings can be done with `gdk_atom_intern()` and `gdk_atom_name()`.) The default clipboard corresponds to the "CLIPBOARD" atom; another commonly used clipboard is the "PRIMARY" clipboard, which, in X, traditionally contains the currently selected text.

To support having a number of different formats on the clipboard at the same time, the clipboard mechanism allows providing callbacks instead of the actual data. When you set the contents of the clipboard, you can either supply the data directly (via functions like `gtk_clipboard_set_text()`), or you can supply a callback to be called at a later time when the data is needed (via `gtk_clipboard_set_with_data()` or `gtk_clipboard_set_with_owner()`.) Providing a callback also avoids having to make copies of the data when it is not needed.

`gtk_clipboard_set_with_data()` and `gtk_clipboard_set_with_owner()` are quite similar; the choice between the two depends mostly on which is more convenient in a particular situation. The former is most useful when you want to have a blob of data with callbacks to convert it into the various data types that you advertise. When the *`clear_func`* you provided is called, you simply free the data blob. The latter is more useful when the contents of clipboard reflect the internal state of a GObject (As an example, for the PRIMARY clipboard, when an entry widget provides the clipboard's contents the contents are simply the text within the selected region.) If the contents change, the entry widget can call `gtk_clipboard_set_with_owner()` to update the timestamp for clipboard ownership, without having to worry about *`clear_func`* being called.

Requesting the data from the clipboard is essentially asynchronous. If the contents of the clipboard are provided within the same process, then a direct function call will be made to retrieve the data, but if they are provided by another process, then the data needs to be retrieved from the other process, which may take some time. To avoid blocking the user interface, the call to request the selection, `gtk_clipboard_request_contents()` takes a callback that will be called when the contents are received (or when the request fails.) If you don't want to deal with providing a separate callback, you can also use `gtk_clipboard_wait_for_contents()`. What this does is run the GLib main loop recursively waiting for the contents. This can simplify the code flow, but you still have to be aware that other callbacks in your program can be called while this recursive mainloop is running.

Along with the functions to get the clipboard contents as an arbitrary data chunk, there are also functions to retrieve it as text, `gtk_clipboard_request_text()` and `gtk_clipboard_wait_for_text()`. These functions take care of determining which formats are advertised by the clipboard provider, asking for the clipboard in the best available format and converting the results into the UTF-8 encoding. (The standard form for representing strings in GTK+.)

Details

GtkClipboard

```
typedef struct _GtkClipboard GtkClipboard;
```

GtkClipboardReceivedFunc ()

```
void        (*GtkClipboardReceivedFunc)     (GtkClipboard *clipboard,
                                             GtkSelectionData *selection_data,
                                             gpointer data);
```

A function to be called when the results of `gtk_clipboard_request_contents()` are received, or when the request fails.

clipboard :	the GtkClipboard
selection_data :	a GtkSelectionData containing the data was received. If retrieving the data failed, then then length field of *selection_data* will be negative.
data :	the *user_data* supplied to gtk_clipboard_request_contents().

GtkClipboardTextReceivedFunc ()

```
void        (*GtkClipboardTextReceivedFunc) (GtkClipboard *clipboard,
                                             const gchar *text,
                                             gpointer data);
```

A function to be called when the results of gtk_clipboard_request_text() are received, or when the request fails.

clipboard :	the GtkClipboard
text :	the text received, as a UTF-8 encoded string, or NULL if retrieving the data failed.
data :	the *user_data* supplied to gtk_clipboard_request_text().

GtkClipboardTargetsReceivedFunc ()

```
void        (*GtkClipboardTargetsReceivedFunc)
                                            (GtkClipboard *clipboard,
                                             GdkAtom *atoms,
                                             gint n_atoms,
                                             gpointer data);
```

A function to be called when the results of gtk_clipboard_request_targets() are received, or when the request fails.

clipboard :	the GtkClipboard
atoms :	the supported targets, as array of GdkAtom, or NULL if retrieving the data failed.
n_atoms :	the length of the *atoms* array.
data :	the *user_data* supplied to gtk_clipboard_request_targets().

Since 2.4

GtkClipboardGetFunc ()

```
void        (*GtkClipboardGetFunc)          (GtkClipboard *clipboard,
                                             GtkSelectionData *selection_data,
                                             guint info,
                                             gpointer user_data_or_owner);
```

A function that will be called to provide the contents of the selection. If multiple types of data were advertised, the requested type can be determined from the *info* parameter or by checking the target field of *selection_data*. If the data could successfully be converted into then it should be stored into the *selection_data* object by calling gtk_selection_data_set() (or related functions such as gtk_selection_data_set_text()). If no data is set, the requestor will be informed that the attempt to get the data failed.

clipboard :	the GtkClipboard
selection_data :	a GtkSelectionData argument in which the requested data should be stored.
info :	the info field corresponding to the requested target from the GtkTargetEntry array passed to gtk_clipboard_set_with_data() or gtk_clipboard_set_with_owner().
user_data_or_owner :	the *user_data* argument passed to gtk_clipboard_set_with_data(), or the *owner* argument passed to gtk_clipboard_set_with_owner()

GtkClipboardClearFunc ()

```
void        (*GtkClipboardClearFunc)        (GtkClipboard *clipboard,
                                             gpointer user_data_or_owner);
```

A function that will be called when the contents of the clipboard are changed or cleared. Once this has called, the *user_data_or_owner* argument will not be used again.

clipboard :	the GtkClipboard
user_data_or_owner :	the *user_data* argument passed to gtk_clipboard_set_with_data(), or the *owner* argument passed to gtk_clipboard_set_with_owner()

---

gtk_clipboard_get ()

```
GtkClipboard* gtk_clipboard_get             (GdkAtom selection);
```

Returns the clipboard object for the given selection. See gtk_clipboard_get_for_display() for complete details.

selection : a GdkAtom which identifies the clipboard to use.

Returns : the appropriate clipboard object. If no clipboard already exists, a new one will be created. Once a clipboard object has been created, it is persistent for all time and cannot be freed.

---

gtk_clipboard_get_for_display ()

```
GtkClipboard* gtk_clipboard_get_for_display (GdkDisplay *display,
                                             GdkAtom selection);
```

Returns the clipboard object for the given selection. Cut/copy/paste menu items and keyboard shortcuts should use the default clipboard, returned by passing GDK_SELECTION_CLIPBOARD for *selection*. (GDK_NONE is supported as a synonym for GDK_SELECTION_CLIPBOARD for backwards compatibility reasons.) The currently-selected object or text should be provided on the clipboard identified by GDK_SELECTION_PRIMARY. Cut/copy/paste menu items conceptually copy the contents of the GDK_SELECTION_PRIMARY clipboard to the default clipboard, i.e. they copy the selection to what the user sees as the clipboard.

(Passing GDK_NONE is the same as using gdk_atom_intern ("CLIPBOARD", FALSE). See http://www.freedesktop.org/standards/clipboards-spec/clipboards.txt for a detailed discussion of the "CLIPBOARD" vs. "PRIMARY" selections under the X window system. On Win32 the GDK_SELECTION_PRIMARY clipboard is essentially ignored.)

It's possible to have arbitrary named clipboards; if you do invent new clipboards, you should prefix the selection name with an underscore (because the ICCCM requires that nonstandard atoms are underscore-prefixed), and namespace it as well. For example, if your application called "Foo" has a special-purpose clipboard, you might call it "_FOO_SPECIAL_CLIPBOARD".

display : the display for which the clipboard is to be retrieved or created

selection : a GdkAtom which identifies the clipboard to use.

Returns : the appropriate clipboard object. If no clipboard already exists, a new one will be created. Once a clipboard object has been created, it is persistent for all time and cannot be freed.

Since 2.2

---

gtk_clipboard_get_display ()

```
GdkDisplay* gtk_clipboard_get_display       (GtkClipboard *clipboard);
```

Gets the GdkDisplay associated with *clipboard*

clipboard : a GtkClipboard

Returns : the GdkDisplay associated with *clipboard*

Since 2.2

---

gtk_clipboard_set_with_data ()

```
gboolean    gtk_clipboard_set_with_data     (GtkClipboard *clipboard,
                                             const GtkTargetEntry *targets,
                                             guint n_targets,
                                             GtkClipboardGetFunc get_func,
                                             GtkClipboardClearFunc clear_func,
                                             gpointer user_data);
```

Virtually sets the contents of the specified clipboard by providing a list of supported formats for the clipboard data and a function to call to get the actual data when it is requested.

clipboard :	a GtkClipboard
targets :	array containing information about the available forms for the clipboard data
n_targets :	number of elements in *targets*
get_func :	function to call to get the actual clipboard data
clear_func :	when the clipboard contents are set again, this function will be called, and *get_func* will not be subsequently called.
user_data :	user data to pass to *get_func* and *clear_func*.
Returns :	TRUE if setting the clipboard data succeeded. If setting the clipboard data failed the provided callback functions will be ignored.

---

gtk_clipboard_set_with_owner ()

```
gboolean    gtk_clipboard_set_with_owner    (GtkClipboard *clipboard,
                                             const GtkTargetEntry *targets,
                                             guint n_targets,
                                             GtkClipboardGetFunc get_func,
                                             GtkClipboardClearFunc clear_func,
                                             GObject *owner);
```

Virtually sets the contents of the specified clipboard by providing a list of supported formats for the clipboard data and a function to call to get the actual data when it is requested.

The difference between this function and `gtk_clipboard_set_with_data()` is that instead of an generic

user_data pointer, a GObject is passed in.

clipboard :	a GtkClipboard
targets :	array containing information about the available forms for the clipboard data
n_targets :	number of elements in *targets*
get_func :	function to call to get the actual clipboard data
clear_func :	when the clipboard contents are set again, this function will be called, and *get_func* will not be subsequently called.
owner :	an object that "owns" the data. This object will be passed to the callbacks when called.
Returns :	TRUE if setting the clipboard data succeeded. If setting the clipboard data failed the provided callback functions will be ignored.

---

gtk_clipboard_get_owner ()

```
GObject*    gtk_clipboard_get_owner         (GtkClipboard *clipboard);
```

If the clipboard contents callbacks were set with gtk_clipboard_set_with_owner(), and the gtk_clipboard_set_with_data() or gtk_clipboard_clear() has not subsequently called, returns the owner set by gtk_clipboard_set_with_owner().

clipboard :	a GtkClipboard
Returns :	the owner of the clipboard, if any; otherwise NULL.

---

gtk_clipboard_clear ()

```
void        gtk_clipboard_clear             (GtkClipboard *clipboard);
```

Clears the contents of the clipboard. Generally this should only be called between the time you call gtk_clipboard_set_with_owner() or gtk_clipboard_set_with_data(), and when the *clear_func* you supplied is called. Otherwise, the clipboard may be owned by someone else.

clipboard :	a GtkClipboard

---

gtk_clipboard_set_text ()

```
void        gtk_clipboard_set_text          (GtkClipboard *clipboard,
                                             const gchar *text,
```

```
                                                gint len);
```

Sets the contents of the clipboard to the given UTF-8 string. GTK+ will make a copy of the text and take responsibility for responding for requests for the text, and for converting the text into the requested format.

clipboard : a GtkClipboard object
text : a UTF-8 string.
len : length of *text*, in bytes, or -1, in which case the length will be determined with `strlen()`.

---

gtk_clipboard_request_contents ()

```
void        gtk_clipboard_request_contents  (GtkClipboard *clipboard,
                                             GdkAtom target,
                                             GtkClipboardReceivedFunc callback,
                                             gpointer user_data);
```

Requests the contents of clipboard as the given target. When the results of the result are later received the supplied callback will be called.

clipboard : a GtkClipboard
target : an atom representing the form into which the clipboard owner should convert the selection.
callback : A function to call when the results are received (or the retrieval fails). If the retrieval fails the length field of *selection_data* will be negative.
user_data : user data to pass to *callback*

---

gtk_clipboard_request_text ()

```
void        gtk_clipboard_request_text      (GtkClipboard *clipboard,
                                             GtkClipboardTextReceivedFunc callback,
                                             gpointer user_data);
```

Requests the contents of the clipboard as text. When the text is later received, it will be converted to UTF-8 if necessary, and *callback* will be called.

The *text* parameter to *callback* will contain the resulting text if the request succeeded, or `NULL` if it failed. This could happen for various reasons, in particular if the clipboard was empty or if the contents of the clipboard could not be converted into text form.

clipboard : a GtkClipboard

callback : a function to call when the text is received, or the retrieval fails. (It will always be called one way or the other.)

user_data : user data to pass to *callback*.

---

gtk_clipboard_request_targets ()

```
void        gtk_clipboard_request_targets   (GtkClipboard *clipboard,
                                             GtkClipboardTargetsReceivedFunc
callback,
                                             gpointer user_data);
```

Requests the contents of the clipboard as list of supported targets. When the list is later received, *callback* will be called.

The *targets* parameter to *callback* will contain the resulting targets if the request succeeded, or NULL if it failed.

clipboard : a GtkClipboard

callback : a function to call when the targets are received, or the retrieval fails. (It will always be called one way or the other.)

user_data : user data to pass to *callback*.

Since 2.4

---

gtk_clipboard_wait_for_contents ()

```
GtkSelectionData* gtk_clipboard_wait_for_contents
                                            (GtkClipboard *clipboard,
                                             GdkAtom target);
```

Requests the contents of the clipboard using the given target. This function waits for the data to be received using the main loop, so events, timeouts, etc, may be dispatched during the wait.

clipboard : a GtkClipboard

target : an atom representing the form into which the clipboard owner should convert the selection.

Returns : a newly-allocated GtkSelectionData object or NULL if retrieving the given target failed. If non-NULL, this value must be freed with gtk_selection_data_free() when you are finished with it.

---

gtk_clipboard_wait_for_text ()

```
gchar*      gtk_clipboard_wait_for_text     (GtkClipboard *clipboard);
```

Requests the contents of the clipboard as text and converts the result to UTF-8 if necessary. This function waits for the data to be received using the main loop, so events, timeouts, etc, may be dispatched during the wait.

clipboard : a GtkClipboard

Returns : a newly-allocated UTF-8 string which must be freed with g_free(), or NULL if retrieving the selection data failed. (This could happen for various reasons, in particular if the clipboard was empty or if the contents of the clipboard could not be converted into text form.)

---

gtk_clipboard_wait_is_text_available ()

```
gboolean    gtk_clipboard_wait_is_text_available
                                            (GtkClipboard *clipboard);
```

Test to see if there is text available to be pasted This is done by requesting the TARGETS atom and checking if it contains any of the names: STRING, TEXT, COMPOUND_TEXT, UTF8_STRING. This function waits for the data to be received using the main loop, so events, timeouts, etc, may be dispatched during the wait.

This function is a little faster than calling gtk_clipboard_wait_for_text() since it doesn't need to retrieve the actual text.

clipboard : a GtkClipboard

Returns : TRUE is there is text available, FALSE otherwise.

---

gtk_clipboard_wait_for_targets ()

```
gboolean    gtk_clipboard_wait_for_targets  (GtkClipboard *clipboard,
                                             GdkAtom **targets,
                                             gint *n_targets);
```

Returns a list of targets that are present on the clipboard, or NULL if there aren't any targets available. The returned list must be freed with g_free(). This function waits for the data to be received using the main loop, so events, timeouts, etc, may be dispatched during the wait.

clipboard : a GtkClipboard

targets : location to store an array of targets. The result stored here must be freed with g_free().

n_targets : location to store number of items in *targets*.

Returns : TRUE if any targets are present on the clipboard, otherwise FALSE.

Since 2.4

---

gtk_clipboard_wait_is_target_available ()

```
gboolean    gtk_clipboard_wait_is_target_available
                                            (GtkClipboard *clipboard,
                                             GdkAtom target);
```

Checks if a clipboard supports pasting data of a given type. This function can be used to determine if a "Paste" menu item should be insensitive or not.

If you want to see if there's text available on the clipboard, use `gtk_clipboard_wait_is_text_available()` instead.

clipboard : a GtkClipboard

target : A GdkAtom indicating which target to look for.

Returns : TRUE if the target is available, FALSE otherwise.

Since 2.6

---

gtk_clipboard_set_can_store ()

```
void        gtk_clipboard_set_can_store     (GtkClipboard *clipboard,
                                             GtkTargetEntry *targets,
                                             gint n_targets);
```

Hints that the clipboard data should be stored somewhere when the application exits or when `gtk_clipboard_store()` is called.

This value is reset when the clipboard owner changes. Where the clipboard data is stored is platform dependent, see `gdk_display_store_clipboard()` for more information.

clipboard : a GtkClipboard

targets : array containing information about which forms should be stored or NULL to indicate that all forms should be stored.

n_targets : number of elements in *targets*

Since 2.6

---

gtk_clipboard_store ()

```
void        gtk_clipboard_store             (GtkClipboard *clipboard);
```

Stores the current clipboard data somewhere so that it will stay around after the application has quit.

clipboard : a GtkClipboard

Since 2.6

See Also

GtkSelection — GtkClipboard provides a high-level wrapper around the lower level routines that deal with X selections. It is also possibly to directly manipulate the X selections, though it is seldom necessary to do so.

<< Accelerator Maps **Drag and Drop >>**

Drag and Drop

Drag and Drop — Functions for controlling drag and drop handling

Synopsis

```
#include <gtk/gtk.h>

enum        GtkDestDefaults;
enum        GtkTargetFlags;

void        gtk_drag_dest_set               (GtkWidget *widget,
                                             GtkDestDefaults flags,
                                             const GtkTargetEntry *targets,
                                             gint n_targets,
                                             GdkDragAction actions);
void        gtk_drag_dest_set_proxy         (GtkWidget *widget,
                                             GdkWindow *proxy_window,
                                             GdkDragProtocol protocol,
                                             gboolean use_coordinates);
void        gtk_drag_dest_unset             (GtkWidget *widget);
GdkAtom     gtk_drag_dest_find_target       (GtkWidget *widget,
                                             GdkDragContext *context,
                                             GtkTargetList *target_list);
GtkTargetList* gtk_drag_dest_get_target_list
                                            (GtkWidget *widget);
void        gtk_drag_dest_set_target_list   (GtkWidget *widget,
                                             GtkTargetList *target_list);
void        gtk_drag_dest_add_text_targets  (GtkWidget *widget);
void        gtk_drag_dest_add_image_targets (GtkWidget *widget);
void        gtk_drag_dest_add_uri_targets   (GtkWidget *widget);
void        gtk_drag_finish                 (GdkDragContext *context,
                                             gboolean success,
                                             gboolean del,
                                             guint32 time_);
void        gtk_drag_get_data               (GtkWidget *widget,
                                             GdkDragContext *context,
```

```
                                                GdkAtom target,
                                                guint32 time_);
GtkWidget*  gtk_drag_get_source_widget      (GdkDragContext *context);
void        gtk_drag_highlight              (GtkWidget *widget);
void        gtk_drag_unhighlight            (GtkWidget *widget);

GdkDragContext* gtk_drag_begin              (GtkWidget *widget,
                                                GtkTargetList *targets,
                                                GdkDragAction actions,
                                                gint button,
                                                GdkEvent *event);
void        gtk_drag_set_icon_widget        (GdkDragContext *context,
                                                GtkWidget *widget,
                                                gint hot_x,
                                                gint hot_y);
void        gtk_drag_set_icon_pixmap        (GdkDragContext *context,
                                                GdkColormap *colormap,
                                                GdkPixmap *pixmap,
                                                GdkBitmap *mask,
                                                gint hot_x,
                                                gint hot_y);
void        gtk_drag_set_icon_pixbuf        (GdkDragContext *context,
                                                GdkPixbuf *pixbuf,
                                                gint hot_x,
                                                gint hot_y);
void        gtk_drag_set_icon_stock         (GdkDragContext *context,
                                                const gchar *stock_id,
                                                gint hot_x,
                                                gint hot_y);
void        gtk_drag_set_icon_default       (GdkDragContext *context);
void        gtk_drag_set_default_icon       (GdkColormap *colormap,
                                                GdkPixmap *pixmap,
                                                GdkBitmap *mask,
                                                gint hot_x,
                                                gint hot_y);
gboolean    gtk_drag_check_threshold        (GtkWidget *widget,
                                                gint start_x,
                                                gint start_y,
                                                gint current_x,
                                                gint current_y);
void        gtk_drag_source_set             (GtkWidget *widget,
                                                GdkModifierType start_button_mask,
                                                const GtkTargetEntry *targets,
                                                gint n_targets,
```

```
                                                GdkDragAction actions);
void        gtk_drag_source_set_icon           (GtkWidget *widget,
                                                GdkColormap *colormap,
                                                GdkPixmap *pixmap,
                                                GdkBitmap *mask);
void        gtk_drag_source_set_icon_pixbuf    (GtkWidget *widget,
                                                GdkPixbuf *pixbuf);
void        gtk_drag_source_set_icon_stock     (GtkWidget *widget,
                                                const gchar *stock_id);
void        gtk_drag_source_unset              (GtkWidget *widget);
void        gtk_drag_source_set_target_list    (GtkWidget *widget,
                                                GtkTargetList *target_list);
GtkTargetList* gtk_drag_source_get_target_list
                                               (GtkWidget *widget);
void        gtk_drag_source_add_text_targets
                                               (GtkWidget *widget);
```

Description

GTK+ has a rich set of functions for doing inter-process communication via the drag-and-drop metaphor. GTK+ can do drag-and-drop (DND) via multiple protocols. The currently supported protocols are the Xdnd and Motif protocols. As well as the functions listed here, applications may need to use some facilities provided for Selections. Also, the Drag and Drop API makes use of signals in the GtkWidget class.

Details

enum GtkDestDefaults

```
typedef enum {
  GTK_DEST_DEFAULT_MOTION     = 1 << 0, /* respond to "drag_motion" */
  GTK_DEST_DEFAULT_HIGHLIGHT  = 1 << 1, /* auto-highlight */
  GTK_DEST_DEFAULT_DROP       = 1 << 2, /* respond to "drag_drop" */
  GTK_DEST_DEFAULT_ALL        = 0x07
} GtkDestDefaults;
```

The GtkDestDefaults enumeration specifies the various types of action that will be taken on behalf of the user for a drag destination site.

GTK_DEST_DEFAULT_MOTION	If set for a widget, GTK+, during a drag over this widget will check if the drag matches this widget's list of possible targets and actions. GTK+ will then call gdk_drag_status() as appropriate.

GTK_DEST_DEFAULT_HIGHLIGHT	If set for a widget, GTK+ will draw a highlight on this widget as long as a drag is over this widget and the widget drag format and action are acceptable.
GTK_DEST_DEFAULT_DROP	If set for a widget, when a drop occurs, GTK+ will will check if the drag matches this widget's list of possible targets and actions. If so, GTK+ will call gtk_drag_get_data() on behalf of the widget. Whether or not the drop is successful, GTK+ will call gtk_drag_finish(). If the action was a move, then if the drag was successful, then TRUE will be passed for the *delete* parameter to gtk_drag_finish().
GTK_DEST_DEFAULT_ALL	If set, specifies that all default actions should be taken.

---

enum GtkTargetFlags

```
typedef enum {
  GTK_TARGET_SAME_APP = 1 << 0,     /*< nick=same-app >*/
  GTK_TARGET_SAME_WIDGET = 1 << 1   /*< nick=same-widget >*/
} GtkTargetFlags;
```

The GtkTargetFlags enumeration is used to specify constraints on an entry in a GtkTargetTable.

GTK_TARGET_SAME_APP	If this is set, the target will only be selected for drags within a single application.
GTK_TARGET_SAME_WIDGET	If this is set, the target will only be selected for drags within a single widget.

---

gtk_drag_dest_set ()

```
void        gtk_drag_dest_set               (GtkWidget *widget,
                                             GtkDestDefaults flags,
                                             const GtkTargetEntry *targets,
                                             gint n_targets,
                                             GdkDragAction actions);
```

Sets a widget as a potential drop destination.

widget : a GtkWidget

flags :	the flags that specify what actions GTK+ should take on behalf of a widget for drops onto that widget. The *targets* and *actions* fields only are used if GTK_DEST_DEFAULT_MOTION or GTK_DEST_DEFAULT_DROP are given.
targets :	a pointer to an array of GtkTargetEntrys indicating the drop types that this widget will accept.
n_targets :	the number of entries in *targets*.
actions :	a bitmask of possible actions for a drop onto this widget.

---

gtk_drag_dest_set_proxy ()

```
void        gtk_drag_dest_set_proxy         (GtkWidget *widget,
                                             GdkWindow *proxy_window,
                                             GdkDragProtocol protocol,
                                             gboolean use_coordinates);
```

Sets this widget as a proxy for drops to another window.

widget :	a GtkWidget
proxy_window :	the window to which to forward drag events
protocol :	the drag protocol which the *proxy_window* accepts (You can use gdk_drag_get_protocol() to determine this)
use_coordinates :	If TRUE, send the same coordinates to the destination, because it is an embedded subwindow.

---

gtk_drag_dest_unset ()

```
void        gtk_drag_dest_unset             (GtkWidget *widget);
```

Clears information about a drop destination set with gtk_drag_dest_set(). The widget will no longer receive notification of drags.

widget : a GtkWidget

---

gtk_drag_dest_find_target ()

```
GdkAtom     gtk_drag_dest_find_target       (GtkWidget *widget,
```

```
                                                GdkDragContext *context,
                                                GtkTargetList *target_list);
```

Looks for a match between *context->targets* and the *dest_target_list*, returning the first matching target, otherwise returning GDK_NONE. *dest_target_list* should usually be the return value from gtk_drag_dest_get_target_list(), but some widgets may have different valid targets for different parts of the widget; in that case, they will have to implement a drag_motion handler that passes the correct target list to this function.

widget : drag destination widget
context : drag context
target_list : list of droppable targets, or NULL to use gtk_drag_dest_get_target_list (*widget*).
Returns : first target that the source offers and the dest can accept, or GDK_NONE

---

gtk_drag_dest_get_target_list ()

```
GtkTargetList* gtk_drag_dest_get_target_list
                                                (GtkWidget *widget);
```

Returns the list of targets this widget can accept from drag-and-drop.

widget : a GtkWidget
Returns : the GtkTargetList, or NULL if none

---

gtk_drag_dest_set_target_list ()

```
void        gtk_drag_dest_set_target_list   (GtkWidget *widget,
                                             GtkTargetList *target_list);
```

Sets the target types that this widget can accept from drag-and-drop. The widget must first be made into a drag destination with gtk_drag_dest_set().

widget : a GtkWidget that's a drag destination
target_list : list of droppable targets, or NULL for none

---

gtk_drag_dest_add_text_targets ()

```
void        gtk_drag_dest_add_text_targets  (GtkWidget *widget);
```

Add the text targets supported by GtkSelection to the target list of the drag destination. The targets are added with *info* = 0. If you need another value, use gtk_target_list_add_text_targets() and gtk_drag_dest_set_target_list().

widget : a GtkWidget that's a drag destination

Since 2.6

---

gtk_drag_dest_add_image_targets ()

```
void        gtk_drag_dest_add_image_targets (GtkWidget *widget);
```

Add the image targets supported by GtkSelection to the target list of the drag destination. The targets are added with *info* = 0. If you need another value, use gtk_target_list_add_image_targets() and gtk_drag_dest_set_target_list().

widget : a GtkWidget that's a drag destination

Since 2.6

---

gtk_drag_dest_add_uri_targets ()

```
void        gtk_drag_dest_add_uri_targets   (GtkWidget *widget);
```

Add the URI targets supported by GtkSelection to the target list of the drag destination. The targets are added with *info* = 0. If you need another value, use gtk_target_list_add_uri_targets() and gtk_drag_dest_set_target_list().

widget : a GtkWidget that's a drag destination

Since 2.6

gtk_drag_finish ()

```
void        gtk_drag_finish                 (GdkDragContext *context,
                                             gboolean success,
                                             gboolean del,
                                             guint32 time_);
```

Informs the drag source that the drop is finished, and that the data of the drag will no longer be required.

context : the drag context.
success : a flag indicating whether the drop was successful
del : a flag indicating whether the source should delete the original data. (This should be TRUE for a move)
*time_* : the timestamp from the "drag_data_drop" signal.

gtk_drag_get_data ()

```
void        gtk_drag_get_data               (GtkWidget *widget,
                                             GdkDragContext *context,
                                             GdkAtom target,
                                             guint32 time_);
```

Gets the data associated with a drag. When the data is received or the retrieval fails, GTK+ will emit a "drag_data_received" signal. Failure of the retrieval is indicated by the length field of the *selection_data* signal parameter being negative. However, when gtk_drag_get_data() is called implicitely because the GTK_DEST_DEFAULT_DROP was set, then the widget will not receive notification of failed drops.

widget : the widget that will receive the "drag_data_received" signal.
context : the drag context
target : the target (form of the data) to retrieve.
*time_* : a timestamp for retrieving the data. This will generally be the time received in a "drag_data_motion" or "drag_data_drop" signal.

gtk_drag_get_source_widget ()

```
GtkWidget*  gtk_drag_get_source_widget      (GdkDragContext *context);
```

Determines the source widget for a drag.

context : a (destination side) drag context.

Returns : if the drag is occurring within a single application, a pointer to the source widget. Otherwise, NULL.

---

gtk_drag_highlight ()

```
void        gtk_drag_highlight              (GtkWidget *widget);
```

Draws a highlight around a widget. This will attach handlers to "expose_event" and "draw", so the highlight will continue to be displayed until gtk_drag_unhighlight() is called.

widget : a widget to highlight

---

gtk_drag_unhighlight ()

```
void        gtk_drag_unhighlight            (GtkWidget *widget);
```

Removes a highlight set by gtk_drag_highlight() from a widget.

widget : a widget to remove the highlight from.

---

gtk_drag_begin ()

```
GdkDragContext* gtk_drag_begin              (GtkWidget *widget,
                                             GtkTargetList *targets,
                                             GdkDragAction actions,
                                             gint button,
                                             GdkEvent *event);
```

Initiates a drag on the source side. The function only needs to be used when the application is starting drags itself, and is not needed when gtk_drag_source_set() is used.

widget : the source widget.
targets : The targets (data formats) in which the source can provide the data.
actions : A bitmask of the allowed drag actions for this drag.
button : The button the user clicked to start the drag.
event : The event that triggered the start of the drag.
Returns : the context for this drag.

---

gtk_drag_set_icon_widget ()

```
void        gtk_drag_set_icon_widget        (GdkDragContext *context,
                                             GtkWidget *widget,
                                             gint hot_x,
                                             gint hot_y);
```

Changes the icon for a widget to a given widget. GTK+ will not destroy the icon, so if you don't want it to persist, you should connect to the "drag_end" signal and destroy it yourself.

context : the context for a drag. (This must be called with a context for the source side of a drag)
widget : a toplevel window to use as an icon.
hot_x : the X offset within *widget* of the hotspot.
hot_y : the Y offset within *widget* of the hotspot.

---

gtk_drag_set_icon_pixmap ()

```
void        gtk_drag_set_icon_pixmap        (GdkDragContext *context,
                                             GdkColormap *colormap,
                                             GdkPixmap *pixmap,
                                             GdkBitmap *mask,
                                             gint hot_x,
                                             gint hot_y);
```

Sets *pixmap* as the icon for a given drag. GTK+ retains references for the arguments, and will release them when they are no longer needed. In general, gtk_drag_set_icon_pixbuf() will be more convenient to use.

context : the context for a drag. (This must be called with a context for the source side of a drag)
colormap : the colormap of the icon

pixmap : the image data for the icon
mask : the transparency mask for the icon
hot_x : the X offset within *pixmap* of the hotspot.
hot_y : the Y offset within *pixmap* of the hotspot.

---

gtk_drag_set_icon_pixbuf ()

```
void        gtk_drag_set_icon_pixbuf        (GdkDragContext *context,
                                             GdkPixbuf *pixbuf,
                                             gint hot_x,
                                             gint hot_y);
```

Sets *pixbuf* as the icon for a given drag.

context : the context for a drag. (This must be called with a context for the source side of a drag)
pixbuf : the GdkPixbuf to use as the drag icon.
hot_x : the X offset within *widget* of the hotspot.
hot_y : the Y offset within *widget* of the hotspot.

---

gtk_drag_set_icon_stock ()

```
void        gtk_drag_set_icon_stock         (GdkDragContext *context,
                                             const gchar *stock_id,
                                             gint hot_x,
                                             gint hot_y);
```

Sets the the icon for a given drag from a stock ID.

context : the context for a drag. (This must be called with a context for the source side of a drag)
stock_id : the ID of the stock icon to use for the drag.
hot_x : the X offset within the icon of the hotspot.
hot_y : the Y offset within the icon of the hotspot.

---

gtk_drag_set_icon_default ()

```
void        gtk_drag_set_icon_default       (GdkDragContext *context);
```

Sets the icon for a particular drag to the default icon.

context : the context for a drag. (This must be called with a context for the source side of a drag)

---

gtk_drag_set_default_icon ()

```
void        gtk_drag_set_default_icon       (GdkColormap *colormap,
                                             GdkPixmap *pixmap,
                                             GdkBitmap *mask,
                                             gint hot_x,
                                             gint hot_y);
```

Warning

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

Changes the default drag icon. GTK+ retains references for the arguments, and will release them when they are no longer needed. This function is obsolete. The default icon should now be changed via the stock system by changing the stock pixbuf for GTK_STOCK_DND.

colormap : the colormap of the icon
pixmap : the image data for the icon
mask : the transparency mask for an image.
hot_x : The X offset within *widget* of the hotspot.
hot_y : The Y offset within *widget* of the hotspot.

---

gtk_drag_check_threshold ()

```
gboolean    gtk_drag_check_threshold        (GtkWidget *widget,
                                             gint start_x,
                                             gint start_y,
                                             gint current_x,
                                             gint current_y);
```

Checks to see if a mouse drag starting at (*start_x*, *start_y*) and ending at (*current_x*, *current_y*) has passed the GTK+ drag threshold, and thus should trigger the beginning of a drag-and-drop operation.

widget : a GtkWidget
start_x : X coordinate of start of drag
start_y : Y coordinate of start of drag
current_x : current X coordinate
current_y : current Y coordinate
Returns : TRUE if the drag threshold has been passed.

---

gtk_drag_source_set ()

```
void        gtk_drag_source_set             (GtkWidget *widget,
                                             GdkModifierType start_button_mask,
                                             const GtkTargetEntry *targets,
                                             gint n_targets,
                                             GdkDragAction actions);
```

Sets up a widget so that GTK+ will start a drag operation when the user clicks and drags on the widget. The widget must have a window.

widget : a GtkWidget
start_button_mask : the bitmask of buttons that can start the drag
targets : the table of targets that the drag will support
n_targets : the number of items in *targets*
actions : the bitmask of possible actions for a drag from this widget.

---

gtk_drag_source_set_icon ()

```
void        gtk_drag_source_set_icon        (GtkWidget *widget,
                                             GdkColormap *colormap,
                                             GdkPixmap *pixmap,
                                             GdkBitmap *mask);
```

Sets the icon that will be used for drags from a particular widget from a pixmap/mask. GTK+ retains references for the arguments, and will release them when they are no longer needed. Use `gtk_drag_source_set_icon_pixbuf()` instead.

widget : a GtkWidget

colormap : the colormap of the icon
pixmap : the image data for the icon
mask : the transparency mask for an image.

---

gtk_drag_source_set_icon_pixbuf ()

```
void        gtk_drag_source_set_icon_pixbuf (GtkWidget *widget,
                                             GdkPixbuf *pixbuf);
```

Sets the icon that will be used for drags from a particular widget from a GdkPixbuf. GTK+ retains a reference for *pixbuf* and will release it when it is no longer needed.

widget : a GtkWidget
pixbuf : the GdkPixbuf for the drag icon

---

gtk_drag_source_set_icon_stock ()

```
void        gtk_drag_source_set_icon_stock  (GtkWidget *widget,
                                             const gchar *stock_id);
```

Sets the icon that will be used for drags from a particular source to a stock icon.

widget : a GtkWidget
stock_id : the ID of the stock icon to use

---

gtk_drag_source_unset ()

```
void        gtk_drag_source_unset           (GtkWidget *widget);
```

Undoes the effects of `gtk_drag_source_set()`.

widget : a GtkWidget

---

gtk_drag_source_set_target_list ()

```
void        gtk_drag_source_set_target_list (GtkWidget *widget,
                                             GtkTargetList *target_list);
```

Changes the target types that this widget offers for drag-and-drop. The widget must first be made into a drag source with `gtk_drag_source_set()`.

widget : a GtkWidget that's a drag source
target_list : list of draggable targets, or NULL for none

Since 2.4

---

gtk_drag_source_get_target_list ()

```
GtkTargetList* gtk_drag_source_get_target_list
                                             (GtkWidget *widget);
```

Gets the list of targets this widget can provide for drag-and-drop.

widget : a GtkWidget
Returns : the GtkTargetList, or NULL if none

Since 2.4

---

gtk_drag_source_add_text_targets ()

```
void        gtk_drag_source_add_text_targets
                                             (GtkWidget *widget);
```

Add the text targets supported by GtkSelection to the target list of the drag source. The targets are added with *info* = 0. If you need another value, use `gtk_target_list_add_text_targets()` and `gtk_drag_source_set_target_list()`.

widget : a GtkWidget that's is a drag source

Since 2.6

<< Clipboards **GtkIconTheme >>**

GtkIconTheme

GtkIconTheme — Looking up icons by name

Synopsis

```
#include <gtk/gtk.h>

            GtkIconInfo;
            GtkIconTheme;
enum        GtkIconLookupFlags;
#define     GTK_ICON_THEME_ERROR
enum        GtkIconThemeError;
GtkIconTheme* gtk_icon_theme_new           (void);
GtkIconTheme* gtk_icon_theme_get_default   (void);
GtkIconTheme* gtk_icon_theme_get_for_screen (GdkScreen *screen);
void        gtk_icon_theme_set_screen      (GtkIconTheme *icon_theme,
                                            GdkScreen *screen);
void        gtk_icon_theme_set_search_path (GtkIconTheme *icon_theme,
                                            const gchar *path[],
                                            gint n_elements);
void        gtk_icon_theme_get_search_path (GtkIconTheme *icon_theme,
                                            gchar **path[],
                                            gint *n_elements);
void        gtk_icon_theme_append_search_path
                                           (GtkIconTheme *icon_theme,
                                            const gchar *path);
void        gtk_icon_theme_prepend_search_path
                                           (GtkIconTheme *icon_theme,
                                            const gchar *path);
void        gtk_icon_theme_set_custom_theme (GtkIconTheme *icon_theme,
                                            const gchar *theme_name);
```

```
gboolean    gtk_icon_theme_has_icon         (GtkIconTheme *icon_theme,
                                             const gchar *icon_name);
GtkIconInfo* gtk_icon_theme_lookup_icon     (GtkIconTheme *icon_theme,
                                             const gchar *icon_name,
                                             gint size,
                                             GtkIconLookupFlags flags);
GdkPixbuf*  gtk_icon_theme_load_icon        (GtkIconTheme *icon_theme,
                                             const gchar *icon_name,
                                             gint size,
                                             GtkIconLookupFlags flags,
                                             GError **error);
GList*      gtk_icon_theme_list_icons       (GtkIconTheme *icon_theme,
                                             const gchar *context);
gint*       gtk_icon_theme_get_icon_sizes   (GtkIconTheme *icon_theme,
                                             const gchar *icon_name);
char*       gtk_icon_theme_get_example_icon_name
                                            (GtkIconTheme *icon_theme);
gboolean    gtk_icon_theme_rescan_if_needed (GtkIconTheme *icon_theme);
void        gtk_icon_theme_add_builtin_icon (const gchar *icon_name,
                                             gint size,
                                             GdkPixbuf *pixbuf);
GtkIconInfo* gtk_icon_info_copy             (GtkIconInfo *icon_info);
void        gtk_icon_info_free              (GtkIconInfo *icon_info);
gint        gtk_icon_info_get_base_size     (GtkIconInfo *icon_info);
G_CONST_RETURN gchar* gtk_icon_info_get_filename
                                            (GtkIconInfo *icon_info);
GdkPixbuf*  gtk_icon_info_get_builtin_pixbuf
                                            (GtkIconInfo *icon_info);
GdkPixbuf*  gtk_icon_info_load_icon         (GtkIconInfo *icon_info,
                                             GError **error);
void        gtk_icon_info_set_raw_coordinates
                                            (GtkIconInfo *icon_info,
                                             gboolean raw_coordinates);
gboolean    gtk_icon_info_get_embedded_rect (GtkIconInfo *icon_info,
                                             GdkRectangle *rectangle);
gboolean    gtk_icon_info_get_attach_points (GtkIconInfo *icon_info,
                                             GdkPoint **points,
                                             gint *n_points);
G_CONST_RETURN gchar* gtk_icon_info_get_display_name
                                            (GtkIconInfo *icon_info);
```

Object Hierarchy

```
GObject
 +----GtkIconTheme
```

Signal Prototypes

```
"changed"   void        user_function      (GtkIconTheme *icon_theme,
                                            gpointer user_data);
```

Description

GtkIconTheme provides a facility for looking up icons by name and size. The main reason for using a name rather than simply providing a filename is to allow different icons to be used depending on what *icon theme* is selecetd by the user. The operation of icon themes on Linux and Unix follows the Icon Theme Specification. There is a default icon theme, named `hicolor` where applications should install their icons, but more additional application themes can be installed as operating system vendors and users choose.

Named icons are similar to the Themeable Stock Images(3) facility, and the distinction between the two may be a bit confusing. A few things to keep in mind:

- Stock images usually are used in conjunction with Stock Items(3)., such as `GTK_STOCK_OK` or `GTK_STOCK_OPEN`. Named icons are easier to set up and therefore are more useful for new icons that an application wants to add, such as application icons or window icons.
- Stock images can only be loaded at the symbolic sizes defined by the GtkIconSize enumeration, or by custom sizes defined by `gtk_icon_size_register()`, while named icons are more flexible and any pixel size can be specified.
- Because stock images are closely tied to stock items, and thus to actions in the user interface, stock images may come in multiple variants for different widget states or writing directions.

A good rule of thumb is that if there is a stock image for what you want to use, use it, otherwise use a named icon. It turns out that internally stock images are generally defined in terms of one or more named

icons. (An example of the more than one case is icons that depend on writing direction; GTK_STOCK_GO_FORWARD uses the two themed icons "gtk-stock-go-forward-ltr" and "gtk-stock-go-forward-rtl".)

In many cases, named themes are used indirectly, via GtkImage or stock items, rather than directly, but looking up icons directly is also simple. The GtkIconTheme object acts as a database of all the icons in the current theme. You can create new GtkIconTheme objects, but its much more efficient to use the standard icon theme for the GdkScreen so that the icon information is shared with other people looking up icons. In the case where the default screen is being used, looking up an icon can be as simple as:

```
GError *error = NULL;
GtkIconTheme *icon_theme;
GdkPixbuf *pixbuf;

icon_theme = gtk_icon_theme_get_default();
pixbuf = gtk_icon_theme_load_icon (icon_theme,
                                   "my-icon-name", /* icon name */
                                   48, /* size */
                                   0,  /* flags */
                                   &error);
if (!pixbuf)
  {
    g_warning ("Couldn't load icon: %s", error->message);
    g_error_free (message);
  }
else
  {
    /* Use the pixbuf */
    g_object_unref (pixbuf);
  }
```

Details

GtkIconInfo

```
typedef struct _GtkIconInfo GtkIconInfo;
```

Contains information found when looking up an icon in an icon theme.

GtkIconTheme

```
typedef struct _GtkIconTheme GtkIconTheme;
```

Acts as a database of information about an icon theme. Normally, you retrieve the icon theme for a particular screen using `gtk_icon_theme_get_for_screen()` and it will contain information about current icon theme for that screen, but you can also create a new GtkIconTheme object and set the icon theme name explicitely using `gtk_icon_theme_set_custom_theme()`.

---

enum GtkIconLookupFlags

```
typedef enum
{
  GTK_ICON_LOOKUP_NO_SVG = 1 << 0,
  GTK_ICON_LOOKUP_FORCE_SVG = 1 << 1,
  GTK_ICON_LOOKUP_USE_BUILTIN = 1 << 2
} GtkIconLookupFlags;
```

Used to specify options for `gtk_icon_theme_lookup_icon()`

`GTK_ICON_LOOKUP_NO_SVG`	Never return SVG icons, even if gdk-pixbuf supports them. Cannot be used together with `GTK_ICON_LOOKUP_FORCE_SVG`.
`GTK_ICON_LOOKUP_FORCE_SVG`	Return SVG icons, even if gdk-pixbuf doesn't support them. Cannot be used together with `GTK_ICON_LOOKUP_NO_SVG`.
`GTK_ICON_LOOKUP_USE_BUILTIN`	When passed to `gtk_icon_theme_lookup_icon()` includes builtin icons as well as files. For a builtin icon, `gdk_icon_info_get_filename()` returns `NULL` and you need to call `gdk_icon_info_get_builtin_pixbuf()`.

GTK_ICON_THEME_ERROR

```
#define GTK_ICON_THEME_ERROR gtk_icon_theme_error_quark ()
```

---

enum GtkIconThemeError

```
typedef enum {
  GTK_ICON_THEME_NOT_FOUND,
  GTK_ICON_THEME_FAILED
} GtkIconThemeError;
```

Error codes for GtkIconTheme operations.

`GTK_ICON_THEME_NOT_FOUND`	The icon specified does not exist in the theme
`GTK_ICON_THEME_FAILED`	An unspecified error occurred.

---

gtk_icon_theme_new ()

```
GtkIconTheme* gtk_icon_theme_new           (void);
```

Creates a new icon theme object. Icon theme objects are used to lookup up an icon by name in a particular icon theme. Usually, you'll want to use `gtk_icon_theme_get_default()` or `gtk_icon_theme_get_for_screen()` rather than creating a new icon theme object for scratch.

Returns : the newly created GtkIconTheme object.

Since 2.4

---

gtk_icon_theme_get_default ()

```
GtkIconTheme* gtk_icon_theme_get_default    (void);
```

Gets the icon theme for the default screen. See gtk_icon_theme_get_for_screen().

Returns : A unique GtkIconTheme associated with the default screen. This icon theme is associated with the screen and can be used as long as the screen is open.

Since 2.4

---

gtk_icon_theme_get_for_screen ()

```
GtkIconTheme* gtk_icon_theme_get_for_screen (GdkScreen *screen);
```

Gets the icon theme object associated with *screen*; if this function has not previously been called for the given screen, a new icon theme object will be created and associated with the screen. Icon theme objects are fairly expensive to create, so using this function is usually a better choice than calling than gtk_icon_theme_new() and setting the screen yourself; by using this function a single icon theme object will be shared between users.

screen : a GdkScreen

Returns : A unique GtkIconTheme associated with the given screen. This icon theme is associated with the screen and can be used as long as the screen is open.

Since 2.4

---

gtk_icon_theme_set_screen ()

```
void        gtk_icon_theme_set_screen       (GtkIconTheme *icon_theme,
                                             GdkScreen *screen);
```

Sets the screen for an icon theme; the screen is used to track the user's currently configured icon theme,

which might be different for different screens.

icon_theme : a GtkIconTheme
screen : a GdkScreen

Since 2.4

---

gtk_icon_theme_set_search_path ()

```
void        gtk_icon_theme_set_search_path  (GtkIconTheme *icon_theme,
                                             const gchar *path[],
                                             gint n_elements);
```

Sets the search path for the icon theme object. When looking for an icon theme, GTK+ will search for a subdirectory of one or more of the directories in *path* with the same name as the icon theme. (Themes from multiple of the path elements are combined to allow themes to be extended by adding icons in the user's home directory.)

In addition if an icon found isn't found either in the current icon theme or the default icon theme, and an image file with the right name is found directly in one of the elements of *path*, then that image will be used for the icon name. (This is legacy feature, and new icons should be put into the default icon theme, which is called DEFAULT_THEME_NAME, rather than directly on the icon path.)

icon_theme : a GtkIconTheme
path : array of directories that are searched for icon themes
n_elements : number of elements in *path*.

Since 2.4

---

gtk_icon_theme_get_search_path ()

```
void        gtk_icon_theme_get_search_path  (GtkIconTheme *icon_theme,
                                             gchar **path[],
```

```
                                                gint *n_elements);
```

Gets the current search path. See gtk_icon_theme_set_search_path().

icon_theme : a GtkIconTheme

path : location to store a list of icon theme path directories or NULL The stored value should be freed with g_strfreev().

n_elements : location to store number of elements in *path*, or NULL

Since 2.4

---

gtk_icon_theme_append_search_path ()

```
void        gtk_icon_theme_append_search_path
                                                (GtkIconTheme *icon_theme,
                                                 const gchar *path);
```

Appends a directory to the search path. See gtk_icon_theme_set_search_path().

icon_theme : a GtkIconTheme

path : directory name to append to the icon path

Since 2.4

---

gtk_icon_theme_prepend_search_path ()

```
void        gtk_icon_theme_prepend_search_path
                                                (GtkIconTheme *icon_theme,
                                                 const gchar *path);
```

Prepends a directory to the search path. See gtk_icon_theme_set_search_path().

icon_theme : a GtkIconTheme
path : directory name to prepend to the icon path

Since 2.4

---

gtk_icon_theme_set_custom_theme ()

```
void        gtk_icon_theme_set_custom_theme (GtkIconTheme *icon_theme,
                                             const gchar *theme_name);
```

Sets the name of the icon theme that the GtkIconTheme object uses overriding system configuration. This function cannot be called on the icon theme objects returned from `gtk_icon_theme_get_default()` and `gtk_icon_theme_get_default()`.

icon_theme : a GtkIconTheme
theme_name : name of icon theme to use instead of configured theme

Since 2.4

---

gtk_icon_theme_has_icon ()

```
gboolean    gtk_icon_theme_has_icon         (GtkIconTheme *icon_theme,
                                             const gchar *icon_name);
```

Checks whether an icon theme includes an icon for a particular name.

icon_theme : a GtkIconTheme
icon_name : the name of an icon
Returns : TRUE if *icon_theme* includes an icon for *icon_name*.

Since 2.4

---

gtk_icon_theme_lookup_icon ()

```
GtkIconInfo* gtk_icon_theme_lookup_icon     (GtkIconTheme *icon_theme,
                                             const gchar *icon_name,
                                             gint size,
                                             GtkIconLookupFlags flags);
```

Looks up a named icon and returns a structure containing information such as the filename of the icon. The icon can then be rendered into a pixbuf using `gtk_icon_info_load_icon()`. (`gtk_icon_theme_load_icon()` combines these two steps if all you need is the pixbuf.)

icon_theme :	a GtkIconTheme
icon_name :	the name of the icon to lookup
size :	desired icon size
flags :	flags modifying the behavior of the icon lookup
Returns :	a GtkIconInfo structure containing information about the icon, or `NULL` if the icon wasn't found. Free with `gtk_icon_info_free()`

Since 2.4

---

gtk_icon_theme_load_icon ()

```
GdkPixbuf*  gtk_icon_theme_load_icon        (GtkIconTheme *icon_theme,
                                             const gchar *icon_name,
                                             gint size,
                                             GtkIconLookupFlags flags,
                                             GError **error);
```

Looks up an icon in an icon theme, scales it to the given size and renders it into a pixbuf. This is a convenience function; if more details about the icon are needed, use

gtk_icon_theme_lookup_icon() followed by gtk_icon_info_load_icon().

icon_theme :	a GtkIconTheme
icon_name :	the name of the icon to lookup
size :	the desired icon size. The resulting icon may not be exactly this size; see gtk_icon_info_load_icon().
flags :	flags modifying the behavior of the icon lookup
error :	Location to store error information on failure, or NULL.
Returns :	the rendered icon; this may be a newly created icon or a new reference to an internal icon, so you must not modify the icon. Use g_object_unref() to release your reference to the icon. NULL if the icon isn't found.

Since 2.4

---

gtk_icon_theme_list_icons ()

```
GList*      gtk_icon_theme_list_icons       (GtkIconTheme *icon_theme,
                                             const gchar *context);
```

Lists the icons in the current icon theme. Only a subset of the icons can be listed by providing a context string. The set of values for the context string is system dependent, but will typically include such values as 'apps' and 'mimetypes'.

icon_theme :	a GtkIconTheme
context :	a string identifying a particular type of icon, or NULL to list all icons.
Returns :	a GList list holding the names of all the icons in the theme. You must first free each element in the list with g_free(), then free the list itself with g_list_free().

Since 2.4

---

gtk_icon_theme_get_icon_sizes ()

```
gint*       gtk_icon_theme_get_icon_sizes   (GtkIconTheme *icon_theme,
                                             const gchar *icon_name);
```

Returns an array of integers describing the sizes at which the icon is available without scaling. A size of -1 means that the icon is available in a scalable format. The array is zero-terminated.

icon_theme : a GtkIconTheme
icon_name : the name of an icon
Returns : An newly allocated array describing the sizes at which the icon is available.

Since 2.6

---

gtk_icon_theme_get_example_icon_name ()

```
char*       gtk_icon_theme_get_example_icon_name
                                            (GtkIconTheme *icon_theme);
```

Gets the name of an icon that is representative of the current theme (for instance, to use when presenting a list of themes to the user.)

icon_theme : a GtkIconTheme
Returns : the name of an example icon or NULL. Free with g_free().

Since 2.4

---

gtk_icon_theme_rescan_if_needed ()

```
gboolean    gtk_icon_theme_rescan_if_needed (GtkIconTheme *icon_theme);
```

Checks to see if the icon theme has changed; if it has, any currently cached information is discarded and will be reloaded next time *icon_theme* is accessed.

icon_theme : a GtkIconTheme

Returns : TRUE if the icon theme has changed and needed to be reloaded.

Since 2.4

---

gtk_icon_theme_add_builtin_icon ()

```
void        gtk_icon_theme_add_builtin_icon (const gchar *icon_name,
                                             gint size,
                                             GdkPixbuf *pixbuf);
```

Registers a built-in icon for icon theme lookups. The idea of built-in icons is to allow an application or library that uses themed icons to function requiring files to be present in the file system. For instance, the default images for all of GTK+'s stock icons are registered as built-icons.

In general, if you use gtk_icon_theme_add_builtin_icon() you should also install the icon in the icon theme, so that the icon is generally available.

This function will generally be used with pixbufs loaded via gdk_pixbuf_new_from_inline().

icon_name : the name of the icon to register

size : the size at which to register the icon (different images can be registered for the same icon name at different sizes.)

pixbuf : GdkPixbuf that contains the image to use for *icon_name*.

Since 2.4

---

gtk_icon_info_copy ()

```
GtkIconInfo* gtk_icon_info_copy             (GtkIconInfo *icon_info);
```

Make a copy of a GtkIconInfo.

icon_info : a GtkIconInfo

Returns : the new GtkIconInfo

Since 2.4

---

gtk_icon_info_free ()

```
void        gtk_icon_info_free              (GtkIconInfo *icon_info);
```

Free a GtkIconInfo and associated information

icon_info : a GtkIconInfo

Since 2.4

---

gtk_icon_info_get_base_size ()

```
gint        gtk_icon_info_get_base_size     (GtkIconInfo *icon_info);
```

Gets the base size for the icon. The base size is a size for the icon that was specified by the icon theme creator. This may be different than the actual size of image; an example of this is small emblem icons that can be attached to a larger icon. These icons will be given the same base size as the larger icons to which they are attached.

icon_info : a GtkIconInfo

Returns : the base size, or 0, if no base size is known for the icon.

Since 2.4

---

gtk_icon_info_get_filename ()

```
G_CONST_RETURN gchar* gtk_icon_info_get_filename
                                                (GtkIconInfo *icon_info);
```

Gets the filename for the icon. If the `GTK_ICON_LOOKUP_USE_BUILTIN` flag was passed to `gtk_icon_theme_lookup_icon()`, there may be no filename if a builtin icon is returned; in this case, you should use `gtk_icon_info_get_builtin_pixbuf()`.

`icon_info` : a GtkIconInfo

Returns : the filename for the icon, or `NULL` if `gtk_icon_info_get_builtin_pixbuf()` should be used instead. The return value is owned by GTK+ and should not be modified or freed.

Since 2.4

---

gtk_icon_info_get_builtin_pixbuf ()

```
GdkPixbuf*  gtk_icon_info_get_builtin_pixbuf
                                                (GtkIconInfo *icon_info);
```

Gets the built-in image for this icon, if any. To allow GTK+ to use built in icon images, you must pass the `GTK_ICON_LOOKUP_USE_BUILTIN` to `gtk_icon_theme_lookup_icon()`.

`icon_info` : a GtkIconInfo structure

Returns : the built-in image pixbuf, or `NULL`. No extra reference is added to the returned pixbuf, so if you want to keep it around, you must use `g_object_ref()`. The returned image must not be modified.

Since 2.4

---

gtk_icon_info_load_icon ()

```
GdkPixbuf*  gtk_icon_info_load_icon         (GtkIconInfo *icon_info,
                                             GError **error);
```

Renders an icon previously looked up in an icon theme using gtk_icon_theme_lookup_icon(); the size will be based on the size passed to gtk_icon_theme_lookup_icon(). Note that the resulting pixbuf may not be exactly this size; an icon theme may have icons that differ slightly from their nominal sizes, and in addition GTK+ will avoid scaling icons that it considers sufficiently close to the requested size or for which the source image would have to be scaled up too far. (This maintains sharpness.)

icon_info : a GtkIconInfo structure from gtk_icon_theme_lookup_icon()

error :

Returns : the rendered icon; this may be a newly created icon or a new reference to an internal icon, so you must not modify the icon. Use g_object_unref() to release your reference to the icon.

Since 2.4

---

gtk_icon_info_set_raw_coordinates ()

```
void        gtk_icon_info_set_raw_coordinates
                                            (GtkIconInfo *icon_info,
                                             gboolean raw_coordinates);
```

Sets whether the coordinates returned by gtk_icon_info_get_embedded_rect() and gtk_icon_info_get_attach_points() should be returned in their original form as specified in the icon theme, instead of scaled appropriately for the pixbuf returned by gtk_icon_info_load_icon().

Raw coordinates are somewhat strange; they are specified to be with respect to the unscaled pixmap for PNG and XPM icons, but for SVG icons, they are in a 1000x1000 coordinate space that is scaled to the final size of the icon. You can determine if the icon is an SVG icon by using gtk_icon_info_get_filename(), and seeing if it is non-NULL and ends in '.svg'.

This function is provided primarily to allow compatibility wrappers for older API's, and is not expected to be useful for applications.

icon_info :	a GtkIconInfo
raw_coordinates :	whether the coordinates of embedded rectangles and attached points should be returned in their original (unscaled) form.

Since 2.4

---

gtk_icon_info_get_embedded_rect ()

```
gboolean    gtk_icon_info_get_embedded_rect (GtkIconInfo *icon_info,
                                             GdkRectangle *rectangle);
```

Gets the coordinates of a rectangle within the icon that can be used for display of information such as a preview of the contents of a text file. See gtk_icon_info_set_raw_coordinates() for further information about the coordinate system.

icon_info :	a GtkIconInfo
rectangle :	GdkRectangle in which to store embedded rectangle coordinates; coordinates are only stored when this function returns TRUE.
Returns :	TRUE if the icon has an embedded rectangle

Since 2.4

---

gtk_icon_info_get_attach_points ()

```
gboolean    gtk_icon_info_get_attach_points (GtkIconInfo *icon_info,
                                             GdkPoint **points,
                                             gint *n_points);
```

Fetches the set of attach points for an icon. An attach point is a location in the icon that can be used as

anchor points for attaching emblems or overlays to the icon.

icon_info :	a GtkIconInfo
points :	location to store pointer to an array of points, or NULL free the array of points with g_free().
n_points :	location to store the number of points in *points*, or NULL
Returns :	TRUE if there are any attach points for the icon.

Since 2.4

---

gtk_icon_info_get_display_name ()

```
G_CONST_RETURN gchar* gtk_icon_info_get_display_name
                                                (GtkIconInfo *icon_info);
```

Gets the display name for an icon. A display name is a string to be used in place of the icon name in a user visible context like a list of icons.

icon_info :	a GtkIconInfo
Returns :	the display name for the icon or NULL, if the icon doesn't have a specified display name. This value is owned *icon_info* and must not be modified or free.

Since 2.4

Signals

The "changed" signal

```
void        user_function                   (GtkIconTheme *icon_theme,
                                             gpointer user_data);
```

Emitted when the current icon theme is switched or GTK+ detects that a change has occurred in the

contents of the current icon theme.

icon_theme : the icon theme
user_data : user data set when the signal handler was connected.

<< Drag and Drop **Stock Items >>**

Stock Items

Stock Items — Prebuilt common menu/toolbar items and corresponding icons

Synopsis

```
#include <gtk/gtk.h>

            GtkStockItem;
void        gtk_stock_add                   (const GtkStockItem *items,
                                             guint n_items);
void        gtk_stock_add_static            (const GtkStockItem *items,
                                             guint n_items);
GtkStockItem* gtk_stock_item_copy           (const GtkStockItem *item);
void        gtk_stock_item_free             (GtkStockItem *item);
GSList*     gtk_stock_list_ids              (void);
gboolean    gtk_stock_lookup                (const gchar *stock_id,
                                             GtkStockItem *item);

#define     GTK_STOCK_ABOUT
#define     GTK_STOCK_ADD
#define     GTK_STOCK_APPLY
#define     GTK_STOCK_BOLD
#define     GTK_STOCK_CANCEL
#define     GTK_STOCK_CDROM
#define     GTK_STOCK_CLEAR
#define     GTK_STOCK_CLOSE
#define     GTK_STOCK_COLOR_PICKER
#define     GTK_STOCK_CONVERT
#define     GTK_STOCK_CONNECT
#define     GTK_STOCK_COPY
#define     GTK_STOCK_CUT
#define     GTK_STOCK_DELETE
```

```
#define     GTK_STOCK_DIALOG_AUTHENTICATION
#define     GTK_STOCK_DIALOG_ERROR
#define     GTK_STOCK_DIALOG_INFO
#define     GTK_STOCK_DIALOG_QUESTION
#define     GTK_STOCK_DIALOG_WARNING
#define     GTK_STOCK_DIRECTORY
#define     GTK_STOCK_DISCONNECT
#define     GTK_STOCK_DND
#define     GTK_STOCK_DND_MULTIPLE
#define     GTK_STOCK_EDIT
#define     GTK_STOCK_EXECUTE
#define     GTK_STOCK_FILE
#define     GTK_STOCK_FIND
#define     GTK_STOCK_FIND_AND_REPLACE
#define     GTK_STOCK_FLOPPY
#define     GTK_STOCK_GOTO_BOTTOM
#define     GTK_STOCK_GOTO_FIRST
#define     GTK_STOCK_GOTO_LAST
#define     GTK_STOCK_GOTO_TOP
#define     GTK_STOCK_GO_BACK
#define     GTK_STOCK_GO_DOWN
#define     GTK_STOCK_GO_FORWARD
#define     GTK_STOCK_GO_UP
#define     GTK_STOCK_HARDDISK
#define     GTK_STOCK_HELP
#define     GTK_STOCK_HOME
#define     GTK_STOCK_INDENT
#define     GTK_STOCK_INDEX
#define     GTK_STOCK_ITALIC
#define     GTK_STOCK_JUMP_TO
#define     GTK_STOCK_JUSTIFY_CENTER
#define     GTK_STOCK_JUSTIFY_FILL
#define     GTK_STOCK_JUSTIFY_LEFT
#define     GTK_STOCK_JUSTIFY_RIGHT
#define     GTK_STOCK_MEDIA_FORWARD
#define     GTK_STOCK_MEDIA_NEXT
#define     GTK_STOCK_MEDIA_PAUSE
#define     GTK_STOCK_MEDIA_PLAY
#define     GTK_STOCK_MEDIA_PREVIOUS
#define     GTK_STOCK_MEDIA_RECORD
```

```
#define     GTK_STOCK_MEDIA_REWIND
#define     GTK_STOCK_MEDIA_STOP
#define     GTK_STOCK_MISSING_IMAGE
#define     GTK_STOCK_NETWORK
#define     GTK_STOCK_NEW
#define     GTK_STOCK_NO
#define     GTK_STOCK_OK
#define     GTK_STOCK_OPEN
#define     GTK_STOCK_PASTE
#define     GTK_STOCK_PREFERENCES
#define     GTK_STOCK_PRINT
#define     GTK_STOCK_PRINT_PREVIEW
#define     GTK_STOCK_PROPERTIES
#define     GTK_STOCK_QUIT
#define     GTK_STOCK_REDO
#define     GTK_STOCK_REFRESH
#define     GTK_STOCK_REMOVE
#define     GTK_STOCK_REVERT_TO_SAVED
#define     GTK_STOCK_SAVE
#define     GTK_STOCK_SAVE_AS
#define     GTK_STOCK_SELECT_COLOR
#define     GTK_STOCK_SELECT_FONT
#define     GTK_STOCK_SORT_ASCENDING
#define     GTK_STOCK_SORT_DESCENDING
#define     GTK_STOCK_SPELL_CHECK
#define     GTK_STOCK_STOP
#define     GTK_STOCK_STRIKETHROUGH
#define     GTK_STOCK_UNDELETE
#define     GTK_STOCK_UNDERLINE
#define     GTK_STOCK_UNDO
#define     GTK_STOCK_UNINDENT
#define     GTK_STOCK_YES
#define     GTK_STOCK_ZOOM_100
#define     GTK_STOCK_ZOOM_FIT
#define     GTK_STOCK_ZOOM_IN
#define     GTK_STOCK_ZOOM_OUT
```

Description

Stock items represent commonly-used menu or toolbar items such as "Open" or "Exit". Each stock item is identified by a stock ID; stock IDs are just strings, but macros such as GTK_STOCK_OPEN are provided to avoid typing mistakes in the strings. Applications can register their own stock items in addition to those built-in to GTK+.

Each stock ID can be associated with a GtkStockItem, which contains the user-visible label, keyboard accelerator, and translation domain of the menu or toolbar item; and/or with an icon stored in a GtkIconFactory. See GtkIconFactory for more information on stock icons. The connection between a GtkStockItem and stock icons is purely conventional (by virtue of using the same stock ID); it's possible to register a stock item but no icon, and vice versa. Stock icons may have a RTL variant which gets used for right-to-left locales.

Details

GtkStockItem

```
typedef struct {
  gchar *stock_id;
  gchar *label;
  GdkModifierType modifier;
  guint keyval;
  gchar *translation_domain;
} GtkStockItem;
```

---

gtk_stock_add ()

```
void        gtk_stock_add                       (const GtkStockItem *items,
                                                 guint n_items);
```

Registers each of the stock items in *items*. If an item already exists with the same stock ID as one of the *items*, the old item gets replaced. The stock items are copied, so GTK+ does not hold any pointer into *items* and *items* can be freed. Use gtk_stock_add_static() if *items* is persistent and GTK+ need not copy the array.

items : a GtkStockItem or array of items

n_items : number of GtkStockItem in *items*

---

gtk_stock_add_static ()

```
void        gtk_stock_add_static            (const GtkStockItem *items,
                                             guint n_items);
```

Same as `gtk_stock_add()`, but doesn't copy *items*, so *items* must persist until application exit.

items : a GtkStockItem or array of GtkStockItem
n_items : number of items

---

gtk_stock_item_copy ()

```
GtkStockItem* gtk_stock_item_copy           (const GtkStockItem *item);
```

Copies a stock item, mostly useful for language bindings and not in applications.

item : a GtkStockItem
Returns : a new GtkStockItem

---

gtk_stock_item_free ()

```
void        gtk_stock_item_free             (GtkStockItem *item);
```

Frees a stock item allocated on the heap, such as one returned by `gtk_stock_item_copy()`. Also frees the fields inside the stock item, if they are not `NULL`.

item : a GtkStockItem

---

gtk_stock_list_ids ()

```
GSList*     gtk_stock_list_ids              (void);
```

Retrieves a list of all known stock IDs added to a GtkIconFactory or registered with `gtk_stock_add()`. The list must be freed with `g_slist_free()`, and each string in the list must be freed with `g_free()`.

Returns : a list of known stock IDs

---

gtk_stock_lookup ()

```
gboolean    gtk_stock_lookup                (const gchar *stock_id,
                                             GtkStockItem *item);
```

Fills *`item`* with the registered values for *`stock_id`*, returning TRUE if *`stock_id`* was known.

`stock_id` : a stock item name
`item` : stock item to initialize with values
Returns : TRUE if *`item`* was initialized

GTK_STOCK_ABOUT

```
#define GTK_STOCK_ABOUT             "gtk-about"
```

The "About" item.

Since 2.6

GTK_STOCK_ADD

```
#define GTK_STOCK_ADD                "gtk-add"
```

The "Add" item. 

GTK_STOCK_APPLY

```
#define GTK_STOCK_APPLY              "gtk-apply"
```

The "Apply" item. 

GTK_STOCK_BOLD

```
#define GTK_STOCK_BOLD               "gtk-bold"
```

The "Bold" item. 

GTK_STOCK_CANCEL

```
#define GTK_STOCK_CANCEL             "gtk-cancel"
```

The "Cancel" item. 

GTK_STOCK_CDROM

```
#define GTK_STOCK_CDROM              "gtk-cdrom"
```

The "CD-Rom" item.

GTK_STOCK_CLEAR

```
#define GTK_STOCK_CLEAR             "gtk-clear"
```

The "Clear" item.

GTK_STOCK_CLOSE

```
#define GTK_STOCK_CLOSE             "gtk-close"
```

The "Close" item.

GTK_STOCK_COLOR_PICKER

```
#define GTK_STOCK_COLOR_PICKER      "gtk-color-picker"
```

The "Color Picker" item.

Since 2.2

GTK_STOCK_CONVERT

```
#define GTK_STOCK_CONVERT           "gtk-convert"
```

The "Convert" item. 

GTK_STOCK_CONNECT

```
#define GTK_STOCK_CONNECT           "gtk-connect"
```

The "Connect" icon. 

Since 2.6

GTK_STOCK_COPY

```
#define GTK_STOCK_COPY              "gtk-copy"
```

The "Copy" item. 

GTK_STOCK_CUT

```
#define GTK_STOCK_CUT               "gtk-cut"
```

The "Cut" item. 

GTK_STOCK_DELETE

```
#define GTK_STOCK_DELETE            "gtk-delete"
```

The "Delete" item.

GTK_STOCK_DIALOG_AUTHENTICATION

```
#define     GTK_STOCK_DIALOG_AUTHENTICATION
```

The "Authentication" item.

Since 2.4

GTK_STOCK_DIALOG_ERROR

```
#define GTK_STOCK_DIALOG_ERROR     "gtk-dialog-error"
```

The "Error" item.

GTK_STOCK_DIALOG_INFO

```
#define GTK_STOCK_DIALOG_INFO      "gtk-dialog-info"
```

The "Information" item.

GTK_STOCK_DIALOG_QUESTION

```
#define GTK_STOCK_DIALOG_QUESTION  "gtk-dialog-question"
```

The "Question" item. 

GTK_STOCK_DIALOG_WARNING

```
#define GTK_STOCK_DIALOG_WARNING   "gtk-dialog-warning"
```

The "Warning" item. 

GTK_STOCK_DIRECTORY

```
#define GTK_STOCK_DIRECTORY        "gtk-directory"
```

The "Directory" icon. 

Since 2.6

GTK_STOCK_DISCONNECT

```
#define GTK_STOCK_DISCONNECT       "gtk-disconnect"
```

The "Disconnect" icon. 

Since 2.6

GTK_STOCK_DND

```
#define GTK_STOCK_DND              "gtk-dnd"
```

The "Drag-And-Drop" icon. 

GTK_STOCK_DND_MULTIPLE

```
#define GTK_STOCK_DND_MULTIPLE     "gtk-dnd-multiple"
```

The "Drag-And-Drop multiple" icon. 

GTK_STOCK_EDIT

```
#define GTK_STOCK_EDIT             "gtk-edit"
```

The "Edit" item. 

Since 2.6

GTK_STOCK_EXECUTE

```
#define GTK_STOCK_EXECUTE          "gtk-execute"
```

The "Execute" item. 

GTK_STOCK_FILE

```
#define GTK_STOCK_FILE             "gtk-file"
```

The "File" icon.

Since 2.6

GTK_STOCK_FIND

```
#define GTK_STOCK_FIND              "gtk-find"
```

The "Find" item.

GTK_STOCK_FIND_AND_REPLACE

```
#define GTK_STOCK_FIND_AND_REPLACE "gtk-find-and-replace"
```

The "Find and Replace" item.

GTK_STOCK_FLOPPY

```
#define GTK_STOCK_FLOPPY            "gtk-floppy"
```

The "Floppy" item.

GTK_STOCK_GOTO_BOTTOM

```
#define GTK_STOCK_GOTO_BOTTOM       "gtk-goto-bottom"
```

The "Bottom" item.

GTK_STOCK_GOTO_FIRST

```
#define GTK_STOCK_GOTO_FIRST       "gtk-goto-first"
```

The "First" item. RTL variant

GTK_STOCK_GOTO_LAST

```
#define GTK_STOCK_GOTO_LAST        "gtk-goto-last"
```

The "Last" item. RTL variant

GTK_STOCK_GOTO_TOP

```
#define GTK_STOCK_GOTO_TOP         "gtk-goto-top"
```

The "Top" item.

GTK_STOCK_GO_BACK

```
#define GTK_STOCK_GO_BACK          "gtk-go-back"
```

The "Back" item. RTL variant

GTK_STOCK_GO_DOWN

```
#define GTK_STOCK_GO_DOWN           "gtk-go-down"
```

The "Down" item.

GTK_STOCK_GO_FORWARD

```
#define GTK_STOCK_GO_FORWARD        "gtk-go-forward"
```

The "Forward" item. RTL variant

GTK_STOCK_GO_UP

```
#define GTK_STOCK_GO_UP             "gtk-go-up"
```

The "Up" item.

GTK_STOCK_HARDDISK

```
#define GTK_STOCK_HARDDISK          "gtk-harddisk"
```

The "Harddisk" item.

Since 2.4

GTK_STOCK_HELP

```
#define GTK_STOCK_HELP              "gtk-help"
```

The "Help" item.

GTK_STOCK_HOME

```
#define GTK_STOCK_HOME              "gtk-home"
```

The "Home" item.

GTK_STOCK_INDENT

```
#define GTK_STOCK_INDENT            "gtk-indent"
```

The "Indent" item.

Since 2.4

GTK_STOCK_INDEX

```
#define GTK_STOCK_INDEX             "gtk-index"
```

The "Index" item.

GTK_STOCK_ITALIC

```
#define GTK_STOCK_ITALIC            "gtk-italic"
```

The "Italic" item.

GTK_STOCK_JUMP_TO

```
#define GTK_STOCK_JUMP_TO           "gtk-jump-to"
```

The "Jump to" item. RTL-variant

GTK_STOCK_JUSTIFY_CENTER

```
#define GTK_STOCK_JUSTIFY_CENTER    "gtk-justify-center"
```

The "Center" item.

GTK_STOCK_JUSTIFY_FILL

```
#define GTK_STOCK_JUSTIFY_FILL      "gtk-justify-fill"
```

The "Fill" item.

GTK_STOCK_JUSTIFY_LEFT

```
#define GTK_STOCK_JUSTIFY_LEFT     "gtk-justify-left"
```

The "Left" item.

GTK_STOCK_JUSTIFY_RIGHT

```
#define GTK_STOCK_JUSTIFY_RIGHT    "gtk-justify-right"
```

The "Right" item.

GTK_STOCK_MEDIA_FORWARD

```
#define GTK_STOCK_MEDIA_FORWARD    "gtk-media-forward"
```

The "Media Forward" item.

Since 2.6

GTK_STOCK_MEDIA_NEXT

```
#define GTK_STOCK_MEDIA_NEXT       "gtk-media-next"
```

The "Media Next" item.

Since 2.6

GTK_STOCK_MEDIA_PAUSE

```
#define GTK_STOCK_MEDIA_PAUSE      "gtk-media-pause"
```

The "Media Pause" item. 

Since 2.6

GTK_STOCK_MEDIA_PLAY

```
#define GTK_STOCK_MEDIA_PLAY       "gtk-media-play"
```

The "Media Play" item. 

Since 2.6

GTK_STOCK_MEDIA_PREVIOUS

```
#define GTK_STOCK_MEDIA_PREVIOUS   "gtk-media-previous"
```

The "Media Previous" item. 

Since 2.6

GTK_STOCK_MEDIA_RECORD

```
#define GTK_STOCK_MEDIA_RECORD     "gtk-media-record"
```

The "Media Record" item. 

Since 2.6

GTK_STOCK_MEDIA_REWIND

```
#define GTK_STOCK_MEDIA_REWIND     "gtk-media-rewind"
```

The "Media Rewind" item.

Since 2.6

GTK_STOCK_MEDIA_STOP

```
#define GTK_STOCK_MEDIA_STOP       "gtk-media-stop"
```

The "Media Stop" item.

Since 2.6

GTK_STOCK_MISSING_IMAGE

```
#define GTK_STOCK_MISSING_IMAGE    "gtk-missing-image"
```

The "Missing image" icon.

GTK_STOCK_NETWORK

```
#define GTK_STOCK_NETWORK          "gtk-network"
```

The "Network" item.

Since 2.4

GTK_STOCK_NEW

```
#define GTK_STOCK_NEW                   "gtk-new"
```

The "New" item.

GTK_STOCK_NO

```
#define GTK_STOCK_NO                    "gtk-no"
```

The "No" item.

GTK_STOCK_OK

```
#define GTK_STOCK_OK                    "gtk-ok"
```

The "OK" item.

GTK_STOCK_OPEN

```
#define GTK_STOCK_OPEN                  "gtk-open"
```

The "Open" item.

GTK_STOCK_PASTE

```
#define GTK_STOCK_PASTE             "gtk-paste"
```

The "Paste" item.

GTK_STOCK_PREFERENCES

```
#define GTK_STOCK_PREFERENCES       "gtk-preferences"
```

The "Preferences" item.

GTK_STOCK_PRINT

```
#define GTK_STOCK_PRINT             "gtk-print"
```

The "Print" item.

GTK_STOCK_PRINT_PREVIEW

```
#define GTK_STOCK_PRINT_PREVIEW     "gtk-print-preview"
```

The "Print Preview" item.

GTK_STOCK_PROPERTIES

```
#define GTK_STOCK_PROPERTIES       "gtk-properties"
```

The "Properties" item.

GTK_STOCK_QUIT

```
#define GTK_STOCK_QUIT             "gtk-quit"
```

The "Quit" item.

GTK_STOCK_REDO

```
#define GTK_STOCK_REDO             "gtk-redo"
```

The "Redo" item. RTL variant

GTK_STOCK_REFRESH

```
#define GTK_STOCK_REFRESH          "gtk-refresh"
```

The "Refresh" item.

GTK_STOCK_REMOVE

```
#define GTK_STOCK_REMOVE            "gtk-remove"
```

The "Remove" item. 

GTK_STOCK_REVERT_TO_SAVED

```
#define GTK_STOCK_REVERT_TO_SAVED  "gtk-revert-to-saved"
```

The "Revert" item.  RTL variant 

GTK_STOCK_SAVE

```
#define GTK_STOCK_SAVE              "gtk-save"
```

The "Save" item. 

GTK_STOCK_SAVE_AS

```
#define GTK_STOCK_SAVE_AS           "gtk-save-as"
```

The "Save As" item. 

GTK_STOCK_SELECT_COLOR

```
#define GTK_STOCK_SELECT_COLOR     "gtk-select-color"
```

The "Color" item.

GTK_STOCK_SELECT_FONT

```
#define GTK_STOCK_SELECT_FONT      "gtk-select-font"
```

The "Font" item.

GTK_STOCK_SORT_ASCENDING

```
#define GTK_STOCK_SORT_ASCENDING   "gtk-sort-ascending"
```

The "Ascending" item.

GTK_STOCK_SORT_DESCENDING

```
#define GTK_STOCK_SORT_DESCENDING  "gtk-sort-descending"
```

The "Descending" item.

GTK_STOCK_SPELL_CHECK

```
#define GTK_STOCK_SPELL_CHECK      "gtk-spell-check"
```

The "Spell Check" item.

GTK_STOCK_STOP

```
#define GTK_STOCK_STOP              "gtk-stop"
```

The "Stop" item.

GTK_STOCK_STRIKETHROUGH

```
#define GTK_STOCK_STRIKETHROUGH     "gtk-strikethrough"
```

The "Strikethrough" item.

GTK_STOCK_UNDELETE

```
#define GTK_STOCK_UNDELETE          "gtk-undelete"
```

The "Undelete" item. RTL variant

GTK_STOCK_UNDERLINE

```
#define GTK_STOCK_UNDERLINE         "gtk-underline"
```

The "Underline" item.

GTK_STOCK_UNDO

```
#define GTK_STOCK_UNDO              "gtk-undo"
```

The "Undo" item. RTL variant

GTK_STOCK_UNINDENT

```
#define GTK_STOCK_UNINDENT          "gtk-unindent"
```

The "Unindent" item.

Since 2.4

GTK_STOCK_YES

```
#define GTK_STOCK_YES               "gtk-yes"
```

The "Yes" item.

GTK_STOCK_ZOOM_100

```
#define GTK_STOCK_ZOOM_100          "gtk-zoom-100"
```

The "Zoom 100%" item.

GTK_STOCK_ZOOM_FIT

```
#define GTK_STOCK_ZOOM_FIT          "gtk-zoom-fit"
```

The "Zoom to Fit" item.

GTK_STOCK_ZOOM_IN

```
#define GTK_STOCK_ZOOM_IN           "gtk-zoom-in"
```

The "Zoom In" item.

GTK_STOCK_ZOOM_OUT

```
#define GTK_STOCK_ZOOM_OUT          "gtk-zoom-out"
```

The "Zoom Out" item.

<< GtkIconTheme **Themeable Stock Images >>**

Themeable Stock Images

Themeable Stock Images — Manipulating stock icons

Synopsis

```
#include <gtk/gtk.h>

            GtkIconSource;
            GtkIconFactory;
            GtkIconSet;
enum        GtkIconSize;
GtkIconSource* gtk_icon_source_copy         (const GtkIconSource *source);
void        gtk_icon_source_free            (GtkIconSource *source);
void        gtk_icon_factory_add            (GtkIconFactory *factory,
                                             const gchar *stock_id,
                                             GtkIconSet *icon_set);
void        gtk_icon_factory_add_default    (GtkIconFactory *factory);
GtkIconSet* gtk_icon_factory_lookup         (GtkIconFactory *factory,
                                             const gchar *stock_id);
GtkIconSet* gtk_icon_factory_lookup_default (const gchar *stock_id);
GtkIconFactory* gtk_icon_factory_new        (void);
void        gtk_icon_factory_remove_default (GtkIconFactory *factory);
void        gtk_icon_set_add_source         (GtkIconSet *icon_set,
                                             const GtkIconSource *source);
GtkIconSet* gtk_icon_set_copy               (GtkIconSet *icon_set);
GtkIconSet* gtk_icon_set_new                (void);
GtkIconSet* gtk_icon_set_new_from_pixbuf    (GdkPixbuf *pixbuf);
GtkIconSet* gtk_icon_set_ref                (GtkIconSet *icon_set);
GdkPixbuf*  gtk_icon_set_render_icon        (GtkIconSet *icon_set,
                                             GtkStyle *style,
                                             GtkTextDirection direction,
                                             GtkStateType state,
                                             GtkIconSize size,
```

```
                                                GtkWidget *widget,
                                                const char *detail);
void        gtk_icon_set_unref                 (GtkIconSet *icon_set);
gboolean    gtk_icon_size_lookup               (GtkIconSize size,
                                                gint *width,
                                                gint *height);
gboolean    gtk_icon_size_lookup_for_settings
                                               (GtkSettings *settings,
                                                GtkIconSize size,
                                                gint *width,
                                                gint *height);
GtkIconSize gtk_icon_size_register             (const gchar *name,
                                                gint width,
                                                gint height);
void        gtk_icon_size_register_alias       (const gchar *alias,
                                                GtkIconSize target);
GtkIconSize gtk_icon_size_from_name            (const gchar *name);
G_CONST_RETURN gchar* gtk_icon_size_get_name
                                               (GtkIconSize size);
void        gtk_icon_set_get_sizes             (GtkIconSet *icon_set,
                                                GtkIconSize **sizes,
                                                gint *n_sizes);
GtkTextDirection gtk_icon_source_get_direction
                                               (const GtkIconSource *source);
gboolean    gtk_icon_source_get_direction_wildcarded
                                               (const GtkIconSource *source);
G_CONST_RETURN gchar* gtk_icon_source_get_filename
                                               (const GtkIconSource *source);
GdkPixbuf*  gtk_icon_source_get_pixbuf         (const GtkIconSource *source);
G_CONST_RETURN gchar* gtk_icon_source_get_icon_name
                                               (const GtkIconSource *source);
GtkIconSize gtk_icon_source_get_size           (const GtkIconSource *source);
gboolean    gtk_icon_source_get_size_wildcarded
                                               (const GtkIconSource *source);
GtkStateType gtk_icon_source_get_state         (const GtkIconSource *source);
gboolean    gtk_icon_source_get_state_wildcarded
                                               (const GtkIconSource *source);
GtkIconSource* gtk_icon_source_new             (void);
void        gtk_icon_source_set_direction      (GtkIconSource *source,
                                                GtkTextDirection direction);
void        gtk_icon_source_set_direction_wildcarded
```

```
                                                (GtkIconSource *source,
                                                 gboolean setting);
void        gtk_icon_source_set_filename        (GtkIconSource *source,
                                                 const gchar *filename);
void        gtk_icon_source_set_pixbuf          (GtkIconSource *source,
                                                 GdkPixbuf *pixbuf);
void        gtk_icon_source_set_icon_name       (GtkIconSource *source,
                                                 const gchar *icon_name);
void        gtk_icon_source_set_size            (GtkIconSource *source,
                                                 GtkIconSize size);
void        gtk_icon_source_set_size_wildcarded
                                                (GtkIconSource *source,
                                                 gboolean setting);
void        gtk_icon_source_set_state           (GtkIconSource *source,
                                                 GtkStateType state);
void        gtk_icon_source_set_state_wildcarded
                                                (GtkIconSource *source,
                                                 gboolean setting);
```

Object Hierarchy

```
  GObject
   +----GtkIconFactory
```

Description

Browse the available stock icons in the list of stock IDs found here. You can also use the gtk-demo application for this purpose.

An icon factory manages a collection of GtkIconSet; a GtkIconSet manages a set of variants of a particular icon (i.e. a GtkIconSet contains variants for different sizes and widget states). Icons in an icon factory are named by a stock ID, which is a simple string identifying the icon. Each GtkStyle has a list of GtkIconFactory derived from the current theme; those icon factories are consulted first when searching for an icon. If the theme doesn't set a particular icon, GTK+ looks for the icon in a list of default icon factories, maintained by `gtk_icon_factory_add_default()` and `gtk_icon_factory_remove_default()`. Applications with icons should add a default icon factory with their icons, which will allow themes to override the icons for the application.

To display an icon, always use `gtk_style_lookup_icon_set()` on the widget that will display the icon, or the convenience function `gtk_widget_render_icon()`. These functions take the theme into account when looking up the icon to use for a given stock ID.

Details

GtkIconSource

```
typedef struct _GtkIconSource GtkIconSource;
```

GtkIconFactory

```
typedef struct _GtkIconFactory GtkIconFactory;
```

GtkIconSet

```
typedef struct _GtkIconSet GtkIconSet;
```

---

enum GtkIconSize

```
typedef enum
{
  GTK_ICON_SIZE_INVALID,
  GTK_ICON_SIZE_MENU,
  GTK_ICON_SIZE_SMALL_TOOLBAR,
  GTK_ICON_SIZE_LARGE_TOOLBAR,
  GTK_ICON_SIZE_BUTTON,
  GTK_ICON_SIZE_DND,
  GTK_ICON_SIZE_DIALOG
} GtkIconSize;
```

gtk_icon_source_copy ()

```
GtkIconSource* gtk_icon_source_copy         (const GtkIconSource *source);
```

Creates a copy of *source*; mostly useful for language bindings.

source : a GtkIconSource

Returns : a new GtkIconSource

gtk_icon_source_free ()

```
void        gtk_icon_source_free            (GtkIconSource *source);
```

Frees a dynamically-allocated icon source, along with its filename, size, and pixbuf fields if those are not NULL.

source : a GtkIconSource

gtk_icon_factory_add ()

```
void        gtk_icon_factory_add            (GtkIconFactory *factory,
                                             const gchar *stock_id,
                                             GtkIconSet *icon_set);
```

Adds the given *icon_set* to the icon factory, under the name *stock_id*. *stock_id* should be namespaced for your application, e.g. "myapp-whatever-icon". Normally applications create a GtkIconFactory, then add it to the list of default factories with gtk_icon_factory_add_default(). Then they pass the *stock_id* to widgets such as GtkImage to display the icon. Themes can provide an icon with the same name (such as "myapp-whatever-icon") to override your application's default icons. If an icon already existed in *factory* for *stock_id*, it is unreferenced and replaced with the new *icon_set*.

factory : a GtkIconFactory

stock_id : icon name
icon_set : icon set

---

gtk_icon_factory_add_default ()

```
void        gtk_icon_factory_add_default    (GtkIconFactory *factory);
```

Adds an icon factory to the list of icon factories searched by `gtk_style_lookup_icon_set()`. This means that, for example, `gtk_image_new_from_stock()` will be able to find icons in *factory*. There will normally be an icon factory added for each library or application that comes with icons. The default icon factories can be overridden by themes.

factory : a GtkIconFactory

---

gtk_icon_factory_lookup ()

```
GtkIconSet* gtk_icon_factory_lookup         (GtkIconFactory *factory,
                                             const gchar *stock_id);
```

Looks up *stock_id* in the icon factory, returning an icon set if found, otherwise NULL. For display to the user, you should use `gtk_style_lookup_icon_set()` on the GtkStyle for the widget that will display the icon, instead of using this function directly, so that themes are taken into account.

factory : a GtkIconFactory
stock_id : an icon name
Returns : icon set of *stock_id*.

---

gtk_icon_factory_lookup_default ()

```
GtkIconSet* gtk_icon_factory_lookup_default (const gchar *stock_id);
```

Looks for an icon in the list of default icon factories. For display to the user, you should use `gtk_style_lookup_icon_set()` on the GtkStyle for the widget that will display the icon, instead of

using this function directly, so that themes are taken into account.

stock_id : an icon name

Returns : a GtkIconSet, or NULL

---

gtk_icon_factory_new ()

```
GtkIconFactory* gtk_icon_factory_new        (void);
```

Creates a new GtkIconFactory. An icon factory manages a collection of GtkIconSets; a GtkIconSet manages a set of variants of a particular icon (i.e. a GtkIconSet contains variants for different sizes and widget states). Icons in an icon factory are named by a stock ID, which is a simple string identifying the icon. Each GtkStyle has a list of GtkIconFactorys derived from the current theme; those icon factories are consulted first when searching for an icon. If the theme doesn't set a particular icon, GTK+ looks for the icon in a list of default icon factories, maintained by `gtk_icon_factory_add_default()` and `gtk_icon_factory_remove_default()`. Applications with icons should add a default icon factory with their icons, which will allow themes to override the icons for the application.

Returns : a new GtkIconFactory

---

gtk_icon_factory_remove_default ()

```
void        gtk_icon_factory_remove_default (GtkIconFactory *factory);
```

Removes an icon factory from the list of default icon factories. Not normally used; you might use it for a library that can be unloaded or shut down.

factory : a GtkIconFactory previously added with `gtk_icon_factory_add_default()`

---

gtk_icon_set_add_source ()

```
void        gtk_icon_set_add_source         (GtkIconSet *icon_set,
                                             const GtkIconSource *source);
```

Icon sets have a list of GtkIconSource, which they use as base icons for rendering icons in different states and sizes. Icons are scaled, made to look insensitive, etc. in `gtk_icon_set_render_icon()`, but GtkIconSet needs base images to work with. The base images and when to use them are described by a GtkIconSource.

This function copies *`source`*, so you can reuse the same source immediately without affecting the icon set.

An example of when you'd use this function: a web browser's "Back to Previous Page" icon might point in a different direction in Hebrew and in English; it might look different when insensitive; and it might change size depending on toolbar mode (small/large icons). So a single icon set would contain all those variants of the icon, and you might add a separate source for each one.

You should nearly always add a "default" icon source with all fields wildcarded, which will be used as a fallback if no more specific source matches. GtkIconSet always prefers more specific icon sources to more generic icon sources. The order in which you add the sources to the icon set does not matter.

`gtk_icon_set_new_from_pixbuf()` creates a new icon set with a default icon source based on the given pixbuf.

`icon_set` : a GtkIconSet
`source` : a GtkIconSource

---

gtk_icon_set_copy ()

```
GtkIconSet* gtk_icon_set_copy                (GtkIconSet *icon_set);
```

Copies *`icon_set`* by value.

`icon_set` : a GtkIconSet
Returns : a new GtkIconSet identical to the first.

---

gtk_icon_set_new ()

```
GtkIconSet* gtk_icon_set_new                 (void);
```

Creates a new GtkIconSet. A GtkIconSet represents a single icon in various sizes and widget states. It can

provide a GdkPixbuf for a given size and state on request, and automatically caches some of the rendered GdkPixbuf objects.

Normally you would use `gtk_widget_render_icon()` instead of using GtkIconSet directly. The one case where you'd use GtkIconSet is to create application-specific icon sets to place in a GtkIconFactory.

Returns : a new GtkIconSet

---

gtk_icon_set_new_from_pixbuf ()

```
GtkIconSet* gtk_icon_set_new_from_pixbuf    (GdkPixbuf *pixbuf);
```

Creates a new GtkIconSet with *pixbuf* as the default/fallback source image. If you don't add any additional GtkIconSource to the icon set, all variants of the icon will be created from *pixbuf*, using scaling, pixelation, etc. as required to adjust the icon size or make the icon look insensitive/prelighted.

pixbuf : a GdkPixbuf

Returns : a new GtkIconSet

---

gtk_icon_set_ref ()

```
GtkIconSet* gtk_icon_set_ref                (GtkIconSet *icon_set);
```

Increments the reference count on *icon_set*.

icon_set : a GtkIconSet.

Returns : *icon_set*.

---

gtk_icon_set_render_icon ()

```
GdkPixbuf*  gtk_icon_set_render_icon        (GtkIconSet *icon_set,
                                             GtkStyle *style,
                                             GtkTextDirection direction,
```

```
                                        GtkStateType state,
                                        GtkIconSize size,
                                        GtkWidget *widget,
                                        const char *detail);
```

Renders an icon using `gtk_style_render_icon()`. In most cases, `gtk_widget_render_icon()` is better, since it automatically provides most of the arguments from the current widget settings. This function never returns `NULL`; if the icon can't be rendered (perhaps because an image file fails to load), a default "missing image" icon will be returned instead.

icon_set :	a GtkIconSet
style :	a GtkStyle associated with *widget*, or `NULL`
direction :	text direction
state :	widget state
size :	icon size. A size of (GtkIconSize)-1 means render at the size of the source and don't scale.
widget :	widget that will display the icon, or `NULL`. The only use that is typically made of this is to determine the appropriate GdkScreen.
detail :	detail to pass to the theme engine, or `NULL`. Note that passing a detail of anything but `NULL` will disable caching.
Returns :	a GdkPixbuf to be displayed

---

gtk_icon_set_unref ()

```
void        gtk_icon_set_unref              (GtkIconSet *icon_set);
```

Decrements the reference count on *icon_set*, and frees memory if the reference count reaches 0.

icon_set : a GtkIconSet

---

gtk_icon_size_lookup ()

```
gboolean    gtk_icon_size_lookup            (GtkIconSize size,
                                             gint *width,
                                             gint *height);
```

Obtains the pixel size of a semantic icon size, possibly modified by user preferences for the default GtkSettings. (See `gtk_icon_size_lookup_for_settings()`.) Normally *`size`* would be GTK_ICON_SIZE_MENU, GTK_ICON_SIZE_BUTTON, etc. This function isn't normally needed, `gtk_widget_render_icon()` is the usual way to get an icon for rendering, then just look at the size of the rendered pixbuf. The rendered pixbuf may not even correspond to the width/height returned by `gtk_icon_size_lookup()`, because themes are free to render the pixbuf however they like, including changing the usual size.

`size` : an icon size
`width` : location to store icon width
`height` : location to store icon height
Returns : TRUE if *`size`* was a valid size

---

gtk_icon_size_lookup_for_settings ()

```
gboolean    gtk_icon_size_lookup_for_settings
                                            (GtkSettings *settings,
                                             GtkIconSize size,
                                             gint *width,
                                             gint *height);
```

Obtains the pixel size of a semantic icon size, possibly modified by user preferences for a particular GtkSettings. Normally *`size`* would be GTK_ICON_SIZE_MENU, GTK_ICON_SIZE_BUTTON, etc. This function isn't normally needed, `gtk_widget_render_icon()` is the usual way to get an icon for rendering, then just look at the size of the rendered pixbuf. The rendered pixbuf may not even correspond to the width/height returned by `gtk_icon_size_lookup()`, because themes are free to render the pixbuf however they like, including changing the usual size.

`settings` : a GtkSettings object, used to determine which set of user preferences to used.
`size` : an icon size
`width` : location to store icon width
`height` : location to store icon height
Returns : TRUE if *`size`* was a valid size

Since 2.2

gtk_icon_size_register ()

```
GtkIconSize gtk_icon_size_register          (const gchar *name,
                                             gint width,
                                             gint height);
```

Registers a new icon size, along the same lines as GTK_ICON_SIZE_MENU, etc. Returns the integer value for the size.

name : name of the icon size
width : the icon width
height : the icon height
Returns : integer value representing the size

---

gtk_icon_size_register_alias ()

```
void        gtk_icon_size_register_alias    (const gchar *alias,
                                             GtkIconSize target);
```

Registers *alias* as another name for *target*. So calling gtk_icon_size_from_name() with *alias* as argument will return *target*.

alias : an alias for *target*
target : an existing icon size

---

gtk_icon_size_from_name ()

```
GtkIconSize gtk_icon_size_from_name         (const gchar *name);
```

Looks up the icon size associated with *name*.

name : the name to look up.

Returns : the icon size with the given name.

---

gtk_icon_size_get_name ()

```
G_CONST_RETURN gchar* gtk_icon_size_get_name
                                                (GtkIconSize size);
```

Gets the canonical name of the given icon size. The returned string is statically allocated and should not be freed.

size : a GtkIconSize.

Returns : the name of the given icon size.

---

gtk_icon_set_get_sizes ()

```
void        gtk_icon_set_get_sizes          (GtkIconSet *icon_set,
                                             GtkIconSize **sizes,
                                             gint *n_sizes);
```

Obtains a list of icon sizes this icon set can render. The returned array must be freed with g_free().

icon_set : a GtkIconSet

sizes : return location for array of sizes

n_sizes : location to store number of elements in returned array

---

gtk_icon_source_get_direction ()

```
GtkTextDirection gtk_icon_source_get_direction
                                                (const GtkIconSource *source);
```

Obtains the text direction this icon source applies to. The return value is only useful/meaningful if the text direction is *not* wildcarded.

source : a GtkIconSource

Returns : text direction this source matches

---

gtk_icon_source_get_direction_wildcarded ()

```
gboolean    gtk_icon_source_get_direction_wildcarded
                                            (const GtkIconSource *source);
```

Gets the value set by gtk_icon_source_set_direction_wildcarded().

source : a GtkIconSource

Returns : TRUE if this icon source is a base for any text direction variant

---

gtk_icon_source_get_filename ()

```
G_CONST_RETURN gchar* gtk_icon_source_get_filename
                                            (const GtkIconSource *source);
```

Retrieves the source filename, or NULL if none is set. The filename is not a copy, and should not be modified or expected to persist beyond the lifetime of the icon source.

source : a GtkIconSource

Returns : image filename. This string must not be modified or freed.

---

gtk_icon_source_get_pixbuf ()

```
GdkPixbuf*  gtk_icon_source_get_pixbuf      (const GtkIconSource *source);
```

Retrieves the source pixbuf, or NULL if none is set. In addition, if a filename source is in use, this function in some cases will return the pixbuf from loaded from the filename. This is, for example, true for the GtkIconSource passed to the GtkStyle::render_icon() virtual function. The reference count on the pixbuf is not incremented.

source : a GtkIconSource

Returns : source pixbuf

---

gtk_icon_source_get_icon_name ()

```
G_CONST_RETURN gchar* gtk_icon_source_get_icon_name
                                                (const GtkIconSource *source);
```

Retrieves the source icon name, or NULL if none is set. The icon_name is not a copy, and should not be modified or expected to persist beyond the lifetime of the icon source.

source : a GtkIconSource

Returns : icon name. This string must not be modified or freed.

---

gtk_icon_source_get_size ()

```
GtkIconSize gtk_icon_source_get_size        (const GtkIconSource *source);
```

Obtains the icon size this source applies to. The return value is only useful/meaningful if the icon size is *not* wildcarded.

source : a GtkIconSource

Returns : icon size this source matches.

---

gtk_icon_source_get_size_wildcarded ()

```
gboolean    gtk_icon_source_get_size_wildcarded
                                                (const GtkIconSource *source);
```

Gets the value set by gtk_icon_source_set_size_wildcarded().

source : a GtkIconSource

Returns : TRUE if this icon source is a base for any icon size variant

---

gtk_icon_source_get_state ()

```
GtkStateType gtk_icon_source_get_state      (const GtkIconSource *source);
```

Obtains the widget state this icon source applies to. The return value is only useful/meaningful if the widget state is *not* wildcarded.

source : a GtkIconSource

Returns : widget state this source matches

---

gtk_icon_source_get_state_wildcarded ()

```
gboolean    gtk_icon_source_get_state_wildcarded
                                            (const GtkIconSource *source);
```

Gets the value set by gtk_icon_source_set_state_wildcarded().

source : a GtkIconSource

Returns : TRUE if this icon source is a base for any widget state variant

---

gtk_icon_source_new ()

```
GtkIconSource* gtk_icon_source_new          (void);
```

Creates a new GtkIconSource. A GtkIconSource contains a GdkPixbuf (or image filename) that serves as the base image for one or more of the icons in a GtkIconSet, along with a specification for which icons in the icon set will be based on that pixbuf or image file. An icon set contains a set of icons that represent "the same" logical concept in different states, different global text directions, and different sizes.

So for example a web browser's "Back to Previous Page" icon might point in a different direction in Hebrew and

in English; it might look different when insensitive; and it might change size depending on toolbar mode (small/large icons). So a single icon set would contain all those variants of the icon. GtkIconSet contains a list of GtkIconSource from which it can derive specific icon variants in the set.

In the simplest case, GtkIconSet contains one source pixbuf from which it derives all variants. The convenience function `gtk_icon_set_new_from_pixbuf()` handles this case; if you only have one source pixbuf, just use that function.

If you want to use a different base pixbuf for different icon variants, you create multiple icon sources, mark which variants they'll be used to create, and add them to the icon set with `gtk_icon_set_add_source()`.

By default, the icon source has all parameters wildcarded. That is, the icon source will be used as the base icon for any desired text direction, widget state, or icon size.

Returns : a new GtkIconSource

---

gtk_icon_source_set_direction ()

```
void        gtk_icon_source_set_direction   (GtkIconSource *source,
                                             GtkTextDirection direction);
```

Sets the text direction this icon source is intended to be used with.

Setting the text direction on an icon source makes no difference if the text direction is wildcarded. Therefore, you should usually call `gtk_icon_source_set_direction_wildcarded()` to un-wildcard it in addition to calling this function.

source : a GtkIconSource

direction : text direction this source applies to

---

gtk_icon_source_set_direction_wildcarded ()

```
void        gtk_icon_source_set_direction_wildcarded
                                            (GtkIconSource *source,
                                             gboolean setting);
```

If the text direction is wildcarded, this source can be used as the base image for an icon in any GtkTextDirection. If the text direction is not wildcarded, then the text direction the icon source applies to should be set with `gtk_icon_source_set_direction()`, and the icon source will only be used with that text direction.

GtkIconSet prefers non-wildcarded sources (exact matches) over wildcarded sources, and will use an exact match when possible.

source : a GtkIconSource
setting : TRUE to wildcard the text direction

---

gtk_icon_source_set_filename ()

```
void        gtk_icon_source_set_filename    (GtkIconSource *source,
                                             const gchar *filename);
```

Sets the name of an image file to use as a base image when creating icon variants for GtkIconSet. The filename must be absolute.

source : a GtkIconSource
filename : image file to use

---

gtk_icon_source_set_pixbuf ()

```
void        gtk_icon_source_set_pixbuf      (GtkIconSource *source,
                                             GdkPixbuf *pixbuf);
```

Sets a pixbuf to use as a base image when creating icon variants for GtkIconSet.

source : a GtkIconSource
pixbuf : pixbuf to use as a source

---

gtk_icon_source_set_icon_name ()

```
void        gtk_icon_source_set_icon_name   (GtkIconSource *source,
                                             const gchar *icon_name);
```

Sets the name of an icon to look up in the current icon theme to use as a base image when creating icon variants for GtkIconSet.

source : a GtkIconSource
icon_name : name of icon to use

---

gtk_icon_source_set_size ()

```
void        gtk_icon_source_set_size        (GtkIconSource *source,
                                             GtkIconSize size);
```

Sets the icon size this icon source is intended to be used with.

Setting the icon size on an icon source makes no difference if the size is wildcarded. Therefore, you should usually call `gtk_icon_source_set_size_wildcarded()` to un-wildcard it in addition to calling this function.

source : a GtkIconSource
size : icon size this source applies to

---

gtk_icon_source_set_size_wildcarded ()

```
void        gtk_icon_source_set_size_wildcarded
                                            (GtkIconSource *source,
                                             gboolean setting);
```

If the icon size is wildcarded, this source can be used as the base image for an icon of any size. If the size is not wildcarded, then the size the source applies to should be set with `gtk_icon_source_set_size()` and the icon source will only be used with that specific size.

GtkIconSet prefers non-wildcarded sources (exact matches) over wildcarded sources, and will use an exact match when possible.

GtkIconSet will normally scale wildcarded source images to produce an appropriate icon at a given size, but will not change the size of source images that match exactly.

source : a GtkIconSource
setting : TRUE to wildcard the widget state

---

gtk_icon_source_set_state ()

```
void        gtk_icon_source_set_state       (GtkIconSource *source,
                                             GtkStateType state);
```

Sets the widget state this icon source is intended to be used with.

Setting the widget state on an icon source makes no difference if the state is wildcarded. Therefore, you should usually call `gtk_icon_source_set_state_wildcarded()` to un-wildcard it in addition to calling this function.

source : a GtkIconSource
state : widget state this source applies to

---

gtk_icon_source_set_state_wildcarded ()

```
void        gtk_icon_source_set_state_wildcarded
                                            (GtkIconSource *source,
                                             gboolean setting);
```

If the widget state is wildcarded, this source can be used as the base image for an icon in any GtkStateType. If the widget state is not wildcarded, then the state the source applies to should be set with `gtk_icon_source_set_state()` and the icon source will only be used with that specific state.

GtkIconSet prefers non-wildcarded sources (exact matches) over wildcarded sources, and will use an exact match when possible.

GtkIconSet will normally transform wildcarded source images to produce an appropriate icon for a given state, for example lightening an image on prelight, but will not modify source images that match exactly.

source: a GtkIconSource

setting: TRUE to wildcard the widget state

<< Stock Items **Resource Files >>**

Resource Files

Resource Files — Routines for handling resource files

Synopsis

```
#include <gtk/gtk.h>

            GtkRcStyle;
enum        GtkRcFlags;
enum        GtkRcTokenType;
GScanner*   gtk_rc_scanner_new              (void);
GtkStyle*   gtk_rc_get_style                (GtkWidget *widget);
GtkStyle*   gtk_rc_get_style_by_paths       (GtkSettings *settings,
                                             const char *widget_path,
                                             const char *class_path,
                                             GType type);
void        gtk_rc_add_widget_name_style    (GtkRcStyle *rc_style,
                                             const gchar *pattern);
void        gtk_rc_add_widget_class_style   (GtkRcStyle *rc_style,
                                             const gchar *pattern);
void        gtk_rc_add_class_style          (GtkRcStyle *rc_style,
                                             const gchar *pattern);
void        gtk_rc_parse                    (const gchar *filename);
void        gtk_rc_parse_string             (const gchar *rc_string);
gboolean    gtk_rc_reparse_all              (void);
gboolean    gtk_rc_reparse_all_for_settings (GtkSettings *settings,
                                             gboolean force_load);
void        gtk_rc_reset_styles             (GtkSettings *settings);
void        gtk_rc_add_default_file         (const gchar *filename);
gchar**     gtk_rc_get_default_files        (void);
void        gtk_rc_set_default_files        (gchar **filenames);
guint       gtk_rc_parse_color              (GScanner *scanner,
                                             GdkColor *color);
guint       gtk_rc_parse_state              (GScanner *scanner,
                                             GtkStateType *state);
guint       gtk_rc_parse_priority           (GScanner *scanner,
```

```
                                                GtkPathPriorityType *priority);
gchar*      gtk_rc_find_module_in_path          (const gchar *module_file);
gchar*      gtk_rc_find_pixmap_in_path          (GtkSettings *settings,
                                                 GScanner *scanner,
                                                 const gchar *pixmap_file);
gchar*      gtk_rc_get_module_dir               (void);
gchar*      gtk_rc_get_im_module_path           (void);
gchar*      gtk_rc_get_im_module_file           (void);
gchar*      gtk_rc_get_theme_dir                (void);
GtkRcStyle* gtk_rc_style_new                    (void);
GtkRcStyle* gtk_rc_style_copy                   (GtkRcStyle *orig);
void        gtk_rc_style_ref                    (GtkRcStyle *rc_style);
void        gtk_rc_style_unref                  (GtkRcStyle *rc_style);
```

Object Hierarchy

```
  GObject
   +----GtkRcStyle
```

Description

GTK+ provides resource file mechanism for configuring various aspects of the operation of a GTK+ program at runtime.

Default files

An application can cause GTK+ to parse a specific RC file by calling `gtk_rc_parse()`. In addition to this, certain files will be read at the end of `gtk_init()`. Unless modified, the files looked for will be `<SYSCONFDIR>/gtk-2.0/gtkrc` and `.gtkrc-2.0` in the users home directory. (`<SYSCONFDIR>` defaults to `/usr/local/etc`. It can be changed with the `--prefix` or `--sysconfdir` options when configuring GTK+.) Note that although the filenames contain the version number 2.0, all 2.x versions of GTK+ look for these files.

The set of these *default* files can be retrieved with `gtk_rc_get_default_files()` and modified with `gtk_rc_add_default_file()` and `gtk_rc_set_default_files()`. Additionally, the `GTK2_RC_FILES` environment variable can be set to a G_SEARCHPATH_SEPARATOR_S-separated list of files in order to overwrite the set of default files at runtime.

For each RC file, in addition to the file itself, GTK+ will look for a locale-specific file that will be parsed after the main file. For instance, if `LANG` is set to `ja_JP.ujis`, when loading the default file `~/.gtkrc` then GTK+ looks for `~/.gtkrc.ja_JP` and `~/.gtkrc.ja`, and parses the first of those that exists.

Pathnames and patterns

A resource file defines a number of styles and key bindings and attaches them to particular widgets. The attachment is done by the `widget`, `widget_class`, and `class` declarations. As an example of such a statement:

```
widget "mywindow.*.GtkEntry" style "my-entry-class"
```

attaches the style `"my-entry-class"` to all widgets whose *widget class* matches the *pattern* `"mywindow.*.GtkEntry"`.

The patterns here are given in the standard shell glob syntax. The `"?"` wildcard matches any character, while `"*"` matches zero or more of any character. The three types of matching are against the widget path, the *class path* and the class hierarchy. Both the widget and the class paths consists of a `"."` separated list of all the parents of the widget and the widget itself from outermost to innermost. The difference is that in the widget path, the name assigned by `gtk_widget_set_name()` is used if present, otherwise the class name of the widget, while for the class path, the class name is always used.

So, if you have a GtkEntry named `"myentry"`, inside of a of a window named `"mywindow"`, then the widget path is: `"mwindow.GtkHBox.myentry"` while the class path is: `"GtkWindow.GtkHBox.GtkEntry"`.

Matching against class is a little different. The pattern match is done against all class names in the widgets class hierarchy (not the layout hierarchy) in sequence, so the pattern:

```
class "GtkButton" style "my-style"
```

will match not just GtkButton widgets, but also GtkToggleButton and GtkCheckButton widgets, since those classes derive from GtkButton.

Additionally, a priority can be specified for each pattern, and styles override other styles first by priority, then by pattern type and then by order of specification (later overrides earlier). The priorities that can be specified are (highest to lowest):

```
highest
rc
theme
application
gtk
lowest
```

`rc` is the default for styles read from an RC file, `theme` is the default for styles read from theme RC files,

`application` should be used for styles an application sets up, and `gtk` is used for styles that GTK+ creates internally.

Toplevel declarations

An RC file is a text file which is composed of a sequence of declarations. `'#'` characters delimit comments and the portion of a line after a `'#'` is ignored when parsing an RC file.

The possible toplevel declarations are:

`binding` *`name`* `{ ... }`	Declares a binding set.
`class` *`pattern`* `[ style | binding ][ :` *`priority`* `] name`	Specifies a style or binding set for a particular branch of the inheritance hierarchy.
`include` *`filename`*	Parses another file at this point. If *`filename`* is not an absolute filename, it is searched in the directories of the currently open RC files. GTK+ also tries to load a locale-specific variant of the included file.
`module_path` *`path`*	Sets a path (a list of directories separated by colons) that will be searched for theme engines referenced in RC files.
`pixmap_path` *`path`*	Sets a path (a list of directories separated by colons) that will be searched for pixmaps referenced in RC files.
`style` *`name`* `[ =` *`parent`* `] { ... }`	Declares a style.
`widget` *`pattern`* `[ style | binding ][ :` *`priority`* `] name`	Specifies a style or binding set for a particular group of widgets by matching on the widget pathname.
`widget_class` *`pattern`* `[ style | binding ][ :` *`priority`* `] name`	Specifies a style or binding set for a particular group of widgets by matching on the class pathname.

Styles

A RC style is specified by a `style` declaration in a RC file, and then bound to widgets with a `widget`, `widget_class`, or `class` declaration. All styles applying to a particular widget are composited together with `widget` declarations overriding `widget_class` declarations which, in turn, override `class` declarations. Within each type of declaration, later declarations override earlier ones.

Within a `style` declaration, the possible elements are:

`bg[`*`state`*`] = `*`color`*	Sets the color used for the background of most widgets.
`fg[`*`state`*`] = `*`color`*	Sets the color used for the foreground of most widgets.
`base[`*`state`*`] = `*`color`*	Sets the color used for the background of widgets displaying editable text. This color is used for the background of, among others, GtkText, GtkEntry, GtkList, and GtkCList.
`text[`*`state`*`] = `*`color`*	Sets the color used for foreground of widgets using `base` for the background color.
`xthickness = `*`number`*	Sets the xthickness, which is used for various horizontal padding values in GTK+.
`ythickness = `*`number`*	Sets the ythickness, which is used for various vertical padding values in GTK+.
`bg_pixmap[`*`state`*`] = `*`pixmap`*	Sets a background pixmap to be used in place of the `bg` color (or for GtkText, in place of the `base` color. The special value `"<parent>"` may be used to indicate that the widget should use the same background pixmap as its parent. The special value `"<none>"` may be used to indicate no background pixmap.
`font = `*`font`*	Starting with GTK+ 2.0, the "font" and "fontset" declarations are ignored; use "font_name" declarations instead.
`fontset = `*`font`*	Starting with GTK+ 2.0, the "font" and "fontset" declarations are ignored; use "font_name" declarations instead.
`font_name = `*`font`*	Sets the font for a widget. *`font`* must be a Pango font name, e.g. `"Sans Italic 10"`. For details about Pango font names, see `pango_font_description_from_string()`.
`stock[`*`"stock-id"`*`] = {` *`icon source specifications`* `}`	Defines the icon for a stock item.
`engine `*`"engine"`* `{` *`engine-specific settings`* `}`	Defines the engine to be used when drawing with this style.
`class``::`*`property`* `= `*`value`*	Sets a style property for a widget class.

The colors and background pixmaps are specified as a function of the state of the widget. The states are:

`NORMAL`	A color used for a widget in its normal state.
`ACTIVE`	A variant of the `NORMAL` color used when the widget is in the `GTK_STATE_ACTIVE` state, and also for the trough of a ScrollBar, tabs of a NoteBook other than the current tab and similar areas. Frequently, this should be a darker variant of the `NORMAL` color.
`PRELIGHT`	A color used for widgets in the `GTK_STATE_PRELIGHT` state. This state is the used for Buttons and MenuItems that have the mouse cursor over them, and for their children.

SELECTED	A color used to highlight data selected by the user. for instance, the selected items in a list widget, and the selection in an editable widget.
INSENSITIVE	A color used for the background of widgets that have been set insensitive with gtk_widget_set_sensitive().

Colors can be specified as a string containing a color name (GTK+ knows all names from the X color database /usr/lib/X11/rgb.txt), in one of the hexadecimal forms #rrrrggggbbbb, #rrrgggbbb, #rrggbb, or #rgb, where r, g and b are hex digits, or they can be specified as a triplet { *r, g, b*}, where r, g and b are either integers in the range 0-65535 or floats in the range 0.0-1.0.

In a stock definition, icon sources are specified as a 4-tuple of image filename or icon name, text direction, widget state, and size, in that order. Each icon source specifies an image filename or icon name to use with a given direction, state, and size. Filenames are specified as a string such as "itemltr.png", while icon names (looked up in the current icon theme), are specified with a leading @, such as @"item-ltr". The * character can be used as a wildcard, and if direction/state/size are omitted they default to *. So for example, the following specifies different icons to use for left-to-right and right-to-left languages:

```
stock["my-stock-item"] =
{
  { "itemltr.png", LTR, *, * },
  { "itemrtl.png", RTL, *, * }
}
```

This could be abbreviated as follows:

```
stock["my-stock-item"] =
{
  { "itemltr.png", LTR },
  { "itemrtl.png", RTL }
}
```

You can specify custom icons for specific sizes, as follows:

```
stock["my-stock-item"] =
{
  { "itemmenusize.png", *, *, "gtk-menu" },
  { "itemtoolbarsize.png", *, *, "gtk-large-toolbar" }
  { "itemgeneric.png" } /* implicit *, *, * as a fallback */
}
```

The sizes that come with GTK+ itself are "gtk-menu", "gtk-small-toolbar", "gtk-large-toolbar", "gtk-button", "gtk-dialog". Applications can define other sizes.

It's also possible to use custom icons for a given state, for example:

```
stock["my-stock-item"] =
{
  { "itemprelight.png", *, PRELIGHT },
  { "iteminsensitive.png", *, INSENSITIVE },
  { "itemgeneric.png" } /* implicit *, *, * as a fallback */
}
```

When selecting an icon source to use, GTK+ will consider text direction most important, state second, and size third. It will select the best match based on those criteria. If an attribute matches exactly (e.g. you specified `PRELIGHT` or specified the size), GTK+ won't modify the image; if the attribute matches with a wildcard, GTK+ will scale or modify the image to match the state and size the user requested.

Key bindings

Key bindings allow the user to specify actions to be taken on particular key presses. The form of a binding set declaration is:

```
binding name {
  bind key {
    signalname (param, ...)
    ...
  }
  ...
}
```

`key` is a string consisting of a series of modifiers followed by the name of a key. The modifiers can be:

```
<alt>
<control>
<mod1>
<mod2>
<mod3>
<mod4>
<mod5>
<release>
<shft>
<shift>
```

`<shft>` is an alias for `<shift>` and `<alt>` is an alias for `<mod1>`.

The action that is bound to the key is a sequence of signal names (strings) followed by parameters for each signal. The signals must be action signals. (See g_signal_new()). Each parameter can be a float, integer, string, or unquoted string representing an enumeration value. The types of the parameters specified must match the types of the parameters of the signal.

Binding sets are connected to widgets in the same manner as styles, with one difference: Binding sets override other binding sets first by pattern type, then by priority and then by order of specification. The priorities that can be specified and their default values are the same as for styles.

Details

GtkRcStyle

```
typedef struct {
  gchar *name;
  gchar *bg_pixmap_name[5];
  PangoFontDescription *font_desc;

  GtkRcFlags color_flags[5];
  GdkColor   fg[5];
  GdkColor   bg[5];
  GdkColor   text[5];
  GdkColor   base[5];

  gint xthickness;
  gint ythickness;
} GtkRcStyle;
```

The GtkRcStyle structure is used to represent a set of information about the appearance of a widget. This can later be composited together with other GtkRcStyle structures to form a GtkStyle.

---

enum GtkRcFlags

```
typedef enum
{
  GTK_RC_FG              = 1 << 0,
  GTK_RC_BG              = 1 << 1,
  GTK_RC_TEXT            = 1 << 2,
  GTK_RC_BASE            = 1 << 3
} GtkRcFlags;
```

The GtkRcFlags enumeration is used as a bitmask to specify which fields of a GtkRcStyle have been set for each state.

GTK_RC_FG	If present, the foreground color has been set for this state.
GTK_RC_BG	If present, the background color has been set for this state.
GTK_RC_TEXT	If present, the text color has been set for this state.
GTK_RC_BASE	If present, the base color has been set for this state.

---

enum GtkRcTokenType

```
typedef enum {
  GTK_RC_TOKEN_INVALID = G_TOKEN_LAST,
  GTK_RC_TOKEN_INCLUDE,
  GTK_RC_TOKEN_NORMAL,
  GTK_RC_TOKEN_ACTIVE,
  GTK_RC_TOKEN_PRELIGHT,
  GTK_RC_TOKEN_SELECTED,
  GTK_RC_TOKEN_INSENSITIVE,
  GTK_RC_TOKEN_FG,
  GTK_RC_TOKEN_BG,
  GTK_RC_TOKEN_TEXT,
  GTK_RC_TOKEN_BASE,
  GTK_RC_TOKEN_XTHICKNESS,
  GTK_RC_TOKEN_YTHICKNESS,
  GTK_RC_TOKEN_FONT,
  GTK_RC_TOKEN_FONTSET,
  GTK_RC_TOKEN_FONT_NAME,
  GTK_RC_TOKEN_BG_PIXMAP,
  GTK_RC_TOKEN_PIXMAP_PATH,
  GTK_RC_TOKEN_STYLE,
  GTK_RC_TOKEN_BINDING,
  GTK_RC_TOKEN_BIND,
  GTK_RC_TOKEN_WIDGET,
  GTK_RC_TOKEN_WIDGET_CLASS,
  GTK_RC_TOKEN_CLASS,
  GTK_RC_TOKEN_LOWEST,
  GTK_RC_TOKEN_GTK,
  GTK_RC_TOKEN_APPLICATION,
  GTK_RC_TOKEN_THEME,
  GTK_RC_TOKEN_RC,
  GTK_RC_TOKEN_HIGHEST,
  GTK_RC_TOKEN_ENGINE,
  GTK_RC_TOKEN_MODULE_PATH,
  GTK_RC_TOKEN_IM_MODULE_PATH,
  GTK_RC_TOKEN_IM_MODULE_FILE,
  GTK_RC_TOKEN_STOCK,
```

```
  GTK_RC_TOKEN_LTR,
  GTK_RC_TOKEN_RTL,
  GTK_RC_TOKEN_LAST
} GtkRcTokenType;
```

The GtkRcTokenType enumeration represents the tokens in the RC file. It is exposed so that theme engines can reuse these tokens when parsing the theme-engine specific portions of a RC file.

---

gtk_rc_scanner_new ()

```
GScanner*   gtk_rc_scanner_new              (void);
```

Returns :

---

gtk_rc_get_style ()

```
GtkStyle*   gtk_rc_get_style                (GtkWidget *widget);
```

Finds all matching RC styles for a given widget, composites them together, and then creates a GtkStyle representing the composite appearance. (GTK+ actually keeps a cache of previously created styles, so a new style may not be created.)

widget : a GtkWidget

Returns : the resulting style. No refcount is added to the returned style, so if you want to save this style around, you should add a reference yourself.

---

gtk_rc_get_style_by_paths ()

```
GtkStyle*   gtk_rc_get_style_by_paths       (GtkSettings *settings,
                                             const char *widget_path,
                                             const char *class_path,
                                             GType type);
```

Creates up a GtkStyle from styles defined in a RC file by providing the raw components used in matching. This function may be useful when creating pseudo-widgets that should be themed like widgets but don't actually have corresponding GTK+ widgets. An example of this would be items inside a GNOME canvas widget.

The action of gtk_rc_get_style() is similar to:

```
gtk_widget_path (widget, NULL, &path, NULL);
gtk_widget_class_path (widget, NULL, &class_path, NULL);
gtk_rc_get_style_by_paths (gtk_widget_get_settings (widget), path, class_path,
                           G_OBJECT_TYPE (widget));
```

settings :	a GtkSettings object
widget_path :	the widget path to use when looking up the style, or NULL if no matching against the widget path should be done
class_path :	the class path to use when looking up the style, or NULL if no matching against the class path should be done.
type :	a type that will be used along with parent types of this type when matching against class styles, or G_TYPE_NONE
Returns :	A style created by matching with the supplied paths, or NULL if nothing matching was specified and the default style should be used. The returned value is owned by GTK+ as part of an internal cache, so you must call g_object_ref() on the returned value if you want to keep a reference to it.

---

gtk_rc_add_widget_name_style ()

```
void        gtk_rc_add_widget_name_style    (GtkRcStyle *rc_style,
                                             const gchar *pattern);
```

Warning

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

Adds a GtkRcStyle that will be looked up by a match against the widget's pathname. This is equivalent to a: widget PATTERN style STYLE statement in a RC file.

rc_style :	the GtkRcStyle to use for widgets matching *pattern*
pattern :	the pattern

---

gtk_rc_add_widget_class_style ()

```
void        gtk_rc_add_widget_class_style   (GtkRcStyle *rc_style,
```

```
                                                const gchar *pattern);
```

Warning

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

Adds a GtkRcStyle that will be looked up by a match against the widget's class pathname. This is equivalent to a: `widget_class PATTERN style STYLE` statement in a RC file.

`rc_style` : the GtkRcStyle to use for widgets matching *`pattern`*

`pattern` : the pattern

---

gtk_rc_add_class_style ()

```
void        gtk_rc_add_class_style          (GtkRcStyle *rc_style,
                                             const gchar *pattern);
```

Warning

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

Adds a GtkRcStyle that will be looked up by a matching against the class hierarchy of the widget. This is equivalent to a: `class PATTERN style STYLE` statement in a RC file.

`rc_style` : the GtkRcStyle to use for widgets deriving from *`pattern`*

`pattern` : the pattern

---

gtk_rc_parse ()

```
void        gtk_rc_parse                    (const gchar *filename);
```

Parses a given resource file.

`filename` : the filename of a file to parse. If *`filename`* is not absolute, it is searched in the current directory.

---

gtk_rc_parse_string ()

```
void        gtk_rc_parse_string             (const gchar *rc_string);
```

Parses resource information directly from a string.

rc_string : a string to parse.

---

gtk_rc_reparse_all ()

```
gboolean    gtk_rc_reparse_all              (void);
```

If the modification time on any previously read file for the default GtkSettings has changed, discard all style information and then reread all previously read RC files.

Returns : TRUE if the files were reread.

---

gtk_rc_reparse_all_for_settings ()

```
gboolean    gtk_rc_reparse_all_for_settings (GtkSettings *settings,
                                             gboolean force_load);
```

If the modification time on any previously read file for the given GtkSettings has changed, discard all style information and then reread all previously read RC files.

settings : a GtkSettings
force_load : load whether or not anything changed
Returns : TRUE if the files were reread.

---

gtk_rc_reset_styles ()

```
void        gtk_rc_reset_styles             (GtkSettings *settings);
```

This function recomputes the styles for all widgets that use a particular GtkSettings object. (There is one GtkSettings

object per GdkScreen, see `gtk_settings_get_for_screen()`); It is useful when some global parameter has changed that affects the appearance of all widgets, because when a widget gets a new style, it will both redraw and recompute any cached information about its appearance. As an example, it is used when the default font size set by the operating system changes. Note that this function doesn't affect widgets that have a style set explicitely on them with `gtk_widget_set_style()`.

`settings` : a GtkSettings

Since 2.4

---

gtk_rc_add_default_file ()

```
void        gtk_rc_add_default_file         (const gchar *filename);
```

Adds a file to the list of files to be parsed at the end of `gtk_init()`.

`filename` : the pathname to the file. If *`filename`* is not absolute, it is searched in the current directory.

---

gtk_rc_get_default_files ()

```
gchar**     gtk_rc_get_default_files        (void);
```

Retrieves the current list of RC files that will be parsed at the end of `gtk_init()`.

Returns : A `NULL`-terminated array of filenames. This memory is owned by GTK+ and must not be freed by the application. If you want to store this information, you should make a copy.

---

gtk_rc_set_default_files ()

```
void        gtk_rc_set_default_files        (gchar **filenames);
```

Sets the list of files that GTK+ will read at the end of `gtk_init()`.

`filenames` : A `NULL`-terminated list of filenames.

gtk_rc_parse_color ()

```
guint       gtk_rc_parse_color              (GScanner *scanner,
                                             GdkColor *color);
```

Parses a color in the format expected in a RC file.

scanner : a GtkScanner
color : a pointer to a GtkColor structure in which to store the result
Returns : G_TOKEN_NONE if parsing succeeded, otherwise the token that was expected but not found.

gtk_rc_parse_state ()

```
guint       gtk_rc_parse_state              (GScanner *scanner,
                                             GtkStateType *state);
```

Parses a GtkStateType variable from the format expected in a RC file.

scanner : a GtkScanner (must be initialized for parsing an RC file)
state : A pointer to a GtkStateType variable in which to store the result.
Returns : G_TOKEN_NONE if parsing succeeded, otherwise the token that was expected but not found.

gtk_rc_parse_priority ()

```
guint       gtk_rc_parse_priority           (GScanner *scanner,
                                             GtkPathPriorityType *priority);
```

Parses a GtkPathPriorityType variable from the format expected in a RC file.

scanner : a GtkScanner (must be initialized for parsing an RC file)
priority : A pointer to GtkPathPriorityType variable in which to store the result.
Returns : G_TOKEN_NONE if parsing succeeded, otherwise the token that was expected but not found.

gtk_rc_find_module_in_path ()

```
gchar*      gtk_rc_find_module_in_path      (const gchar *module_file);
```

Searches for a theme engine in the GTK+ search path. This function is not useful for applications and should not be used.

module_file : name of a theme engine
Returns : The filename, if found (must be freed with g_free()), otherwise NULL.

---

gtk_rc_find_pixmap_in_path ()

```
gchar*      gtk_rc_find_pixmap_in_path      (GtkSettings *settings,
                                             GScanner *scanner,
                                             const gchar *pixmap_file);
```

Looks up a file in pixmap path for the specified GtkSettings. If the file is not found, it outputs a warning message using g_warning() and returns NULL.

settings : a GtkSettings
scanner : Scanner used to get line number information for the warning message, or NULL
pixmap_file : name of the pixmap file to locate.
Returns : the filename.

---

gtk_rc_get_module_dir ()

```
gchar*      gtk_rc_get_module_dir           (void);
```

Returns a directory in which GTK+ looks for theme engines. For full information about the search for theme engines, see the docs for GTK_PATH in Running GTK+ Applications(3).

Returns : the directory. (Must be freed with g_free())

---

gtk_rc_get_im_module_path ()

```
gchar*      gtk_rc_get_im_module_path       (void);
```

Obtains the path in which to look for IM modules. See the documentation of the GTK_PATH environment variable for more details about looking up modules. This function is useful solely for utilities supplied with GTK+ and should not be used by applications under normal circumstances.

Returns : a newly-allocated string containing the path in which to look for IM modules.

---

gtk_rc_get_im_module_file ()

```
gchar*      gtk_rc_get_im_module_file       (void);
```

Obtains the path to the IM modules file. See the documentation of the GTK_IM_MODULE_FILE environment variable for more details.

Returns : a newly-allocated string containing the name of the file listing the IM modules available for loading

---

gtk_rc_get_theme_dir ()

```
gchar*      gtk_rc_get_theme_dir            (void);
```

Returns the standard directory in which themes should be installed. (GTK+ does not actually use this directory itself.)

Returns : The directory (must be freed with g_free()).

---

gtk_rc_style_new ()

```
GtkRcStyle* gtk_rc_style_new                (void);
```

Creates a new GtkRcStyle with no fields set and a reference count of 1.

Returns : the newly-created GtkRcStyle

gtk_rc_style_copy ()

```
GtkRcStyle* gtk_rc_style_copy                (GtkRcStyle *orig);
```

Makes a copy of the specified GtkRcStyle. This function will correctly copy an RC style that is a member of a class derived from GtkRcStyle.

orig : the style to copy

Returns : the resulting GtkRcStyle

gtk_rc_style_ref ()

```
void        gtk_rc_style_ref                 (GtkRcStyle *rc_style);
```

Increments the reference count of a GtkRcStyle.

rc_style : a GtkRcStyle

gtk_rc_style_unref ()

```
void        gtk_rc_style_unref               (GtkRcStyle *rc_style);
```

Decrements the reference count of a GtkRcStyle and frees if the result is 0.

rc_style : a GtkRcStyle

<< Themeable Stock Images **Settings >>**

Settings

Settings — Sharing settings between applications

Synopsis

```
#include <gtk/gtk.h>

            GtkSettings;
            GtkSettingsValue;
GtkSettings* gtk_settings_get_default       (void);
GtkSettings* gtk_settings_get_for_screen    (GdkScreen *screen);
void        gtk_settings_install_property   (GParamSpec *pspec);
void        gtk_settings_install_property_parser
                                            (GParamSpec *pspec,
                                             GtkRcPropertyParser parser);
gboolean    gtk_rc_property_parse_color     (const GParamSpec *pspec,
                                             const GString *gstring,
                                             GValue *property_value);
gboolean    gtk_rc_property_parse_enum      (const GParamSpec *pspec,
                                             const GString *gstring,
                                             GValue *property_value);
gboolean    gtk_rc_property_parse_flags     (const GParamSpec *pspec,
                                             const GString *gstring,
                                             GValue *property_value);
gboolean    gtk_rc_property_parse_requisition
                                            (const GParamSpec *pspec,
                                             const GString *gstring,
                                             GValue *property_value);
gboolean    gtk_rc_property_parse_border    (const GParamSpec *pspec,
                                             const GString *gstring,
                                             GValue *property_value);
void        gtk_settings_set_property_value (GtkSettings *settings,
                                             const gchar *name,
                                             const GtkSettingsValue *svalue);
void        gtk_settings_set_string_property
                                            (GtkSettings *settings,
```

```
                                                const gchar *name,
                                                const gchar *v_string,
                                                const gchar *origin);
void            gtk_settings_set_long_property  (GtkSettings *settings,
                                                const gchar *name,
                                                glong v_long,
                                                const gchar *origin);
void            gtk_settings_set_double_property
                                               (GtkSettings *settings,
                                                const gchar *name,
                                                gdouble v_double,
                                                const gchar *origin);
```

Object Hierarchy

```
  GObject
   +----GtkSettings
```

Properties

```
  "gtk-alternative-button-order" gboolean              : Read / Write
  "gtk-button-images"    gboolean              : Read / Write
  "gtk-can-change-accels" gboolean             : Read / Write
  "gtk-color-palette"    gchararray            : Read / Write
  "gtk-cursor-blink"     gboolean              : Read / Write
  "gtk-cursor-blink-time" gint                 : Read / Write
  "gtk-dnd-drag-threshold" gint                  : Read / Write
  "gtk-double-click-distance" gint                  : Read / Write
  "gtk-double-click-time" gint                 : Read / Write
  "gtk-entry-select-on-focus" gboolean              : Read / Write
  "gtk-font-name"        gchararray            : Read / Write
  "gtk-icon-sizes"       gchararray            : Read / Write
  "gtk-icon-theme-name"  gchararray            : Read / Write
  "gtk-key-theme-name"   gchararray            : Read / Write
  "gtk-menu-bar-accel"   gchararray            : Read / Write
  "gtk-menu-bar-popup-delay" gint                  : Read / Write
  "gtk-menu-images"      gboolean              : Read / Write
  "gtk-menu-popdown-delay" gint                  : Read / Write
```

```
  "gtk-menu-popup-delay" gint                  : Read / Write
  "gtk-modules"          gchararray            : Read / Write
  "gtk-split-cursor"     gboolean              : Read / Write
  "gtk-theme-name"       gchararray            : Read / Write
  "gtk-toolbar-icon-size" GtkIconSize           : Read / Write
  "gtk-toolbar-style"    GtkToolbarStyle       : Read / Write
  "gtk-xft-antialias"    gint                  : Read / Write
  "gtk-xft-dpi"          gint                  : Read / Write
  "gtk-xft-hinting"      gint                  : Read / Write
  "gtk-xft-hintstyle"    gchararray            : Read / Write
  "gtk-xft-rgba"         gchararray            : Read / Write
```

Description

Details

GtkSettings

```
typedef struct _GtkSettings GtkSettings;
```

GtkSettingsValue

```
typedef struct {
  /* origin should be something like "filename:linenumber" for rc files,
   * or e.g. "XProperty" for other sources
   */
  gchar *origin;

  /* valid types are LONG, DOUBLE and STRING corresponding to the token parsed,
   * or a GSTRING holding an unparsed statement
   */
  GValue value;
} GtkSettingsValue;
```

---

gtk_settings_get_default ()

```
```

```
GtkSettings* gtk_settings_get_default       (void);
```

Gets the GtkSettings object for the default GDK screen, creating it if necessary. See `gtk_settings_get_for_screen()`.

Returns : a GtkSettings object. If there is no default screen, then returns `NULL`.

---

gtk_settings_get_for_screen ()

```
GtkSettings* gtk_settings_get_for_screen    (GdkScreen *screen);
```

Gets the GtkSettings object for *`screen`*, creating it if necessary.

`screen` : a GdkScreen.
Returns : a GtkSettings object.

Since 2.2

---

gtk_settings_install_property ()

```
void        gtk_settings_install_property   (GParamSpec *pspec);
```

`pspec` :

---

gtk_settings_install_property_parser ()

```
void        gtk_settings_install_property_parser
                                            (GParamSpec *pspec,
                                             GtkRcPropertyParser parser);
```

`pspec` :
`parser` :

---

gtk_rc_property_parse_color ()

```
gboolean    gtk_rc_property_parse_color     (const GParamSpec *pspec,
                                             const GString *gstring,
                                             GValue *property_value);
```

A GtkRcPropertyParser for use with `gtk_settings_install_property_parser()` or `gtk_widget_class_install_style_property_parser()` which parses a color given either by its name or in the form `{ red, green, blue }` where `red`, `green` and `blue` are integers between 0 and 65535 or floating-point numbers between 0 and 1.

pspec :	a GParamSpec
gstring :	the GString to be parsed
property_value :	a GValue which must hold GdkColor values.
Returns :	TRUE if *gstring* could be parsed and *property_value* has been set to the resulting GdkColor.

---

gtk_rc_property_parse_enum ()

```
gboolean    gtk_rc_property_parse_enum      (const GParamSpec *pspec,
                                             const GString *gstring,
                                             GValue *property_value);
```

A GtkRcPropertyParser for use with `gtk_settings_install_property_parser()` or `gtk_widget_class_install_style_property_parser()` which parses a single enumeration value.

The enumeration value can be specified by its name, its nickname or its numeric value. For consistency with flags parsing, the value may be surrounded by parentheses.

pspec :	a GParamSpec
gstring :	the GString to be parsed
property_value :	a GValue which must hold enum values.
Returns :	TRUE if *gstring* could be parsed and *property_value* has been set to the resulting GEnumValue.

---

gtk_rc_property_parse_flags ()

```
gboolean    gtk_rc_property_parse_flags     (const GParamSpec *pspec,
                                             const GString *gstring,
                                             GValue *property_value);
```

A GtkRcPropertyParser for use with `gtk_settings_install_property_parser()` or `gtk_widget_class_install_style_property_parser()` which parses flags.

Flags can be specified by their name, their nickname or numerically. Multiple flags can be specified in the form `"( flag1 | flag2 | ... )"`.

pspec :	a GParamSpec
gstring :	the GString to be parsed
property_value :	a GValue which must hold flags values.
Returns :	TRUE if *gstring* could be parsed and *property_value* has been set to the resulting flags value.

---

gtk_rc_property_parse_requisition ()

```
gboolean    gtk_rc_property_parse_requisition
                                            (const GParamSpec *pspec,
                                             const GString *gstring,
                                             GValue *property_value);
```

A GtkRcPropertyParser for use with `gtk_settings_install_property_parser()` or `gtk_widget_class_install_style_property_parser()` which parses a requisition in the form `"{ width, height }"` for integers `width` and `height`.

pspec :	a GParamSpec
gstring :	the GString to be parsed
property_value :	a GValue which must hold boxed values.
Returns :	TRUE if *gstring* could be parsed and *property_value* has been set to the resulting GtkRequisition.

---

gtk_rc_property_parse_border ()

```
gboolean    gtk_rc_property_parse_border    (const GParamSpec *pspec,
```

```
                                                const GString *gstring,
                                                GValue *property_value);
```

A GtkRcPropertyParser for use with `gtk_settings_install_property_parser()` or `gtk_widget_class_install_style_property_parser()` which parses borders in the form `"{ left, right, top, bottom }"` for integers `left`, `right`, `top` and `bottom`.

pspec :	a GParamSpec
gstring :	the GString to be parsed
property_value :	a GValue which must hold boxed values.
Returns :	TRUE if *gstring* could be parsed and *property_value* has been set to the resulting GtkBorder.

---

gtk_settings_set_property_value ()

```
void        gtk_settings_set_property_value (GtkSettings *settings,
                                             const gchar *name,
                                             const GtkSettingsValue *svalue);
```

settings :	
name :	
svalue :	

---

gtk_settings_set_string_property ()

```
void        gtk_settings_set_string_property
                                            (GtkSettings *settings,
                                             const gchar *name,
                                             const gchar *v_string,
                                             const gchar *origin);
```

settings :	
name :	
v_string :	
origin :	

---

gtk_settings_set_long_property ()

```
void        gtk_settings_set_long_property  (GtkSettings *settings,
                                             const gchar *name,
                                             glong v_long,
                                             const gchar *origin);
```

settings :
name :
v_long :
origin :

---

gtk_settings_set_double_property ()

```
void        gtk_settings_set_double_property
                                            (GtkSettings *settings,
                                             const gchar *name,
                                             gdouble v_double,
                                             const gchar *origin);
```

settings :
name :
v_double :
origin :

Properties

The "gtk-alternative-button-order" property

```
  "gtk-alternative-button-order" gboolean              : Read / Write
```

Whether buttons in dialogs should use the alternative button order.

Default value: FALSE

The "gtk-button-images" property

```
"gtk-button-images"    gboolean              : Read / Write
```

Whether stock icons should be shown in buttons.

Default value: TRUE

The "gtk-can-change-accels" property

```
"gtk-can-change-accels" gboolean              : Read / Write
```

Whether menu accelerators can be changed by pressing a key over the menu item.

Default value: FALSE

The "gtk-color-palette" property

```
"gtk-color-palette"    gchararray            : Read / Write
```

Palette to use in the color selector.

Default value: "black:white:gray50:red:purple:blue:light blue:green:yellow:orange:lavender:brown:goldenrod4:dodger blue:pink:light green:gray10:gray30:gray75:gray90"

The "gtk-cursor-blink" property

```
"gtk-cursor-blink"     gboolean              : Read / Write
```

Whether the cursor should blink.

Default value: TRUE

The "gtk-cursor-blink-time" property

```
"gtk-cursor-blink-time" gint                  : Read / Write
```

Length of the cursor blink cycle, in milleseconds.

Allowed values: >= 100

Default value: 1200

The "gtk-dnd-drag-threshold" property

```
"gtk-dnd-drag-threshold" gint                  : Read / Write
```

Number of pixels the cursor can move before dragging.

Allowed values: >= 1

Default value: 8

The "gtk-double-click-distance" property

```
"gtk-double-click-distance" gint                  : Read / Write
```

Maximum distance allowed between two clicks for them to be considered a double click (in pixels).

Allowed values: >= 0

Default value: 5

The "gtk-double-click-time" property

```
"gtk-double-click-time" gint                  : Read / Write
```

Maximum time allowed between two clicks for them to be considered a double click (in milliseconds).

Allowed values: >= 0

Default value: 250

The "gtk-entry-select-on-focus" property

```
"gtk-entry-select-on-focus" gboolean              : Read / Write
```

Whether to select the contents of an entry when it is focused.

Default value: TRUE

The "gtk-font-name" property

```
"gtk-font-name"       gchararray            : Read / Write
```

Name of default font to use.

Default value: "Sans 10"

The "gtk-icon-sizes" property

```
"gtk-icon-sizes"      gchararray            : Read / Write
```

List of icon sizes (gtk-menu=16,16;gtk-button=20,20...

Default value: NULL

The "gtk-icon-theme-name" property

```
  "gtk-icon-theme-name"  gchararray            : Read / Write
```

Name of icon theme to use.

Default value: "hicolor"

The "gtk-key-theme-name" property

```
  "gtk-key-theme-name"   gchararray            : Read / Write
```

Name of key theme RC file to load.

Default value: NULL

The "gtk-menu-bar-accel" property

```
  "gtk-menu-bar-accel"   gchararray            : Read / Write
```

Keybinding to activate the menu bar.

Default value: "F10"

The "gtk-menu-bar-popup-delay" property

```
  "gtk-menu-bar-popup-delay" gint                    : Read / Write
```

Delay before the submenus of a menu bar appear.

Allowed values: >= 0

Default value: 0

The "gtk-menu-images" property

```
"gtk-menu-images"      gboolean              : Read / Write
```

Whether images should be shown in menus.

Default value: TRUE

The "gtk-menu-popdown-delay" property

```
"gtk-menu-popdown-delay" gint                  : Read / Write
```

The time before hiding a submenu when the pointer is moving towards the submenu.

Allowed values: >= 0

Default value: 1000

The "gtk-menu-popup-delay" property

```
"gtk-menu-popup-delay" gint                    : Read / Write
```

Minimum time the pointer must stay over a menu item before the submenu appear.

Allowed values: >= 0

Default value: 225

The "gtk-modules" property

```
"gtk-modules"          gchararray             : Read / Write
```

List of currently active GTK modules.

Default value: NULL

The "gtk-split-cursor" property

```
"gtk-split-cursor"     gboolean              : Read / Write
```

Whether two cursors should be displayed for mixed left-to-right and right-to-left text.

Default value: TRUE

The "gtk-theme-name" property

```
"gtk-theme-name"       gchararray            : Read / Write
```

Name of theme RC file to load.

Default value: "Default"

The "gtk-toolbar-icon-size" property

```
"gtk-toolbar-icon-size" GtkIconSize           : Read / Write
```

Size of icons in default toolbars.

Default value: GTK_ICON_SIZE_LARGE_TOOLBAR

The "gtk-toolbar-style" property

```
"gtk-toolbar-style"    GtkToolbarStyle        : Read / Write
```

Whether default toolbars have text only, text and icons, icons only, etc.

Default value: GTK_TOOLBAR_BOTH

The "gtk-xft-antialias" property

```
  "gtk-xft-antialias"    gint                : Read / Write
```

Whether to antialias Xft fonts; 0=no, 1=yes, -1=default.

Allowed values: [-1,1]

Default value: -1

The "gtk-xft-dpi" property

```
  "gtk-xft-dpi"          gint                : Read / Write
```

Resolution for Xft, in 1024 * dots/inch. -1 to use default value.

Allowed values: [-1,1048576]

Default value: -1

The "gtk-xft-hinting" property

```
  "gtk-xft-hinting"      gint                : Read / Write
```

Whether to hint Xft fonts; 0=no, 1=yes, -1=default.

Allowed values: [-1,1]

Default value: -1

The "gtk-xft-hintstyle" property

```
  "gtk-xft-hintstyle"    gchararray            : Read / Write
```

What degree of hinting to use; none, slight, medium, or full.

Default value: NULL

The "gtk-xft-rgba" property

```
  "gtk-xft-rgba"         gchararray            : Read / Write
```

Type of subpixel antialiasing; none, rgb, bgr, vrgb, vbgr.

Default value: NULL

<< Resource Files **Bindings >>**

Bindings

Bindings — Key bindings for individual widgets

Synopsis

```
#include <gtk/gtk.h>

            GtkBindingSet;
            GtkBindingEntry;
            GtkBindingSignal;
            GtkBindingArg;
GtkBindingSet* gtk_binding_set_new          (const gchar *set_name);
GtkBindingSet* gtk_binding_set_by_class     (gpointer object_class);
GtkBindingSet* gtk_binding_set_find         (const gchar *set_name);
gboolean    gtk_bindings_activate           (GtkObject *object,
                                             guint keyval,
                                             GdkModifierType modifiers);
gboolean    gtk_bindings_activate_event     (GtkObject *object,
                                             GdkEventKey *event);
gboolean    gtk_binding_set_activate        (GtkBindingSet *binding_set,
                                             guint keyval,
                                             GdkModifierType modifiers,
                                             GtkObject *object);
#define     gtk_binding_entry_add
void        gtk_binding_entry_clear         (GtkBindingSet *binding_set,
                                             guint keyval,
                                             GdkModifierType modifiers);
void        gtk_binding_entry_add_signal    (GtkBindingSet *binding_set,
                                             guint keyval,
                                             GdkModifierType modifiers,
                                             const gchar *signal_name,
                                             guint n_args,
                                             ...);
void        gtk_binding_set_add_path        (GtkBindingSet *binding_set,
```

```
                                                GtkPathType path_type,
                                                const gchar *path_pattern,
                                                GtkPathPriorityType priority);
void        gtk_binding_entry_remove           (GtkBindingSet *binding_set,
                                                guint keyval,
                                                GdkModifierType modifiers);
void        gtk_binding_entry_add_signall      (GtkBindingSet *binding_set,
                                                guint keyval,
                                                GdkModifierType modifiers,
                                                const gchar *signal_name,
                                                GSList *binding_args);
guint       gtk_binding_parse_binding          (GScanner *scanner);
```

Description

Details

GtkBindingSet

```
typedef struct {
  gchar                *set_name;
  gint                  priority;
  GSList               *widget_path_pspecs;
  GSList               *widget_class_pspecs;
  GSList               *class_branch_pspecs;
  GtkBindingEntry      *entries;
  GtkBindingEntry      *current;
  guint                 parsed : 1; /* From RC content */
} GtkBindingSet;
```

GtkBindingEntry

```
typedef struct {
  /* key portion
   */
  guint                 keyval;
  GdkModifierType       modifiers;
```

```
  GtkBindingSet          *binding_set;
  guint                  destroyed : 1;
  guint                  in_emission : 1;
  GtkBindingEntry        *set_next;
  GtkBindingEntry        *hash_next;
  GtkBindingSignal       *signals;
} GtkBindingEntry;
```

GtkBindingSignal

```
typedef struct {
  GtkBindingSignal       *next;
  gchar                  *signal_name;
  guint                   n_args;
  GtkBindingArg          *args;
} GtkBindingSignal;
```

GtkBindingArg

```
typedef struct {
  GType           arg_type;
  union {
    glong         long_data;
    gdouble       double_data;
    gchar        *string_data;
  } d;
} GtkBindingArg;
```

---

gtk_binding_set_new ()

```
GtkBindingSet* gtk_binding_set_new          (const gchar *set_name);
```

set_name:

Returns :

---

gtk_binding_set_by_class ()

```
GtkBindingSet* gtk_binding_set_by_class     (gpointer object_class);
```

object_class :
Returns :

---

gtk_binding_set_find ()

```
GtkBindingSet* gtk_binding_set_find         (const gchar *set_name);
```

set_name :
Returns :

---

gtk_bindings_activate ()

```
gboolean    gtk_bindings_activate           (GtkObject *object,
                                             guint keyval,
                                             GdkModifierType modifiers);
```

object :
keyval :
modifiers :
Returns :

---

gtk_bindings_activate_event ()

```
gboolean    gtk_bindings_activate_event     (GtkObject *object,
```

```
                                                GdkEventKey *event);
```

Looks up key bindings for *object* to find one matching *event*, and if one was found, activate it.

object : a GtkObject (generally must be a widget)
event : a GdkEventKey
Returns : TRUE if a matching key binding was found

---

gtk_binding_set_activate ()

```
gboolean    gtk_binding_set_activate        (GtkBindingSet *binding_set,
                                             guint keyval,
                                             GdkModifierType modifiers,
                                             GtkObject *object);
```

binding_set :
keyval :
modifiers :
object :
Returns :

---

gtk_binding_entry_add

```
#define  gtk_binding_entry_add          gtk_binding_entry_clear
```

---

gtk_binding_entry_clear ()

```
void        gtk_binding_entry_clear         (GtkBindingSet *binding_set,
                                             guint keyval,
                                             GdkModifierType modifiers);
```

binding_set:
keyval:
modifiers:

---

gtk_binding_entry_add_signal ()

```
void        gtk_binding_entry_add_signal    (GtkBindingSet *binding_set,
                                             guint keyval,
                                             GdkModifierType modifiers,
                                             const gchar *signal_name,
                                             guint n_args,
                                             ...);
```

binding_set:
keyval:
modifiers:
signal_name:
n_args:
...:

---

gtk_binding_set_add_path ()

```
void        gtk_binding_set_add_path        (GtkBindingSet *binding_set,
                                             GtkPathType path_type,
                                             const gchar *path_pattern,
                                             GtkPathPriorityType priority);
```

binding_set:
path_type:
path_pattern:
priority:

---

gtk_binding_entry_remove ()

```
void        gtk_binding_entry_remove        (GtkBindingSet *binding_set,
                                             guint keyval,
                                             GdkModifierType modifiers);
```

binding_set :
keyval :
modifiers :

---

gtk_binding_entry_add_signall ()

```
void        gtk_binding_entry_add_signall   (GtkBindingSet *binding_set,
                                             guint keyval,
                                             GdkModifierType modifiers,
                                             const gchar *signal_name,
                                             GSList *binding_args);
```

binding_set :
keyval :
modifiers :
signal_name :
binding_args :

---

gtk_binding_parse_binding ()

```
guint       gtk_binding_parse_binding       (GScanner *scanner);
```

scanner :
Returns :

<< Settings **Standard Enumerations >>**

Standard Enumerations

Standard Enumerations — Public enumerated types used throughout GTK+

Synopsis

```
#include <gtk/gtk.h>

enum        GtkAccelFlags;
enum        GtkAnchorType;
enum        GtkArrowType;
enum        GtkAttachOptions;
enum        GtkButtonBoxStyle;
enum        GtkCornerType;
enum        GtkCurveType;
enum        GtkDeleteType;
enum        GtkDirectionType;
enum        GtkExpanderStyle;
enum        GtkIMPreeditStyle;
enum        GtkIMStatusStyle;
enum        GtkJustification;
enum        GtkMatchType;
enum        GtkMetricType;
enum        GtkMovementStep;
enum        GtkOrientation;
enum        GtkPackType;
enum        GtkPathPriorityType;
enum        GtkPathType;
enum        GtkPolicyType;
enum        GtkPositionType;
enum        GtkPreviewType;
enum        GtkReliefStyle;
enum        GtkResizeMode;
enum        GtkScrollStep;
enum        GtkScrollType;
enum        GtkSelectionMode;
enum        GtkShadowType;
enum        GtkSideType;
enum        GtkStateType;
enum        GtkSubmenuDirection;
```

```
enum        GtkSubmenuPlacement;
enum        GtkToolbarStyle;
enum        GtkUpdateType;
enum        GtkVisibility;
enum        GtkWindowPosition;
enum        GtkWindowType;
enum        GtkSortType;
```

Description

Details

enum GtkAccelFlags

```
typedef enum
{
  GTK_ACCEL_VISIBLE        = 1 << 0,    /* display in GtkAccelLabel? */
  GTK_ACCEL_LOCKED         = 1 << 1,    /* is it removable? */
  GTK_ACCEL_MASK           = 0x07
} GtkAccelFlags;
```

---

enum GtkAnchorType

```
typedef enum
{
  GTK_ANCHOR_CENTER,
  GTK_ANCHOR_NORTH,
  GTK_ANCHOR_NORTH_WEST,
  GTK_ANCHOR_NORTH_EAST,
  GTK_ANCHOR_SOUTH,
  GTK_ANCHOR_SOUTH_WEST,
  GTK_ANCHOR_SOUTH_EAST,
  GTK_ANCHOR_WEST,
  GTK_ANCHOR_EAST,
  GTK_ANCHOR_N          = GTK_ANCHOR_NORTH,
  GTK_ANCHOR_NW         = GTK_ANCHOR_NORTH_WEST,
  GTK_ANCHOR_NE         = GTK_ANCHOR_NORTH_EAST,
  GTK_ANCHOR_S          = GTK_ANCHOR_SOUTH,
  GTK_ANCHOR_SW         = GTK_ANCHOR_SOUTH_WEST,
  GTK_ANCHOR_SE         = GTK_ANCHOR_SOUTH_EAST,
  GTK_ANCHOR_W          = GTK_ANCHOR_WEST,
  GTK_ANCHOR_E          = GTK_ANCHOR_EAST
} GtkAnchorType;
```

---

enum GtkArrowType

```
typedef enum
{
  GTK_ARROW_UP,
  GTK_ARROW_DOWN,
  GTK_ARROW_LEFT,
  GTK_ARROW_RIGHT
} GtkArrowType;
```

Used to indicate the direction in which a GtkArrow should point.

GTK_ARROW_UP	Represents an upward pointing arrow.
GTK_ARROW_DOWN	Represents a downward pointing arrow.
GTK_ARROW_LEFT	Represents a left pointing arrow.
GTK_ARROW_RIGHT	Represents a right pointing arrow.

---

enum GtkAttachOptions

```
typedef enum
{
  GTK_EXPAND = 1 << 0,
  GTK_SHRINK = 1 << 1,
  GTK_FILL   = 1 << 2
} GtkAttachOptions;
```

Denotes the expansion properties that a widget will have when it (or its parent) is resized.

GTK_EXPAND	the widget should expand to take up any extra space in its container that has been allocated.
GTK_SHRINK	the widget should shrink as and when possible.
GTK_FILL	the widget should fill the space allocated to it.

---

enum GtkButtonBoxStyle

```
typedef enum
{
  GTK_BUTTONBOX_DEFAULT_STYLE,
  GTK_BUTTONBOX_SPREAD,
  GTK_BUTTONBOX_EDGE,
  GTK_BUTTONBOX_START,
```

```
  GTK_BUTTONBOX_END
} GtkButtonBoxStyle;
```

Used to dictate the style that a GtkButtonBox uses to layout the buttons it contains. (See also: GtkVButtonBox and GtkHButtonBox).

GTK_BUTTONBOX_DEFAULT_STYLE	Default packing.
GTK_BUTTONBOX_SPREAD	Buttons are evenly spread across the ButtonBox.
GTK_BUTTONBOX_EDGE	Buttons are placed at the edges of the ButtonBox.
GTK_BUTTONBOX_START	Buttons are grouped towards the start of box, (on the left for a HBox, or the top for a VBox).
GTK_BUTTONBOX_END	Buttons are grouped towards the end of a box, (on the right for a HBox, or the bottom for a VBox).

---

enum GtkCornerType

```
typedef enum
{
  GTK_CORNER_TOP_LEFT,
  GTK_CORNER_BOTTOM_LEFT,
  GTK_CORNER_TOP_RIGHT,
  GTK_CORNER_BOTTOM_RIGHT
} GtkCornerType;
```

Specifies which corner a child widget should be placed in when packed into a GtkScrolledWindow. This is effectively the opposite of where the scroll bars are placed.

GTK_CORNER_TOP_LEFT	Place the scrollbars on the right and bottom of the widget (default behaviour).
GTK_CORNER_BOTTOM_LEFT	Place the scrollbars on the top and right of the widget.
GTK_CORNER_TOP_RIGHT	Place the scrollbars on the left and bottom of the widget.
GTK_CORNER_BOTTOM_RIGHT	Place the scrollbars on the top and left of the widget.

---

enum GtkCurveType

```
typedef enum
{
  GTK_CURVE_TYPE_LINEAR,       /* linear interpolation */
  GTK_CURVE_TYPE_SPLINE,       /* spline interpolation */
  GTK_CURVE_TYPE_FREE          /* free form curve */
} GtkCurveType;
```

---

enum GtkDeleteType

```
typedef enum {
  GTK_DELETE_CHARS,
  GTK_DELETE_WORD_ENDS,           /* delete only the portion of the word to the
                                   * left/right of cursor if we're in the middle
                                   * of a word */
  GTK_DELETE_WORDS,
  GTK_DELETE_DISPLAY_LINES,
  GTK_DELETE_DISPLAY_LINE_ENDS,
  GTK_DELETE_PARAGRAPH_ENDS,      /* like C-k in Emacs (or its reverse) */
  GTK_DELETE_PARAGRAPHS,          /* C-k in pico, kill whole line */
  GTK_DELETE_WHITESPACE           /* M-\ in Emacs */
} GtkDeleteType;
```

---

enum GtkDirectionType

```
typedef enum
{
  GTK_DIR_TAB_FORWARD,
  GTK_DIR_TAB_BACKWARD,
  GTK_DIR_UP,
  GTK_DIR_DOWN,
  GTK_DIR_LEFT,
  GTK_DIR_RIGHT
} GtkDirectionType;
```

---

enum GtkExpanderStyle

```
typedef enum
{
  GTK_EXPANDER_COLLAPSED,
  GTK_EXPANDER_SEMI_COLLAPSED,
  GTK_EXPANDER_SEMI_EXPANDED,
  GTK_EXPANDER_EXPANDED
} GtkExpanderStyle;
```

Used to specify the style of the expanders drawn by a GtkTreeView.

GTK_EXPANDER_COLLAPSED	The style used for a collapsed subtree.
GTK_EXPANDER_SEMI_COLLAPSED	Intermediate style used during animation.

GTK_EXPANDER_SEMI_EXPANDED	Intermediate style used during animation.
GTK_EXPANDER_EXPANDED	The style used for an expanded subtree.

---

enum GtkIMPreeditStyle

```
typedef enum
{
  GTK_IM_PREEDIT_NOTHING,
  GTK_IM_PREEDIT_CALLBACK,
  GTK_IM_PREEDIT_NONE
} GtkIMPreeditStyle;
```

---

enum GtkIMStatusStyle

```
typedef enum
{
  GTK_IM_STATUS_NOTHING,
  GTK_IM_STATUS_CALLBACK,
  GTK_IM_STATUS_NONE
} GtkIMStatusStyle;
```

---

enum GtkJustification

```
typedef enum
{
  GTK_JUSTIFY_LEFT,
  GTK_JUSTIFY_RIGHT,
  GTK_JUSTIFY_CENTER,
  GTK_JUSTIFY_FILL
} GtkJustification;
```

Used for justifying the text inside a GtkLabel widget. (See also GtkAlignment).

GTK_JUSTIFY_LEFT	The text is placed at the left edge of the label.
GTK_JUSTIFY_RIGHT	The text is placed at the right edge of the label.
GTK_JUSTIFY_CENTER	The text is placed in the center of the label.
GTK_JUSTIFY_FILL	The text is placed is distributed across the label.

---

enum GtkMatchType

```
typedef enum
{
  GTK_MATCH_ALL,        /* "*A?A*" */
  GTK_MATCH_ALL_TAIL,   /* "*A?AA" */
  GTK_MATCH_HEAD,       /* "AAAA*" */
  GTK_MATCH_TAIL,       /* "*AAAA" */
  GTK_MATCH_EXACT,      /* "AAAAA" */
  GTK_MATCH_LAST
} GtkMatchType;
```

Warning

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

---

enum GtkMetricType

```
typedef enum
{
  GTK_PIXELS,
  GTK_INCHES,
  GTK_CENTIMETERS
} GtkMetricType;
```

Used to indicate which metric is used by a GtkRuler.

`GTK_PIXELS`	Pixels.
`GTK_INCHES`	Inches.
`GTK_CENTIMETERS`	Centimeters.

---

enum GtkMovementStep

```
typedef enum
{
  GTK_MOVEMENT_LOGICAL_POSITIONS, /* move by forw/back graphemes */
  GTK_MOVEMENT_VISUAL_POSITIONS,  /* move by left/right graphemes */
  GTK_MOVEMENT_WORDS,             /* move by forward/back words */
  GTK_MOVEMENT_DISPLAY_LINES,     /* move up/down lines (wrapped lines) */
  GTK_MOVEMENT_DISPLAY_LINE_ENDS, /* move up/down lines (wrapped lines) */
  GTK_MOVEMENT_PARAGRAPHS,        /* move up/down paragraphs (newline-ended lines) */
  GTK_MOVEMENT_PARAGRAPH_ENDS,    /* move to either end of a paragraph */
  GTK_MOVEMENT_PAGES,             /* move by pages */
```

```
  GTK_MOVEMENT_BUFFER_ENDS,       /* move to ends of the buffer */
  GTK_MOVEMENT_HORIZONTAL_PAGES   /* move horizontally by pages */
} GtkMovementStep;
```

---

enum GtkOrientation

```
typedef enum
{
  GTK_ORIENTATION_HORIZONTAL,
  GTK_ORIENTATION_VERTICAL
} GtkOrientation;
```

Represents the orientation of widgets which can be switched between horizontal and vertical orientation on the fly, like GtkToolbar.

GTK_ORIENTATION_HORIZONTAL	The widget is in horizontal orientation.
GTK_ORIENTATION_VERTICAL	The widget is in vertical orientation.

---

enum GtkPackType

```
typedef enum
{
  GTK_PACK_START,
  GTK_PACK_END
} GtkPackType;
```

Represents the packing location GtkBox children. (See: GtkVBox, GtkHBox, and GtkButtonBox).

GTK_PACK_START	The child is packed into the start of the box
GTK_PACK_END	The child is packed into the end of the box

---

enum GtkPathPriorityType

```
typedef enum
{
  GTK_PATH_PRIO_LOWEST      = 0,
  GTK_PATH_PRIO_GTK         = 4,
  GTK_PATH_PRIO_APPLICATION = 8,
  GTK_PATH_PRIO_THEME       = 10,
  GTK_PATH_PRIO_RC          = 12,
```

```
  GTK_PATH_PRIO_HIGHEST     = 15
} GtkPathPriorityType;
```

---

enum GtkPathType

```
typedef enum
{
  GTK_PATH_WIDGET,
  GTK_PATH_WIDGET_CLASS,
  GTK_PATH_CLASS
} GtkPathType;
```

---

enum GtkPolicyType

```
typedef enum
{
  GTK_POLICY_ALWAYS,
  GTK_POLICY_AUTOMATIC,
  GTK_POLICY_NEVER
} GtkPolicyType;
```

Determines when a scroll bar will be visible.

GTK_POLICY_ALWAYS	The scrollbar is always visible.
GTK_POLICY_AUTOMATIC	The scrollbar will appear and disappear as necessary. For example, when all of a GtkCList can not be seen.
GTK_POLICY_NEVER	The scrollbar will never appear.

---

enum GtkPositionType

```
typedef enum
{
  GTK_POS_LEFT,
  GTK_POS_RIGHT,
  GTK_POS_TOP,
  GTK_POS_BOTTOM
} GtkPositionType;
```

Describes which edge of a widget a certain feature is positioned at, e.g. the tabs of a GtkNotebook, the handle of a

GtkHandleBox or the label of a GtkScale.

GTK_POS_LEFT	The feature is at the left edge.
GTK_POS_RIGHT	The feature is at the right edge.
GTK_POS_TOP	The feature is at the top edge.
GTK_POS_BOTTOM	The feature is at the bottom edge.

---

enum GtkPreviewType

```
typedef enum
{
  GTK_PREVIEW_COLOR,
  GTK_PREVIEW_GRAYSCALE
} GtkPreviewType;
```

Warning

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

An enumeration which describes whether a preview contains grayscale or red-green-blue data.

GTK_PREVIEW_COLOR	the preview contains red-green-blue data.
GTK_PREVIEW_GRAYSCALE	The preview contains grayscale data.

---

enum GtkReliefStyle

```
typedef enum
{
  GTK_RELIEF_NORMAL,
  GTK_RELIEF_HALF,
  GTK_RELIEF_NONE
} GtkReliefStyle;
```

Indicated the relief to be drawn around a GtkButton.

GTK_RELIEF_NORMAL	Draw a normal relief.
GTK_RELIEF_HALF	A half relief.
GTK_RELIEF_NONE	No relief.

---

enum GtkResizeMode

```
typedef enum
{
  GTK_RESIZE_PARENT,           /* Pass resize request to the parent */
  GTK_RESIZE_QUEUE,            /* Queue resizes on this widget */
  GTK_RESIZE_IMMEDIATE         /* Perform the resizes now */
} GtkResizeMode;
```

GTK_RESIZE_PARENT

GTK_RESIZE_QUEUE

GTK_RESIZE_IMMEDIATE Deprecated.

---

enum GtkScrollStep

```
typedef enum
{
  GTK_SCROLL_STEPS,
  GTK_SCROLL_PAGES,
  GTK_SCROLL_ENDS,
  GTK_SCROLL_HORIZONTAL_STEPS,
  GTK_SCROLL_HORIZONTAL_PAGES,
  GTK_SCROLL_HORIZONTAL_ENDS
} GtkScrollStep;
```

---

enum GtkScrollType

```
typedef enum
{
  GTK_SCROLL_NONE,
  GTK_SCROLL_JUMP,
  GTK_SCROLL_STEP_BACKWARD,
  GTK_SCROLL_STEP_FORWARD,
  GTK_SCROLL_PAGE_BACKWARD,
  GTK_SCROLL_PAGE_FORWARD,
  GTK_SCROLL_STEP_UP,
  GTK_SCROLL_STEP_DOWN,
  GTK_SCROLL_PAGE_UP,
  GTK_SCROLL_PAGE_DOWN,
  GTK_SCROLL_STEP_LEFT,
  GTK_SCROLL_STEP_RIGHT,
  GTK_SCROLL_PAGE_LEFT,
  GTK_SCROLL_PAGE_RIGHT,
  GTK_SCROLL_START,
  GTK_SCROLL_END
} GtkScrollType;
```

enum GtkSelectionMode

```
typedef enum
{
  GTK_SELECTION_NONE,                                /* Nothing can be selected */
  GTK_SELECTION_SINGLE,
  GTK_SELECTION_BROWSE,
  GTK_SELECTION_MULTIPLE,
  GTK_SELECTION_EXTENDED = GTK_SELECTION_MULTIPLE /* Deprecated */
} GtkSelectionMode;
```

Used to control what selections users are allowed to make.

GTK_SELECTION_NONE	No selection is possible.
GTK_SELECTION_SINGLE	Zero or one element may be selected.
GTK_SELECTION_BROWSE	Exactly one element is selected. In some circumstances, such as initially or during a search operation, it's possible for no element to be selected with GTK_SELECTION_BROWSE. What is really enforced is that the user can't deselect a currently selected element except by selecting another element.
GTK_SELECTION_MULTIPLE	Any number of elements may be selected. Clicks toggle the state of an item. Any number of elements may be selected. Click-drag selects a range of elements; the Ctrl key may be used to enlarge the selection, and Shift key to select between the focus and the child pointed to.
GTK_SELECTION_EXTENDED	Deprecated, behaves identical to GTK_SELECTION_MULTIPLE.

enum GtkShadowType

```
typedef enum
{
  GTK_SHADOW_NONE,
  GTK_SHADOW_IN,
  GTK_SHADOW_OUT,
  GTK_SHADOW_ETCHED_IN,
  GTK_SHADOW_ETCHED_OUT
} GtkShadowType;
```

Used to change the appearance of an outline typically provided by a GtkFrame.

GTK_SHADOW_NONE	No outline.
GTK_SHADOW_IN	The outline is bevelled inwards.
GTK_SHADOW_OUT	The outline is bevelled outwards like a button.

GTK_SHADOW_ETCHED_IN	The outline itself is an inward bevel, but the frame does
GTK_SHADOW_ETCHED_OUT	

---

enum GtkSideType

```
typedef enum
{
  GTK_SIDE_TOP,
  GTK_SIDE_BOTTOM,
  GTK_SIDE_LEFT,
  GTK_SIDE_RIGHT
} GtkSideType;
```

Warning

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

---

enum GtkStateType

```
typedef enum
{
  GTK_STATE_NORMAL,
  GTK_STATE_ACTIVE,
  GTK_STATE_PRELIGHT,
  GTK_STATE_SELECTED,
  GTK_STATE_INSENSITIVE
} GtkStateType;
```

This type indicates the current state of a widget; the state determines how the widget is drawn. The GtkStateType enumeration is also used to identify different colors in a GtkStyle for drawing, so states can be used for subparts of a widget as well as entire widgets.

GTK_STATE_NORMAL	State during normal operation.
GTK_STATE_ACTIVE	State of a currently active widget, such as a depressed button.
GTK_STATE_PRELIGHT	State indicating that the mouse pointer is over the widget and the widget will respond to mouse clicks.
GTK_STATE_SELECTED	State of a selected item, such the selected row in a list.
GTK_STATE_INSENSITIVE	State indicating that the widget is unresponsive to user actions.

---

enum GtkSubmenuDirection

```
typedef enum
{
  GTK_DIRECTION_LEFT,
  GTK_DIRECTION_RIGHT
} GtkSubmenuDirection;
```

Warning

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

Indicates the direction a sub-menu will appear.

`GTK_DIRECTION_LEFT` A sub-menu will appear
`GTK_DIRECTION_RIGHT`

---

enum GtkSubmenuPlacement

```
typedef enum
{
  GTK_TOP_BOTTOM,
  GTK_LEFT_RIGHT
} GtkSubmenuPlacement;
```

Warning

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

---

enum GtkToolbarStyle

```
typedef enum
{
  GTK_TOOLBAR_ICONS,
  GTK_TOOLBAR_TEXT,
  GTK_TOOLBAR_BOTH,
  GTK_TOOLBAR_BOTH_HORIZ
} GtkToolbarStyle;
```

Used to customize the appearance of a GtkToolbar. Note that setting the toolbar style overrides the user's preferences for the default toolbar style.

`GTK_TOOLBAR_ICONS` Buttons display only icons in the toolbar.

GTK_TOOLBAR_TEXT	Buttons display only text labels in the toolbar.
GTK_TOOLBAR_BOTH	Buttons display text and icons in the toolbar.
GTK_TOOLBAR_BOTH_HORIZ	Buttons display icons and text alongside each other, rather than vertically stacked

---

enum GtkUpdateType

```
typedef enum
{
  GTK_UPDATE_CONTINUOUS,
  GTK_UPDATE_DISCONTINUOUS,
  GTK_UPDATE_DELAYED
} GtkUpdateType;
```

---

enum GtkVisibility

```
typedef enum
{
  GTK_VISIBILITY_NONE,
  GTK_VISIBILITY_PARTIAL,
  GTK_VISIBILITY_FULL
} GtkVisibility;
```

Used by GtkCList and GtkCTree to indicate whether a row is visible.

GTK_VISIBILITY_NONE	The row is not visible.
GTK_VISIBILITY_PARTIAL	The row is partially visible.
GTK_VISIBILITY_FULL	The row is fully visible.

---

enum GtkWindowPosition

```
typedef enum
{
  GTK_WIN_POS_NONE,
  GTK_WIN_POS_CENTER,
  GTK_WIN_POS_MOUSE,
  GTK_WIN_POS_CENTER_ALWAYS,
  GTK_WIN_POS_CENTER_ON_PARENT
} GtkWindowPosition;
```

Window placement can be influenced using this enumeration.

GTK_WIN_POS_NONE	No influence is made on placement.
GTK_WIN_POS_CENTER	Windows should be placed in the center of the screen.
GTK_WIN_POS_MOUSE	Windows should be placed at the current mouse position.
GTK_WIN_POS_CENTER_ALWAYS	Keep window centered as it changes size, etc.
GTK_WIN_POS_CENTER_ON_PARENT	Center the window on its transient parent (see gtk_window_set_transient_for()).

---

enum GtkWindowType

```
typedef enum
{
  GTK_WINDOW_TOPLEVEL,
  GTK_WINDOW_POPUP
} GtkWindowType;
```

A GtkWindow can be one of these types. Most things you'd consider a "window" should have type GTK_WINDOW_TOPLEVEL; windows with this type are managed by the window manager and have a frame by default (call gtk_window_set_decorated() to toggle the frame). Windows with type GTK_WINDOW_POPUP are ignored by the window manager; window manager keybindings won't work on them, the window manager won't decorate the window with a frame, many GTK+ features that rely on the window manager will not work (e.g. resize grips and maximization/minimization). GTK_WINDOW_POPUP is used to implement widgets such as GtkMenu or tooltips that you normally don't think of as windows per se. Nearly all windows should be GTK_WINDOW_TOPLEVEL. In particular, do not use GTK_WINDOW_POPUP just to turn off the window borders; use gtk_window_set_decorated() for that.

GTK_WINDOW_TOPLEVEL	A regular window, such as a dialog.
GTK_WINDOW_POPUP	A special window such as a tooltip.

---

enum GtkSortType

```
typedef enum
{
  GTK_SORT_ASCENDING,
  GTK_SORT_DESCENDING
} GtkSortType;
```

Determines the direction of a sort.

GTK_SORT_ASCENDING	Sorting is in ascending order.
GTK_SORT_DESCENDING	Sorting is in descending order.

Graphics Contexts

Graphics Contexts — A shared pool of GdkGC objects

Synopsis

```
#include <gtk/gtk.h>

GdkGC*      gtk_gc_get                      (gint depth,
                                             GdkColormap *colormap,
                                             GdkGCValues *values,
                                             GdkGCValuesMask values_mask);
void        gtk_gc_release                  (GdkGC *gc);
```

Description

These functions provide access to a shared pool of GdkGC objects. When a new GdkGC is needed, `gtk_gc_get()` is called with the required depth, colormap and GdkGCValues. If a GdkGC with the required properties already exists then that is returned. If not, a new GdkGC is created. When the GdkGC is no longer needed, `gtk_gc_release()` should be called.

Details

gtk_gc_get ()

```
GdkGC*      gtk_gc_get                      (gint depth,
                                             GdkColormap *colormap,
                                             GdkGCValues *values,
                                             GdkGCValuesMask values_mask);
```

Gets a GdkGC with the given depth, colormap and GdkGCValues. If a GdkGC with the given properties already

exists then it is returned, otherwise a new GdkGC is created. The returned GdkGC should be released with `gtk_gc_release()` when it is no longer needed.

depth :	the depth of the GdkGC to create.
colormap :	the GdkColormap (FIXME: I don't know why this is needed).
values :	a GdkGCValues struct containing settings for the GdkGC.
values_mask :	a set of flags indicating which of the fields in *values* has been set.
Returns :	a GdkGC.

---

gtk_gc_release ()

```
void        gtk_gc_release                  (GdkGC *gc);
```

Releases a GdkGC allocated using `gtk_gc_get()`.

gc : a GdkGC.

<< Standard Enumerations **Styles >>**

Styles

Styles — Functions for drawing widget parts

Synopsis

```
#include <gtk/gtk.h>

#define     GTK_STYLE_ATTACHED              (style)
            GtkStyle;
GtkStyle*   gtk_style_new                   (void);
GtkStyle*   gtk_style_copy                  (GtkStyle *style);
GtkStyle*   gtk_style_attach                (GtkStyle *style,
                                             GdkWindow *window);
void        gtk_style_detach                (GtkStyle *style);
GtkStyle*   gtk_style_ref                   (GtkStyle *style);
void        gtk_style_unref                 (GtkStyle *style);
void        gtk_style_set_background        (GtkStyle *style,
                                             GdkWindow *window,
                                             GtkStateType state_type);
void        gtk_style_apply_default_background
                                            (GtkStyle *style,
                                             GdkWindow *window,
                                             gboolean set_bg,
                                             GtkStateType state_type,
                                             GdkRectangle *area,
                                             gint x,
                                             gint y,
                                             gint width,
                                             gint height);
#define     gtk_style_apply_default_pixmap  (s,gw,st,a,x,y,w,h)
GtkIconSet* gtk_style_lookup_icon_set       (GtkStyle *style,
                                             const gchar *stock_id);
GdkPixbuf*  gtk_style_render_icon           (GtkStyle *style,
                                             const GtkIconSource *source,
                                             GtkTextDirection direction,
```

```
                                        GtkStateType state,
                                        GtkIconSize size,
                                        GtkWidget *widget,
                                        const gchar *detail);
GdkFont*    gtk_style_get_font          (GtkStyle *style);
void        gtk_style_set_font          (GtkStyle *style,
                                        GdkFont *font);
void        gtk_draw_hline              (GtkStyle *style,
                                        GdkWindow *window,
                                        GtkStateType state_type,
                                        gint x1,
                                        gint x2,
                                        gint y);
void        gtk_draw_vline              (GtkStyle *style,
                                        GdkWindow *window,
                                        GtkStateType state_type,
                                        gint y1_,
                                        gint y2_,
                                        gint x);
void        gtk_draw_shadow             (GtkStyle *style,
                                        GdkWindow *window,
                                        GtkStateType state_type,
                                        GtkShadowType shadow_type,
                                        gint x,
                                        gint y,
                                        gint width,
                                        gint height);
void        gtk_draw_polygon            (GtkStyle *style,
                                        GdkWindow *window,
                                        GtkStateType state_type,
                                        GtkShadowType shadow_type,
                                        GdkPoint *points,
                                        gint npoints,
                                        gboolean fill);
void        gtk_draw_arrow              (GtkStyle *style,
                                        GdkWindow *window,
                                        GtkStateType state_type,
                                        GtkShadowType shadow_type,
                                        GtkArrowType arrow_type,
                                        gboolean fill,
                                        gint x,
                                        gint y,
                                        gint width,
```

```
                                        gint height);
void        gtk_draw_diamond            (GtkStyle *style,
                                         GdkWindow *window,
                                         GtkStateType state_type,
                                         GtkShadowType shadow_type,
                                         gint x,
                                         gint y,
                                         gint width,
                                         gint height);
void        gtk_draw_string             (GtkStyle *style,
                                         GdkWindow *window,
                                         GtkStateType state_type,
                                         gint x,
                                         gint y,
                                         const gchar *string);
void        gtk_draw_box                (GtkStyle *style,
                                         GdkWindow *window,
                                         GtkStateType state_type,
                                         GtkShadowType shadow_type,
                                         gint x,
                                         gint y,
                                         gint width,
                                         gint height);
void        gtk_draw_box_gap            (GtkStyle *style,
                                         GdkWindow *window,
                                         GtkStateType state_type,
                                         GtkShadowType shadow_type,
                                         gint x,
                                         gint y,
                                         gint width,
                                         gint height,
                                         GtkPositionType gap_side,
                                         gint gap_x,
                                         gint gap_width);
void        gtk_draw_check              (GtkStyle *style,
                                         GdkWindow *window,
                                         GtkStateType state_type,
                                         GtkShadowType shadow_type,
                                         gint x,
                                         gint y,
                                         gint width,
                                         gint height);
void        gtk_draw_extension          (GtkStyle *style,
```

```
                                    GdkWindow *window,
                                    GtkStateType state_type,
                                    GtkShadowType shadow_type,
                                    gint x,
                                    gint y,
                                    gint width,
                                    gint height,
                                    GtkPositionType gap_side);
void        gtk_draw_flat_box       (GtkStyle *style,
                                    GdkWindow *window,
                                    GtkStateType state_type,
                                    GtkShadowType shadow_type,
                                    gint x,
                                    gint y,
                                    gint width,
                                    gint height);
void        gtk_draw_focus          (GtkStyle *style,
                                    GdkWindow *window,
                                    gint x,
                                    gint y,
                                    gint width,
                                    gint height);
void        gtk_draw_handle         (GtkStyle *style,
                                    GdkWindow *window,
                                    GtkStateType state_type,
                                    GtkShadowType shadow_type,
                                    gint x,
                                    gint y,
                                    gint width,
                                    gint height,
                                    GtkOrientation orientation);
void        gtk_draw_option         (GtkStyle *style,
                                    GdkWindow *window,
                                    GtkStateType state_type,
                                    GtkShadowType shadow_type,
                                    gint x,
                                    gint y,
                                    gint width,
                                    gint height);
void        gtk_draw_shadow_gap     (GtkStyle *style,
                                    GdkWindow *window,
                                    GtkStateType state_type,
                                    GtkShadowType shadow_type,
```

```
                                    gint x,
                                    gint y,
                                    gint width,
                                    gint height,
                                    GtkPositionType gap_side,
                                    gint gap_x,
                                    gint gap_width);
void        gtk_draw_slider         (GtkStyle *style,
                                    GdkWindow *window,
                                    GtkStateType state_type,
                                    GtkShadowType shadow_type,
                                    gint x,
                                    gint y,
                                    gint width,
                                    gint height,
                                    GtkOrientation orientation);
void        gtk_draw_tab            (GtkStyle *style,
                                    GdkWindow *window,
                                    GtkStateType state_type,
                                    GtkShadowType shadow_type,
                                    gint x,
                                    gint y,
                                    gint width,
                                    gint height);
void        gtk_draw_expander       (GtkStyle *style,
                                    GdkWindow *window,
                                    GtkStateType state_type,
                                    gint x,
                                    gint y,
                                    GtkExpanderStyle expander_style);
void        gtk_draw_layout         (GtkStyle *style,
                                    GdkWindow *window,
                                    GtkStateType state_type,
                                    gboolean use_text,
                                    gint x,
                                    gint y,
                                    PangoLayout *layout);
void        gtk_draw_resize_grip    (GtkStyle *style,
                                    GdkWindow *window,
                                    GtkStateType state_type,
                                    GdkWindowEdge edge,
                                    gint x,
                                    gint y,
```

```
                                    gint width,
                                    gint height);
void        gtk_paint_arrow         (GtkStyle *style,
                                    GdkWindow *window,
                                    GtkStateType state_type,
                                    GtkShadowType shadow_type,
                                    GdkRectangle *area,
                                    GtkWidget *widget,
                                    const gchar *detail,
                                    GtkArrowType arrow_type,
                                    gboolean fill,
                                    gint x,
                                    gint y,
                                    gint width,
                                    gint height);
void        gtk_paint_box           (GtkStyle *style,
                                    GdkWindow *window,
                                    GtkStateType state_type,
                                    GtkShadowType shadow_type,
                                    GdkRectangle *area,
                                    GtkWidget *widget,
                                    const gchar *detail,
                                    gint x,
                                    gint y,
                                    gint width,
                                    gint height);
void        gtk_paint_box_gap       (GtkStyle *style,
                                    GdkWindow *window,
                                    GtkStateType state_type,
                                    GtkShadowType shadow_type,
                                    GdkRectangle *area,
                                    GtkWidget *widget,
                                    gchar *detail,
                                    gint x,
                                    gint y,
                                    gint width,
                                    gint height,
                                    GtkPositionType gap_side,
                                    gint gap_x,
                                    gint gap_width);
void        gtk_paint_check         (GtkStyle *style,
                                    GdkWindow *window,
                                    GtkStateType state_type,
```

```
                                    GtkShadowType shadow_type,
                                    GdkRectangle *area,
                                    GtkWidget *widget,
                                    const gchar *detail,
                                    gint x,
                                    gint y,
                                    gint width,
                                    gint height);
void        gtk_paint_diamond       (GtkStyle *style,
                                    GdkWindow *window,
                                    GtkStateType state_type,
                                    GtkShadowType shadow_type,
                                    GdkRectangle *area,
                                    GtkWidget *widget,
                                    const gchar *detail,
                                    gint x,
                                    gint y,
                                    gint width,
                                    gint height);
void        gtk_paint_extension     (GtkStyle *style,
                                    GdkWindow *window,
                                    GtkStateType state_type,
                                    GtkShadowType shadow_type,
                                    GdkRectangle *area,
                                    GtkWidget *widget,
                                    gchar *detail,
                                    gint x,
                                    gint y,
                                    gint width,
                                    gint height,
                                    GtkPositionType gap_side);
void        gtk_paint_flat_box      (GtkStyle *style,
                                    GdkWindow *window,
                                    GtkStateType state_type,
                                    GtkShadowType shadow_type,
                                    GdkRectangle *area,
                                    GtkWidget *widget,
                                    const gchar *detail,
                                    gint x,
                                    gint y,
                                    gint width,
                                    gint height);
void        gtk_paint_focus         (GtkStyle *style,
```

```
                                    GdkWindow *window,
                                    GtkStateType state_type,
                                    GdkRectangle *area,
                                    GtkWidget *widget,
                                    const gchar *detail,
                                    gint x,
                                    gint y,
                                    gint width,
                                    gint height);
void        gtk_paint_handle        (GtkStyle *style,
                                    GdkWindow *window,
                                    GtkStateType state_type,
                                    GtkShadowType shadow_type,
                                    GdkRectangle *area,
                                    GtkWidget *widget,
                                    const gchar *detail,
                                    gint x,
                                    gint y,
                                    gint width,
                                    gint height,
                                    GtkOrientation orientation);
void        gtk_paint_hline         (GtkStyle *style,
                                    GdkWindow *window,
                                    GtkStateType state_type,
                                    GdkRectangle *area,
                                    GtkWidget *widget,
                                    const gchar *detail,
                                    gint x1,
                                    gint x2,
                                    gint y);
void        gtk_paint_option        (GtkStyle *style,
                                    GdkWindow *window,
                                    GtkStateType state_type,
                                    GtkShadowType shadow_type,
                                    GdkRectangle *area,
                                    GtkWidget *widget,
                                    const gchar *detail,
                                    gint x,
                                    gint y,
                                    gint width,
                                    gint height);
void        gtk_paint_polygon       (GtkStyle *style,
                                    GdkWindow *window,
```

```
                                    GtkStateType state_type,
                                    GtkShadowType shadow_type,
                                    GdkRectangle *area,
                                    GtkWidget *widget,
                                    const gchar *detail,
                                    GdkPoint *points,
                                    gint npoints,
                                    gboolean fill);
void        gtk_paint_shadow        (GtkStyle *style,
                                    GdkWindow *window,
                                    GtkStateType state_type,
                                    GtkShadowType shadow_type,
                                    GdkRectangle *area,
                                    GtkWidget *widget,
                                    const gchar *detail,
                                    gint x,
                                    gint y,
                                    gint width,
                                    gint height);
void        gtk_paint_shadow_gap    (GtkStyle *style,
                                    GdkWindow *window,
                                    GtkStateType state_type,
                                    GtkShadowType shadow_type,
                                    GdkRectangle *area,
                                    GtkWidget *widget,
                                    gchar *detail,
                                    gint x,
                                    gint y,
                                    gint width,
                                    gint height,
                                    GtkPositionType gap_side,
                                    gint gap_x,
                                    gint gap_width);
void        gtk_paint_slider        (GtkStyle *style,
                                    GdkWindow *window,
                                    GtkStateType state_type,
                                    GtkShadowType shadow_type,
                                    GdkRectangle *area,
                                    GtkWidget *widget,
                                    const gchar *detail,
                                    gint x,
                                    gint y,
                                    gint width,
```

```
                                        gint height,
                                        GtkOrientation orientation);
void        gtk_paint_string           (GtkStyle *style,
                                        GdkWindow *window,
                                        GtkStateType state_type,
                                        GdkRectangle *area,
                                        GtkWidget *widget,
                                        const gchar *detail,
                                        gint x,
                                        gint y,
                                        const gchar *string);
void        gtk_paint_tab              (GtkStyle *style,
                                        GdkWindow *window,
                                        GtkStateType state_type,
                                        GtkShadowType shadow_type,
                                        GdkRectangle *area,
                                        GtkWidget *widget,
                                        const gchar *detail,
                                        gint x,
                                        gint y,
                                        gint width,
                                        gint height);
void        gtk_paint_vline            (GtkStyle *style,
                                        GdkWindow *window,
                                        GtkStateType state_type,
                                        GdkRectangle *area,
                                        GtkWidget *widget,
                                        const gchar *detail,
                                        gint y1_,
                                        gint y2_,
                                        gint x);
void        gtk_paint_expander         (GtkStyle *style,
                                        GdkWindow *window,
                                        GtkStateType state_type,
                                        GdkRectangle *area,
                                        GtkWidget *widget,
                                        const gchar *detail,
                                        gint x,
                                        gint y,
                                        GtkExpanderStyle expander_style);
void        gtk_paint_layout           (GtkStyle *style,
                                        GdkWindow *window,
                                        GtkStateType state_type,
```

```
                                                gboolean use_text,
                                                GdkRectangle *area,
                                                GtkWidget *widget,
                                                const gchar *detail,
                                                gint x,
                                                gint y,
                                                PangoLayout *layout);
void            gtk_paint_resize_grip           (GtkStyle *style,
                                                GdkWindow *window,
                                                GtkStateType state_type,
                                                GdkRectangle *area,
                                                GtkWidget *widget,
                                                const gchar *detail,
                                                GdkWindowEdge edge,
                                                gint x,
                                                gint y,
                                                gint width,
                                                gint height);
void            gtk_draw_insertion_cursor       (GtkWidget *widget,
                                                GdkDrawable *drawable,
                                                GdkRectangle *area,
                                                GdkRectangle *location,
                                                gboolean is_primary,
                                                GtkTextDirection direction,
                                                gboolean draw_arrow);

                GtkBorder;
GtkBorder*      gtk_border_copy                 (const GtkBorder *border_);
void            gtk_border_free                 (GtkBorder *border_);

                GtkRcProperty;
gboolean        (*GtkRcPropertyParser)          (const GParamSpec *pspec,
                                                const GString *rc_string,
                                                GValue *property_value);
```

Object Hierarchy

```
  GObject
   +----GtkStyle
```

Signal Prototypes

```
"realize"   void        user_function      (GtkStyle *style,
                                            gpointer user_data);
"unrealize" void        user_function      (GtkStyle *style,
                                            gpointer user_data);
```

Description

Details

GTK_STYLE_ATTACHED()

```
#define GTK_STYLE_ATTACHED(style)       (GTK_STYLE (style)->attach_count > 0)
```

Returns whether the style is attached to a window.

style : a GtkStyle.

GtkStyle

```
typedef struct {
  GdkColor fg[5];
  GdkColor bg[5];
  GdkColor light[5];
  GdkColor dark[5];
  GdkColor mid[5];
  GdkColor text[5];
  GdkColor base[5];
  GdkColor text_aa[5];          /* Halfway between text/base */

  GdkColor black;
  GdkColor white;
  PangoFontDescription *font_desc;

  gint xthickness;
  gint ythickness;
```

```
  GdkGC *fg_gc[5];
  GdkGC *bg_gc[5];
  GdkGC *light_gc[5];
  GdkGC *dark_gc[5];
  GdkGC *mid_gc[5];
  GdkGC *text_gc[5];
  GdkGC *base_gc[5];
  GdkGC *text_aa_gc[5];
  GdkGC *black_gc;
  GdkGC *white_gc;

  GdkPixmap *bg_pixmap[5];
} GtkStyle;
```

---

gtk_style_new ()

```
GtkStyle*   gtk_style_new                   (void);
```

Creates a new GtkStyle.

Returns : a new GtkStyle.

---

gtk_style_copy ()

```
GtkStyle*   gtk_style_copy                  (GtkStyle *style);
```

style :

Returns :

---

gtk_style_attach ()

```
GtkStyle*   gtk_style_attach                (GtkStyle *style,
                                             GdkWindow *window);
```

Attaches a style to a window; this process allocates the colors and creates the GC's for the style - it specializes it to a

particular visual and colormap. The process may involve the creation of a new style if the style has already been attached to a window with a different style and colormap.

style : a GtkStyle.

window : a GdkWindow.

Returns : Either *style*, or a newly-created GtkStyle. If the style is newly created, the style parameter will be dereferenced, and the new style will have a reference count belonging to the caller.

---

gtk_style_detach ()

```
void        gtk_style_detach                (GtkStyle *style);
```

style :

---

gtk_style_ref ()

```
GtkStyle*   gtk_style_ref                   (GtkStyle *style);
```

Warning

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

Deprecated equivalent of `g_object_ref()`.

style : a GtkStyle.

Returns : *style*.

---

gtk_style_unref ()

```
void        gtk_style_unref                 (GtkStyle *style);
```

Warning

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

Deprecated equivalent of `g_object_unref()`.

style : a GtkStyle.

---

gtk_style_set_background ()

```
void        gtk_style_set_background        (GtkStyle *style,
                                             GdkWindow *window,
                                             GtkStateType state_type);
```

Sets the background of *window* to the background color or pixmap specified by *style* for the given state.

style : a GtkStyle
window : a GdkWindow
state_type : a state

---

gtk_style_apply_default_background ()

```
void        gtk_style_apply_default_background
                                            (GtkStyle *style,
                                             GdkWindow *window,
                                             gboolean set_bg,
                                             GtkStateType state_type,
                                             GdkRectangle *area,
                                             gint x,
                                             gint y,
                                             gint width,
                                             gint height);
```

style :
window :
set_bg :
state_type :
area :
x :
y :

width :
height :

---

gtk_style_apply_default_pixmap()

```
#define gtk_style_apply_default_pixmap(s,gw,st,a,x,y,w,h)
gtk_style_apply_default_background (s,gw,1,st,a,x,y,w,h)
```

Warning

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

Deprecated alias for `gtk_style_apply_default_background()`.

s :
gw :
st :
a :
x :
y :
w :
h :

---

gtk_style_lookup_icon_set ()

```
GtkIconSet* gtk_style_lookup_icon_set       (GtkStyle *style,
                                             const gchar *stock_id);
```

style :
stock_id :
Returns :

---

gtk_style_render_icon ()

```
GdkPixbuf*  gtk_style_render_icon           (GtkStyle *style,
```

```
                                                const GtkIconSource *source,
                                                GtkTextDirection direction,
                                                GtkStateType state,
                                                GtkIconSize size,
                                                GtkWidget *widget,
                                                const gchar *detail);
```

Renders the icon specified by *source* at the given *size* according to the given parameters and returns the result in a pixbuf.

style :	a GtkStyle
source :	the GtkIconSource specifying the icon to render
direction :	a text direction
state :	a state
size :	the size to render the icon at. A size of (GtkIconSize)-1 means render at the size of the source and don't scale.
widget :	the widget
detail :	a style detail
Returns :	a newly-created GdkPixbuf containing the rendered icon

---

gtk_style_get_font ()

```
GdkFont*    gtk_style_get_font              (GtkStyle *style);
```

Warning

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

Gets the GdkFont to use for the given style. This is meant only as a replacement for direct access to *style->font* and should not be used in new code. New code should use *style->font_desc* instead.

style :	a GtkStyle
Returns :	the GdkFont for the style. This font is owned by the style; if you want to keep around a copy, you must call `gdk_font_ref()`.

---

gtk_style_set_font ()

```
void        gtk_style_set_font              (GtkStyle *style,
                                             GdkFont *font);
```

Warning

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

Sets the GdkFont to use for a given style. This is meant only as a replacement for direct access to style->font and should not be used in new code. New code should use style->font_desc instead.

style : a GtkStyle.

font : a GdkFont, or `NULL` to use the GdkFont corresponding to style->font_desc.

---

gtk_draw_hline ()

```
void        gtk_draw_hline                  (GtkStyle *style,
                                             GdkWindow *window,
                                             GtkStateType state_type,
                                             gint x1,
                                             gint x2,
                                             gint y);
```

Warning

`gtk_draw_hline` is deprecated and should not be used in newly-written code. Use `gtk_paint_hline()` instead.

Draws a horizontal line from (*x1*, *y*) to (*x2*, *y*) in *window* using the given style and state.

style : a GtkStyle

window : a GdkWindow

state_type : a state

x1 : the starting x coordinate

x2 : the ending x coordinate

y : the y coordinate

---

gtk_draw_vline ()

```
void        gtk_draw_vline                  (GtkStyle *style,
                                             GdkWindow *window,
                                             GtkStateType state_type,
                                             gint y1_,
                                             gint y2_,
                                             gint x);
```

Warning

`gtk_draw_vline` is deprecated and should not be used in newly-written code. Use `gtk_paint_vline()` instead.

Draws a vertical line from (*x*, *y1_*) to (*x*, *y2_*) in *window* using the given style and state.

style :	a GtkStyle
window :	a GdkWindow
state_type :	a state
*y1_* :	the starting y coordinate
*y2_* :	the ending y coordinate
x :	the x coordinate

---

gtk_draw_shadow ()

```
void        gtk_draw_shadow                 (GtkStyle *style,
                                             GdkWindow *window,
                                             GtkStateType state_type,
                                             GtkShadowType shadow_type,
                                             gint x,
                                             gint y,
                                             gint width,
                                             gint height);
```

Warning

`gtk_draw_shadow` is deprecated and should not be used in newly-written code. Use `gtk_paint_shadow()` instead.

Draws a shadow around the given rectangle in *window* using the given style and state and shadow type.

style : a GtkStyle
window : a GdkWindow
state_type : a state
shadow_type : type of shadow to draw
x : x origin of the rectangle
y : y origin of the rectangle
width : width of the rectangle
height : width of the rectangle

---

gtk_draw_polygon ()

```
void        gtk_draw_polygon                (GtkStyle *style,
                                             GdkWindow *window,
                                             GtkStateType state_type,
                                             GtkShadowType shadow_type,
                                             GdkPoint *points,
                                             gint npoints,
                                             gboolean fill);
```

Warning

`gtk_draw_polygon` is deprecated and should not be used in newly-written code. Use `gtk_paint_polygon()` instead.

Draws a polygon on *window* with the given parameters.

style : a GtkStyle
window : a GdkWindow
state_type : a state
shadow_type : type of shadow to draw
points : an array of GdkPoints
npoints : length of *points*
fill : TRUE if the polygon should be filled

---

gtk_draw_arrow ()

```
void        gtk_draw_arrow                  (GtkStyle *style,
                                             GdkWindow *window,
                                             GtkStateType state_type,
                                             GtkShadowType shadow_type,
                                             GtkArrowType arrow_type,
                                             gboolean fill,
                                             gint x,
                                             gint y,
                                             gint width,
                                             gint height);
```

Warning

`gtk_draw_arrow` is deprecated and should not be used in newly-written code. Use `gtk_paint_arrow()` instead.

Draws an arrow in the given rectangle on *`window`* using the given parameters. *`arrow_type`* determines the direction of the arrow.

`style` :	a GtkStyle
`window` :	a GdkWindow
`state_type` :	a state
`shadow_type` :	the type of shadow to draw
`arrow_type` :	the type of arrow to draw
`fill` :	TRUE if the arrow tip should be filled
`x` :	x origin of the rectangle to draw the arrow in
`y` :	y origin of the rectangle to draw the arrow in
`width` :	width of the rectangle to draw the arrow in
`height` :	height of the rectangle to draw the arrow in

---

gtk_draw_diamond ()

```
void        gtk_draw_diamond                (GtkStyle *style,
                                             GdkWindow *window,
                                             GtkStateType state_type,
                                             GtkShadowType shadow_type,
                                             gint x,
                                             gint y,
                                             gint width,
```

```
                                                gint height);
```

Warning

`gtk_draw_diamond` is deprecated and should not be used in newly-written code. Use `gtk_paint_diamond()` instead.

Draws a diamond in the given rectangle on *`window`* using the given parameters.

`style` :	a GtkStyle
`window` :	a GdkWindow
`state_type` :	a state
`shadow_type` :	the type of shadow to draw
`x` :	x origin of the rectangle to draw the diamond in
`y` :	y origin of the rectangle to draw the diamond in
`width` :	width of the rectangle to draw the diamond in
`height` :	height of the rectangle to draw the diamond in

---

gtk_draw_string ()

```
void        gtk_draw_string                 (GtkStyle *style,
                                             GdkWindow *window,
                                             GtkStateType state_type,
                                             gint x,
                                             gint y,
                                             const gchar *string);
```

Warning

`gtk_draw_string` is deprecated and should not be used in newly-written code. Use `gtk_paint_layout()` instead.

Draws a text string on *`window`* with the given parameters.

`style` :	a GtkStyle
`window` :	a GdkWindow
`state_type` :	a state
`x` :	x origin

y : y origin
string : the string to draw

---

gtk_draw_box ()

```
void        gtk_draw_box                    (GtkStyle *style,
                                             GdkWindow *window,
                                             GtkStateType state_type,
                                             GtkShadowType shadow_type,
                                             gint x,
                                             gint y,
                                             gint width,
                                             gint height);
```

Warning

`gtk_draw_box` is deprecated and should not be used in newly-written code. Use `gtk_paint_box()` instead.

Draws a box on *window* with the given parameters.

style : a GtkStyle
window : a GdkWindow
state_type : a state
shadow_type : the type of shadow to draw
x : x origin of the box
y : y origin of the box
width : the width of the box
height : the height of the box

---

gtk_draw_box_gap ()

```
void        gtk_draw_box_gap                (GtkStyle *style,
                                             GdkWindow *window,
                                             GtkStateType state_type,
                                             GtkShadowType shadow_type,
                                             gint x,
```

```
                                                gint y,
                                                gint width,
                                                gint height,
                                                GtkPositionType gap_side,
                                                gint gap_x,
                                                gint gap_width);
```

Warning

`gtk_draw_box_gap` is deprecated and should not be used in newly-written code. Use `gtk_paint_box_gap()` instead.

Draws a box in *`window`* using the given style and state and shadow type, leaving a gap in one side.

`style` :	a GtkStyle
`window` :	a GdkWindow
`state_type` :	a state
`shadow_type` :	type of shadow to draw
`x` :	x origin of the rectangle
`y` :	y origin of the rectangle
`width` :	width of the rectangle
`height` :	width of the rectangle
`gap_side` :	side in which to leave the gap
`gap_x` :	starting position of the gap
`gap_width` :	width of the gap

---

gtk_draw_check ()

```
void        gtk_draw_check                  (GtkStyle *style,
                                             GdkWindow *window,
                                             GtkStateType state_type,
                                             GtkShadowType shadow_type,
                                             gint x,
                                             gint y,
                                             gint width,
                                             gint height);
```

Warning

`gtk_draw_check` is deprecated and should not be used in newly-written code. Use `gtk_paint_check()` instead.

Draws a check button indicator in the given rectangle on *`window`* with the given parameters.

`style` :	a GtkStyle
`window` :	a GdkWindow
`state_type` :	a state
`shadow_type` :	the type of shadow to draw
`x` :	x origin of the rectangle to draw the check in
`y` :	y origin of the rectangle to draw the check in
`width` :	the width of the rectangle to draw the check in
`height` :	the height of the rectangle to draw the check in

---

gtk_draw_extension ()

```
void        gtk_draw_extension              (GtkStyle *style,
                                             GdkWindow *window,
                                             GtkStateType state_type,
                                             GtkShadowType shadow_type,
                                             gint x,
                                             gint y,
                                             gint width,
                                             gint height,
                                             GtkPositionType gap_side);
```

Warning

`gtk_draw_extension` is deprecated and should not be used in newly-written code. Use `gtk_paint_extension()` instead.

Draws an extension, i.e. a notebook tab.

`style` :	a GtkStyle
`window` :	a GdkWindow
`state_type` :	a state
`shadow_type` :	type of shadow to draw
`x` :	x origin of the extension

y :	y origin of the extension
width :	width of the extension
height :	width of the extension
gap_side :	the side on to which the extension is attached

---

gtk_draw_flat_box ()

```
void        gtk_draw_flat_box               (GtkStyle *style,
                                             GdkWindow *window,
                                             GtkStateType state_type,
                                             GtkShadowType shadow_type,
                                             gint x,
                                             gint y,
                                             gint width,
                                             gint height);
```

Warning

`gtk_draw_flat_box` is deprecated and should not be used in newly-written code. Use `gtk_paint_flat_box()` instead.

Draws a flat box on *window* with the given parameters.

style :	a GtkStyle
window :	a GdkWindow
state_type :	a state
shadow_type :	the type of shadow to draw
x :	x origin of the box
y :	y origin of the box
width :	the width of the box
height :	the height of the box

---

gtk_draw_focus ()

```
void        gtk_draw_focus                  (GtkStyle *style,
                                             GdkWindow *window,
                                             gint x,
```

```
                                                gint y,
                                                gint width,
                                                gint height);
```

Warning

`gtk_draw_focus` is deprecated and should not be used in newly-written code. Use `gtk_paint_focus()` instead.

Draws a focus indicator around the given rectangle on *`window`* using the given style.

`style`: a GtkStyle
`window`: a GdkWindow
`x`: the x origin of the rectangle around which to draw a focus indicator
`y`: the y origin of the rectangle around which to draw a focus indicator
`width`: the width of the rectangle around which to draw a focus indicator
`height`: the height of the rectangle around which to draw a focus indicator

---

gtk_draw_handle ()

```
void        gtk_draw_handle                 (GtkStyle *style,
                                             GdkWindow *window,
                                             GtkStateType state_type,
                                             GtkShadowType shadow_type,
                                             gint x,
                                             gint y,
                                             gint width,
                                             gint height,
                                             GtkOrientation orientation);
```

Warning

`gtk_draw_handle` is deprecated and should not be used in newly-written code. Use `gtk_paint_handle()` instead.

Draws a handle as used in GtkHandleBox and GtkPaned.

`style`: a GtkStyle

window : a GdkWindow
state_type : a state
shadow_type : type of shadow to draw
x : x origin of the handle
y : y origin of the handle
width : with of the handle
height : height of the handle
orientation : the orientation of the handle

---

gtk_draw_option ()

```
void        gtk_draw_option                     (GtkStyle *style,
                                                 GdkWindow *window,
                                                 GtkStateType state_type,
                                                 GtkShadowType shadow_type,
                                                 gint x,
                                                 gint y,
                                                 gint width,
                                                 gint height);
```

Warning

`gtk_draw_option` is deprecated and should not be used in newly-written code. Use `gtk_paint_option()` instead.

Draws a radio button indicator in the given rectangle on *window* with the given parameters.

style : a GtkStyle
window : a GdkWindow
state_type : a state
shadow_type : the type of shadow to draw
x : x origin of the rectangle to draw the option in
y : y origin of the rectangle to draw the option in
width : the width of the rectangle to draw the option in
height : the height of the rectangle to draw the option in

---

gtk_draw_shadow_gap ()

```
void        gtk_draw_shadow_gap             (GtkStyle *style,
                                             GdkWindow *window,
                                             GtkStateType state_type,
                                             GtkShadowType shadow_type,
                                             gint x,
                                             gint y,
                                             gint width,
                                             gint height,
                                             GtkPositionType gap_side,
                                             gint gap_x,
                                             gint gap_width);
```

Warning

`gtk_draw_shadow_gap` is deprecated and should not be used in newly-written code. Use `gtk_paint_shadow_gap()` instead.

Draws a shadow around the given rectangle in *`window`* using the given style and state and shadow type, leaving a gap in one side.

`style` :	a GtkStyle
`window` :	a GdkWindow
`state_type` :	a state
`shadow_type` :	type of shadow to draw
`x` :	x origin of the rectangle
`y` :	y origin of the rectangle
`width` :	width of the rectangle
`height` :	width of the rectangle
`gap_side` :	side in which to leave the gap
`gap_x` :	starting position of the gap
`gap_width` :	width of the gap

---

gtk_draw_slider ()

```
void        gtk_draw_slider                 (GtkStyle *style,
                                             GdkWindow *window,
                                             GtkStateType state_type,
                                             GtkShadowType shadow_type,
```

```
                                                gint x,
                                                gint y,
                                                gint width,
                                                gint height,
                                                GtkOrientation orientation);
```

Warning

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

style:
window:
state_type:
shadow_type:
x:
y:
width:
height:
orientation:

---

gtk_draw_tab ()

```
void        gtk_draw_tab                        (GtkStyle *style,
                                                 GdkWindow *window,
                                                 GtkStateType state_type,
                                                 GtkShadowType shadow_type,
                                                 gint x,
                                                 gint y,
                                                 gint width,
                                                 gint height);
```

Warning

`gtk_draw_tab` is deprecated and should not be used in newly-written code. Use `gtk_paint_tab()` instead.

Draws an option menu tab (i.e. the up and down pointing arrows) in the given rectangle on *window* using the given parameters.

style : a GtkStyle
window : a GdkWindow
state_type : a state
shadow_type : the type of shadow to draw
x : x origin of the rectangle to draw the tab in
y : y origin of the rectangle to draw the tab in
width : the width of the rectangle to draw the tab in
height : the height of the rectangle to draw the tab in

---

gtk_draw_expander ()

```
void        gtk_draw_expander               (GtkStyle *style,
                                             GdkWindow *window,
                                             GtkStateType state_type,
                                             gint x,
                                             gint y,
                                             GtkExpanderStyle expander_style);
```

Warning

`gtk_draw_expander` is deprecated and should not be used in newly-written code. Use `gtk_paint_expander()` instead.

Draws an expander as used in GtkTreeView.

style : a GtkStyle
window : a GdkWindow
state_type : a state
x : the x position to draw the expander at
y : the y position to draw the expander at
expander_style : the style to draw the expander in

---

gtk_draw_layout ()

```
void        gtk_draw_layout                 (GtkStyle *style,
                                             GdkWindow *window,
                                             GtkStateType state_type,
```

```
                                                gboolean use_text,
                                                gint x,
                                                gint y,
                                                PangoLayout *layout);
```

Warning

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

style :
window :
state_type :
use_text :
x :
y :
layout :

---

gtk_draw_resize_grip ()

```
void        gtk_draw_resize_grip            (GtkStyle *style,
                                             GdkWindow *window,
                                             GtkStateType state_type,
                                             GdkWindowEdge edge,
                                             gint x,
                                             gint y,
                                             gint width,
                                             gint height);
```

Warning

`gtk_draw_resize_grip` is deprecated and should not be used in newly-written code. Use `gtk_paint_resize_grip()` instead.

Draws a resize grip in the given rectangle on *window* using the given parameters.

style : a GtkStyle
window : a GdkWindow
state_type : a state

edge : the edge in which to draw the resize grip
x : the x origin of the rectangle in which to draw the resize grip
y : the y origin of the rectangle in which to draw the resize grip
width : the width of the rectangle in which to draw the resize grip
height : the height of the rectangle in which to draw the resize grip

---

gtk_paint_arrow ()

```
void        gtk_paint_arrow                 (GtkStyle *style,
                                             GdkWindow *window,
                                             GtkStateType state_type,
                                             GtkShadowType shadow_type,
                                             GdkRectangle *area,
                                             GtkWidget *widget,
                                             const gchar *detail,
                                             GtkArrowType arrow_type,
                                             gboolean fill,
                                             gint x,
                                             gint y,
                                             gint width,
                                             gint height);
```

Draws an arrow in the given rectangle on *window* using the given parameters. *arrow_type* determines the direction of the arrow.

style : a GtkStyle
window : a GdkWindow
state_type : a state
shadow_type : the type of shadow to draw
area : clip rectangle
widget : the widget
detail : a style detail
arrow_type : the type of arrow to draw
fill : TRUE if the arrow tip should be filled
x : x origin of the rectangle to draw the arrow in
y : y origin of the rectangle to draw the arrow in
width : width of the rectangle to draw the arrow in
height : height of the rectangle to draw the arrow in

---

gtk_paint_box ()

```
void        gtk_paint_box                   (GtkStyle *style,
                                             GdkWindow *window,
                                             GtkStateType state_type,
                                             GtkShadowType shadow_type,
                                             GdkRectangle *area,
                                             GtkWidget *widget,
                                             const gchar *detail,
                                             gint x,
                                             gint y,
                                             gint width,
                                             gint height);
```

Draws a box on *window* with the given parameters.

style :	a GtkStyle
window :	a GdkWindow
state_type :	a state
shadow_type :	the type of shadow to draw
area :	clip rectangle
widget :	the widget
detail :	a style detail
x :	x origin of the box
y :	y origin of the box
width :	the width of the box
height :	the height of the box

---

gtk_paint_box_gap ()

```
void        gtk_paint_box_gap               (GtkStyle *style,
                                             GdkWindow *window,
                                             GtkStateType state_type,
                                             GtkShadowType shadow_type,
                                             GdkRectangle *area,
                                             GtkWidget *widget,
                                             gchar *detail,
```

```
                                                gint x,
                                                gint y,
                                                gint width,
                                                gint height,
                                                GtkPositionType gap_side,
                                                gint gap_x,
                                                gint gap_width);
```

Draws a box in *window* using the given style and state and shadow type, leaving a gap in one side.

style :	a GtkStyle
window :	a GdkWindow
state_type :	a state
shadow_type :	type of shadow to draw
area :	clip rectangle
widget :	the widget
detail :	a style detail
x :	x origin of the rectangle
y :	y origin of the rectangle
width :	width of the rectangle
height :	width of the rectangle
gap_side :	side in which to leave the gap
gap_x :	starting position of the gap
gap_width :	width of the gap

---

gtk_paint_check ()

```
void        gtk_paint_check                 (GtkStyle *style,
                                             GdkWindow *window,
                                             GtkStateType state_type,
                                             GtkShadowType shadow_type,
                                             GdkRectangle *area,
                                             GtkWidget *widget,
                                             const gchar *detail,
                                             gint x,
                                             gint y,
                                             gint width,
                                             gint height);
```

Draws a check button indicator in the given rectangle on *window* with the given parameters.

style :	a GtkStyle
window :	a GdkWindow
state_type :	a state
shadow_type :	the type of shadow to draw
area :	clip rectangle
widget :	the widget
detail :	a style detail
x :	x origin of the rectangle to draw the check in
y :	y origin of the rectangle to draw the check in
width :	the width of the rectangle to draw the check in
height :	the height of the rectangle to draw the check in

---

gtk_paint_diamond ()

```
void        gtk_paint_diamond               (GtkStyle *style,
                                             GdkWindow *window,
                                             GtkStateType state_type,
                                             GtkShadowType shadow_type,
                                             GdkRectangle *area,
                                             GtkWidget *widget,
                                             const gchar *detail,
                                             gint x,
                                             gint y,
                                             gint width,
                                             gint height);
```

Draws a diamond in the given rectangle on *window* using the given parameters.

style :	a GtkStyle
window :	a GdkWindow
state_type :	a state
shadow_type :	the type of shadow to draw
area :	clip rectangle
widget :	the widget
detail :	a style detail
x :	x origin of the rectangle to draw the diamond in

y :	y origin of the rectangle to draw the diamond in
width :	width of the rectangle to draw the diamond in
height :	height of the rectangle to draw the diamond in

---

gtk_paint_extension ()

```
void        gtk_paint_extension             (GtkStyle *style,
                                             GdkWindow *window,
                                             GtkStateType state_type,
                                             GtkShadowType shadow_type,
                                             GdkRectangle *area,
                                             GtkWidget *widget,
                                             gchar *detail,
                                             gint x,
                                             gint y,
                                             gint width,
                                             gint height,
                                             GtkPositionType gap_side);
```

Draws an extension, i.e. a notebook tab.

style :	a GtkStyle
window :	a GdkWindow
state_type :	a state
shadow_type :	type of shadow to draw
area :	clip rectangle
widget :	the widget
detail :	a style detail
x :	x origin of the extension
y :	y origin of the extension
width :	width of the extension
height :	width of the extension
gap_side :	the side on to which the extension is attached

---

gtk_paint_flat_box ()

```
void        gtk_paint_flat_box              (GtkStyle *style,
```

```
                                    GdkWindow *window,
                                    GtkStateType state_type,
                                    GtkShadowType shadow_type,
                                    GdkRectangle *area,
                                    GtkWidget *widget,
                                    const gchar *detail,
                                    gint x,
                                    gint y,
                                    gint width,
                                    gint height);
```

Draws a flat box on *window* with the given parameters.

style :	a GtkStyle
window :	a GdkWindow
state_type :	a state
shadow_type :	the type of shadow to draw
area :	clip rectangle
widget :	the widget
detail :	a style detail
x :	x origin of the box
y :	y origin of the box
width :	the width of the box
height :	the height of the box

---

gtk_paint_focus ()

```
void        gtk_paint_focus                 (GtkStyle *style,
                                             GdkWindow *window,
                                             GtkStateType state_type,
                                             GdkRectangle *area,
                                             GtkWidget *widget,
                                             const gchar *detail,
                                             gint x,
                                             gint y,
                                             gint width,
                                             gint height);
```

Draws a focus indicator around the given rectangle on *window* using the given style.

style : a GtkStyle
window : a GdkWindow
state_type : a state
area : clip rectangle
widget : the widget
detail : a style detail
x : the x origin of the rectangle around which to draw a focus indicator
y : the y origin of the rectangle around which to draw a focus indicator
width : the width of the rectangle around which to draw a focus indicator
height : the height of the rectangle around which to draw a focus indicator

---

gtk_paint_handle ()

```
void        gtk_paint_handle                    (GtkStyle *style,
                                                 GdkWindow *window,
                                                 GtkStateType state_type,
                                                 GtkShadowType shadow_type,
                                                 GdkRectangle *area,
                                                 GtkWidget *widget,
                                                 const gchar *detail,
                                                 gint x,
                                                 gint y,
                                                 gint width,
                                                 gint height,
                                                 GtkOrientation orientation);
```

Draws a handle as used in GtkHandleBox and GtkPaned.

style : a GtkStyle
window : a GdkWindow
state_type : a state
shadow_type : type of shadow to draw
area : clip rectangle
widget : the widget
detail : a style detail
x : x origin of the handle
y : y origin of the handle

width : with of the handle
height : height of the handle
orientation : the orientation of the handle

---

gtk_paint_hline ()

```
void        gtk_paint_hline                 (GtkStyle *style,
                                             GdkWindow *window,
                                             GtkStateType state_type,
                                             GdkRectangle *area,
                                             GtkWidget *widget,
                                             const gchar *detail,
                                             gint x1,
                                             gint x2,
                                             gint y);
```

Draws a horizontal line from (*x1*, *y*) to (*x2*, *y*) in *window* using the given style and state.

style : a GtkStyle
window : a GdkWindow
state_type : a state
area : rectangle to which the output is clipped
widget : the widget
detail : a style detail
x1 : the starting x coordinate
x2 : the ending x coordinate
y : the y coordinate

---

gtk_paint_option ()

```
void        gtk_paint_option                (GtkStyle *style,
                                             GdkWindow *window,
                                             GtkStateType state_type,
                                             GtkShadowType shadow_type,
                                             GdkRectangle *area,
                                             GtkWidget *widget,
                                             const gchar *detail,
```

```
                                                gint x,
                                                gint y,
                                                gint width,
                                                gint height);
```

Draws a radio button indicator in the given rectangle on *window* with the given parameters.

style :	a GtkStyle
window :	a GdkWindow
state_type :	a state
shadow_type :	the type of shadow to draw
area :	clip rectangle
widget :	the widget
detail :	a style detail
x :	x origin of the rectangle to draw the option in
y :	y origin of the rectangle to draw the option in
width :	the width of the rectangle to draw the option in
height :	the height of the rectangle to draw the option in

---

gtk_paint_polygon ()

```
void        gtk_paint_polygon               (GtkStyle *style,
                                             GdkWindow *window,
                                             GtkStateType state_type,
                                             GtkShadowType shadow_type,
                                             GdkRectangle *area,
                                             GtkWidget *widget,
                                             const gchar *detail,
                                             GdkPoint *points,
                                             gint npoints,
                                             gboolean fill);
```

Draws a polygon on *window* with the given parameters.

style :	a GtkStyle
window :	a GdkWindow
state_type :	a state
shadow_type :	type of shadow to draw

area :	clip rectangle
widget :	the widget
detail :	a style detail
points :	an array of GdkPoints
npoints :	length of *points*
fill :	TRUE if the polygon should be filled

---

gtk_paint_shadow ()

```
void        gtk_paint_shadow                    (GtkStyle *style,
                                                 GdkWindow *window,
                                                 GtkStateType state_type,
                                                 GtkShadowType shadow_type,
                                                 GdkRectangle *area,
                                                 GtkWidget *widget,
                                                 const gchar *detail,
                                                 gint x,
                                                 gint y,
                                                 gint width,
                                                 gint height);
```

Draws a shadow around the given rectangle in *window* using the given style and state and shadow type.

style :	a GtkStyle
window :	a GdkWindow
state_type :	a state
shadow_type :	type of shadow to draw
area :	clip rectangle
widget :	the widget
detail :	a style detail
x :	x origin of the rectangle
y :	y origin of the rectangle
width :	width of the rectangle
height :	width of the rectangle

---

gtk_paint_shadow_gap ()

```
void        gtk_paint_shadow_gap            (GtkStyle *style,
                                             GdkWindow *window,
                                             GtkStateType state_type,
                                             GtkShadowType shadow_type,
                                             GdkRectangle *area,
                                             GtkWidget *widget,
                                             gchar *detail,
                                             gint x,
                                             gint y,
                                             gint width,
                                             gint height,
                                             GtkPositionType gap_side,
                                             gint gap_x,
                                             gint gap_width);
```

Draws a shadow around the given rectangle in *window* using the given style and state and shadow type, leaving a gap in one side.

style :	a GtkStyle
window :	a GdkWindow
state_type :	a state
shadow_type :	type of shadow to draw
area :	clip rectangle
widget :	the widget
detail :	a style detail
x :	x origin of the rectangle
y :	y origin of the rectangle
width :	width of the rectangle
height :	width of the rectangle
gap_side :	side in which to leave the gap
gap_x :	starting position of the gap
gap_width :	width of the gap

---

gtk_paint_slider ()

```
void        gtk_paint_slider                (GtkStyle *style,
                                             GdkWindow *window,
                                             GtkStateType state_type,
                                             GtkShadowType shadow_type,
```

```
                                    GdkRectangle *area,
                                    GtkWidget *widget,
                                    const gchar *detail,
                                    gint x,
                                    gint y,
                                    gint width,
                                    gint height,
                                    GtkOrientation orientation);
```

style:
window:
state_type:
shadow_type:
area:
widget:
detail:
x:
y:
width:
height:
orientation:

---

gtk_paint_string ()

```
void        gtk_paint_string                (GtkStyle *style,
                                             GdkWindow *window,
                                             GtkStateType state_type,
                                             GdkRectangle *area,
                                             GtkWidget *widget,
                                             const gchar *detail,
                                             gint x,
                                             gint y,
                                             const gchar *string);
```

Warning

`gtk_paint_string` is deprecated and should not be used in newly-written code. Use `gtk_paint_layout()` instead.

Draws a text string on *window* with the given parameters.

style :	a GtkStyle
window :	a GdkWindow
state_type :	a state
area :	clip rectangle
widget :	the widget
detail :	a style detail
x :	x origin
y :	y origin
string :	the string to draw

---

gtk_paint_tab ()

```
void        gtk_paint_tab                       (GtkStyle *style,
                                                 GdkWindow *window,
                                                 GtkStateType state_type,
                                                 GtkShadowType shadow_type,
                                                 GdkRectangle *area,
                                                 GtkWidget *widget,
                                                 const gchar *detail,
                                                 gint x,
                                                 gint y,
                                                 gint width,
                                                 gint height);
```

Draws an option menu tab (i.e. the up and down pointing arrows) in the given rectangle on *window* using the given parameters.

style :	a GtkStyle
window :	a GdkWindow
state_type :	a state
shadow_type :	the type of shadow to draw
area :	clip rectangle
widget :	the widget
detail :	a style detail
x :	x origin of the rectangle to draw the tab in
y :	y origin of the rectangle to draw the tab in

width : the width of the rectangle to draw the tab in
height : the height of the rectangle to draw the tab in

---

gtk_paint_vline ()

```
void        gtk_paint_vline                 (GtkStyle *style,
                                             GdkWindow *window,
                                             GtkStateType state_type,
                                             GdkRectangle *area,
                                             GtkWidget *widget,
                                             const gchar *detail,
                                             gint y1_,
                                             gint y2_,
                                             gint x);
```

Draws a vertical line from (*x*, *y1_*) to (*x*, *y2_*) in *window* using the given style and state.

style : a GtkStyle
window : a GdkWindow
state_type : a state
area : rectangle to which the output is clipped
widget : the widget
detail : a style detail
*y1_* : the starting y coordinate
*y2_* : the ending y coordinate
x : the x coordinate

---

gtk_paint_expander ()

```
void        gtk_paint_expander              (GtkStyle *style,
                                             GdkWindow *window,
                                             GtkStateType state_type,
                                             GdkRectangle *area,
                                             GtkWidget *widget,
                                             const gchar *detail,
                                             gint x,
                                             gint y,
```

```
                                                GtkExpanderStyle expander_style);
```

Draws an expander as used in GtkTreeView.

style :	a GtkStyle
window :	a GdkWindow
state_type :	a state
area :	clip rectangle
widget :	the widget
detail :	a style detail
x :	the x position to draw the expander at
y :	the y position to draw the expander at
expander_style :	the style to draw the expander in

---

gtk_paint_layout ()

```
void        gtk_paint_layout                (GtkStyle *style,
                                             GdkWindow *window,
                                             GtkStateType state_type,
                                             gboolean use_text,
                                             GdkRectangle *area,
                                             GtkWidget *widget,
                                             const gchar *detail,
                                             gint x,
                                             gint y,
                                             PangoLayout *layout);
```

style :	
window :	
state_type :	
use_text :	
area :	
widget :	
detail :	
x :	
y :	
layout :	

---

gtk_paint_resize_grip ()

```
void        gtk_paint_resize_grip           (GtkStyle *style,
                                             GdkWindow *window,
                                             GtkStateType state_type,
                                             GdkRectangle *area,
                                             GtkWidget *widget,
                                             const gchar *detail,
                                             GdkWindowEdge edge,
                                             gint x,
                                             gint y,
                                             gint width,
                                             gint height);
```

Draws a resize grip in the given rectangle on *window* using the given parameters.

style :	a GtkStyle
window :	a GdkWindow
state_type :	a state
area :	clip rectangle
widget :	the widget
detail :	a style detail
edge :	the edge in which to draw the resize grip
x :	the x origin of the rectangle in which to draw the resize grip
y :	the y origin of the rectangle in which to draw the resize grip
width :	the width of the rectangle in which to draw the resize grip
height :	the height of the rectangle in which to draw the resize grip

---

gtk_draw_insertion_cursor ()

```
void        gtk_draw_insertion_cursor       (GtkWidget *widget,
                                             GdkDrawable *drawable,
                                             GdkRectangle *area,
                                             GdkRectangle *location,
                                             gboolean is_primary,
                                             GtkTextDirection direction,
                                             gboolean draw_arrow);
```

Draws a text caret on *drawable* at *location*. This is not a style function but merely a convenience function for drawing the standard cursor shape.

widget :	a GtkWidget
drawable :	a GdkDrawable
area :	rectangle to which the output is clipped, or NULL if the output should not be clipped
location :	location where to draw the cursor (*location->width* is ignored)
is_primary :	if the cursor should be the primary cursor color.
direction :	whether the cursor is left-to-right or right-to-left. Should never be GTK_TEXT_DIR_NONE
draw_arrow :	TRUE to draw a directional arrow on the cursor. Should be FALSE unless the cursor is split.

Since 2.4

GtkBorder

```
typedef struct {
  gint left;
  gint right;
  gint top;
  gint bottom;
} GtkBorder;
```

---

gtk_border_copy ()

```
GtkBorder*  gtk_border_copy                 (const GtkBorder *border_);
```

Copies a GtkBorder structure.

*border_* :	a GtkBorder.
Returns :	a copy of *border_*.

---

gtk_border_free ()

```
void        gtk_border_free                 (GtkBorder *border_);
```

Frees a GtkBorder structure.

*border_* : a GtkBorder.

GtkRcProperty

```
typedef struct {
  /* quark-ified property identifier like "GtkScrollbar::spacing" */
  GQuark type_name;
  GQuark property_name;

  /* fields similar to GtkSettingsValue */
  gchar *origin;
  GValue value;
} GtkRcProperty;
```

GtkRcPropertyParser ()

```
gboolean    (*GtkRcPropertyParser)          (const GParamSpec *pspec,
                                             const GString *rc_string,
                                             GValue *property_value);
```

pspec :
rc_string :
property_value :
Returns :

Signals

The "realize" signal

```
void        user_function                   (GtkStyle *style,
```

```
                                                gpointer user_data);
```

Emitted when the style has been initialized for a particular colormap and depth. Connecting to this signal is probably seldom useful since most of the time applications and widgets only deal with styles that have been already realized.

style : the object which received the signal
user_data : user data set when the signal handler was connected.

Since 2.4

The "unrealize" signal

```
void        user_function                   (GtkStyle *style,
                                             gpointer user_data);
```

Emitted when the aspects of the style specific to a particular colormap and depth are being cleaned up. A connection to this signal can be useful if a widget wants to cache objects like a GdkGC as object data on GtkStyle. This signal provides a convenient place to free such cached objects.

style : the object which received the signal
user_data : user data set when the signal handler was connected.

Since 2.4

<< Graphics Contexts **Selections >>**

Selections

Selections — Functions for handling inter-process communication via selections

Synopsis

```
#include <gtk/gtk.h>

            GtkTargetEntry;
            GtkTargetList;
            GtkTargetPair;
GtkTargetList* gtk_target_list_new          (const GtkTargetEntry *targets,
                                             guint ntargets);
void        gtk_target_list_ref             (GtkTargetList *list);
void        gtk_target_list_unref           (GtkTargetList *list);
void        gtk_target_list_add             (GtkTargetList *list,
                                             GdkAtom target,
                                             guint flags,
                                             guint info);
void        gtk_target_list_add_table       (GtkTargetList *list,
                                             const GtkTargetEntry *targets,
                                             guint ntargets);
void        gtk_target_list_add_text_targets
                                            (GtkTargetList *list,
                                             guint info);
void        gtk_target_list_add_image_targets
                                            (GtkTargetList *list,
                                             guint info,
                                             gboolean writable);
void        gtk_target_list_add_uri_targets (GtkTargetList *list,
                                             guint info);
void        gtk_target_list_remove          (GtkTargetList *list,
                                             GdkAtom target);
gboolean    gtk_target_list_find            (GtkTargetList *list,
                                             GdkAtom target,
                                             guint *info);
gboolean    gtk_selection_owner_set         (GtkWidget *widget,
```

```
                                                GdkAtom selection,
                                                guint32 time_);
gboolean    gtk_selection_owner_set_for_display
                                               (GdkDisplay *display,
                                                GtkWidget *widget,
                                                GdkAtom selection,
                                                guint32 time_);
void        gtk_selection_add_target           (GtkWidget *widget,
                                                GdkAtom selection,
                                                GdkAtom target,
                                                guint info);
void        gtk_selection_add_targets          (GtkWidget *widget,
                                                GdkAtom selection,
                                                const GtkTargetEntry *targets,
                                                guint ntargets);
void        gtk_selection_clear_targets        (GtkWidget *widget,
                                                GdkAtom selection);
gboolean    gtk_selection_convert              (GtkWidget *widget,
                                                GdkAtom selection,
                                                GdkAtom target,
                                                guint32 time_);
void        gtk_selection_data_set             (GtkSelectionData *selection_data,
                                                GdkAtom type,
                                                gint format,
                                                const guchar *data,
                                                gint length);
gboolean    gtk_selection_data_set_text        (GtkSelectionData *selection_data,
                                                const gchar *str,
                                                gint len);
guchar*     gtk_selection_data_get_text        (GtkSelectionData *selection_data);
gboolean    gtk_selection_data_set_pixbuf      (GtkSelectionData *selection_data,
                                                GdkPixbuf *pixbuf);
GdkPixbuf*  gtk_selection_data_get_pixbuf      (GtkSelectionData *selection_data);
gboolean    gtk_selection_data_set_uris        (GtkSelectionData *selection_data,
                                                gchar **uris);
gchar**     gtk_selection_data_get_uris        (GtkSelectionData *selection_data);
gboolean    gtk_selection_data_get_targets     (GtkSelectionData *selection_data,
                                                GdkAtom **targets,
                                                gint *n_atoms);
gboolean    gtk_selection_data_targets_include_text
                                               (GtkSelectionData *selection_data);
void        gtk_selection_remove_all           (GtkWidget *widget);
gboolean    gtk_selection_clear                (GtkWidget *widget,
                                                GdkEventSelection *event);
```

```
GtkSelectionData* gtk_selection_data_copy   (GtkSelectionData *data);
void        gtk_selection_data_free         (GtkSelectionData *data);
```

Description

The selection mechanism provides the basis for different types of communication between processes. In particular, drag and drop and GtkClipboard work via selections. You will very seldom or never need to use most of the functions in this section directly; GtkClipboard provides a nicer interface to the same functionality.

Some of the datatypes defined this section are used in the GtkClipboard and drag-and-drop API's as well. The GtkTargetEntry structure and GtkTargetList objects represent lists of data types that are supported when sending or receiving data. The GtkSelectionData object is used to store a chunk of data along with the data type and other associated information.

Details

GtkTargetEntry

```
typedef struct {
  gchar *target;
  guint  flags;
  guint  info;
} GtkTargetEntry;
```

A GtkTargetEntry structure represents a single type of data than can be supplied for by a widget for a selection or for supplied or received during drag-and-drop. It contains a string representing the drag type, a flags field (used only for drag and drop - see GtkTargetFlags), and an application assigned integer ID. The integer ID will later be passed as a signal parameter for signals like "selection_get". It allows the application to identify the target type without extensive string compares.

GtkTargetList

```
typedef struct {
  GList *list;
  guint ref_count;
} GtkTargetList;
```

A GtkTargetList structure is a reference counted list of GtkTargetPair. It is used to represent the same information as a

table of GtkTargetEntry, but in an efficient form. This structure should be treated as opaque.

GtkTargetPair

```
typedef struct {
  GdkAtom   target;
  guint     flags;
  guint     info;
} GtkTargetPair;
```

Internally used structure in the drag-and-drop and selection handling code.

---

gtk_target_list_new ()

```
GtkTargetList* gtk_target_list_new          (const GtkTargetEntry *targets,
                                             guint ntargets);
```

Creates a new GtkTargetList from an array of GtkTargetEntry.

targets : Pointer to an array of GtkTargetEntry
ntargets : number of entries in *targets*.
Returns : the new GtkTargetList.

---

gtk_target_list_ref ()

```
void        gtk_target_list_ref             (GtkTargetList *list);
```

Increases the reference count of a GtkTargetList by one.

list : a GtkTargetList

---

gtk_target_list_unref ()

```
void        gtk_target_list_unref           (GtkTargetList *list);
```

Decreases the reference count of a GtkTargetList by one. If the resulting reference count is zero, frees the list.

list : a GtkTargetList

---

gtk_target_list_add ()

```
void        gtk_target_list_add             (GtkTargetList *list,
                                             GdkAtom target,
                                             guint flags,
                                             guint info);
```

Adds another target to a GtkTargetList.

list : a GtkTargetList
target : the interned atom representing the target
flags : the flags for this target
info : an ID that will be passed back to the application

---

gtk_target_list_add_table ()

```
void        gtk_target_list_add_table       (GtkTargetList *list,
                                             const GtkTargetEntry *targets,
                                             guint ntargets);
```

Adds a table of GtkTargetEntry into a target list.

list : a GtkTargetList
targets : the table of GtkTargetEntry
ntargets : number of targets in the table

---

gtk_target_list_add_text_targets ()

```
void        gtk_target_list_add_text_targets
                                            (GtkTargetList *list,
                                             guint info);
```

Adds the text targets supported by GtkSelection to the target list. All targets are added with the same *info*.

list : a GtkTargetList
info : an ID that will be passed back to the application

Since 2.6

---

gtk_target_list_add_image_targets ()

```
void        gtk_target_list_add_image_targets
                                            (GtkTargetList *list,
                                             guint info,
                                             gboolean writable);
```

Adds the image targets supported by GtkSelection to the target list. All targets are added with the same *info*.

list : a GtkTargetList
info : an ID that will be passed back to the application
writable : whether to add only targets for which GTK+ knows how to convert a pixbuf into the format

Since 2.6

---

gtk_target_list_add_uri_targets ()

```
void        gtk_target_list_add_uri_targets (GtkTargetList *list,
                                             guint info);
```

Adds the URI targets supported by GtkSelection to the target list. All targets are added with the same *info*.

list : a GtkTargetList
info : an ID that will be passed back to the application

Since 2.6

---

gtk_target_list_remove ()

```
void        gtk_target_list_remove          (GtkTargetList *list,
                                             GdkAtom target);
```

Removes a target from a target list.

list : a GtkTargetList
target : the interned atom representing the target

---

gtk_target_list_find ()

```
gboolean    gtk_target_list_find            (GtkTargetList *list,
                                             GdkAtom target,
                                             guint *info);
```

Looks up a given target in a GtkTargetList.

list : a GtkTargetList
target : an interned atom representing the target to search for
info : a pointer to the location to store application info for target
Returns : TRUE if the target was found, otherwise FALSE

---

gtk_selection_owner_set ()

```
gboolean    gtk_selection_owner_set         (GtkWidget *widget,
                                             GdkAtom selection,
                                             guint32 time_);
```

Claims ownership of a given selection for a particular widget, or, if *widget* is NULL, release ownership of the selection.

widget : a GtkWidget, or NULL.
selection : an interned atom representing the selection to claim
*time_* : timestamp with which to claim the selection
Returns : TRUE if the operation succeeded

---

gtk_selection_owner_set_for_display ()

```
gboolean    gtk_selection_owner_set_for_display
                                            (GdkDisplay *display,
                                             GtkWidget *widget,
                                             GdkAtom selection,
                                             guint32 time_);
```

Claim ownership of a given selection for a particular widget, or, if *widget* is NULL, release ownership of the selection.

display : the Gdkdisplay where the selection is set
widget : new selection owner (a GdkWidget), or NULL.
selection : an interned atom representing the selection to claim.
*time_* : timestamp with which to claim the selection
Returns : TRUE if the operation succeeded

Since 2.2

---

gtk_selection_add_target ()

```
void        gtk_selection_add_target        (GtkWidget *widget,
                                             GdkAtom selection,
                                             GdkAtom target,
                                             guint info);
```

Adds specified target to the list of supported targets for a given widget and selection.

widget : a GtkTarget
selection : the selection
target : target to add.

info : A unsigned integer which will be passed back to the application.

---

gtk_selection_add_targets ()

```
void        gtk_selection_add_targets       (GtkWidget *widget,
                                             GdkAtom selection,
                                             const GtkTargetEntry *targets,
                                             guint ntargets);
```

Adds a table of targets to the list of supported targets for a given widget and selection.

widget : a GtkWidget
selection : the selection
targets : a table of targets to add
ntargets : number of entries in *targets*

---

gtk_selection_clear_targets ()

```
void        gtk_selection_clear_targets     (GtkWidget *widget,
                                             GdkAtom selection);
```

Remove all targets registered for the given selection for the widget.

widget : a GtkWidget
selection : an atom representing a selection

---

gtk_selection_convert ()

```
gboolean    gtk_selection_convert           (GtkWidget *widget,
                                             GdkAtom selection,
                                             GdkAtom target,
                                             guint32 time_);
```

Requests the contents of a selection. When received, a "selection_received" signal will be generated.

widget :	The widget which acts as requestor
selection :	Which selection to get
target :	Form of information desired (e.g., STRING)
*time_* :	Time of request (usually of triggering event) In emergency, you could use GDK_CURRENT_TIME
Returns :	TRUE if requested succeeded. FALSE if we could not process request. (e.g., there was already a request in process for this widget).

---

gtk_selection_data_set ()

```
void        gtk_selection_data_set          (GtkSelectionData *selection_data,
                                             GdkAtom type,
                                             gint format,
                                             const guchar *data,
                                             gint length);
```

Stores new data into a GtkSelectionData object. Should *only* be called from a selection handler callback. Zero-terminates the stored data.

selection_data :	
type :	the type of selection data
format :	format (number of bits in a unit)
data :	pointer to the data (will be copied)
length :	length of the data

---

gtk_selection_data_set_text ()

```
gboolean    gtk_selection_data_set_text     (GtkSelectionData *selection_data,
                                             const gchar *str,
                                             gint len);
```

Sets the contents of the selection from a UTF-8 encoded string. The string is converted to the form determined by *selection_data->target*.

selection_data :	a GtkSelectionData
str :	a UTF-8 string
len :	the length of *str*, or -1 if *str* is nul-terminated.

Returns : TRUE if the selection was successfully set, otherwise FALSE.

---

gtk_selection_data_get_text ()

```
guchar*     gtk_selection_data_get_text     (GtkSelectionData *selection_data);
```

Gets the contents of the selection data as a UTF-8 string.

selection_data : a GtkSelectionData

Returns : if the selection data contained a recognized text type and it could be converted to UTF-8, a newly allocated string containing the converted text, otherwise NULL. If the result is non-NULL it must be freed with g_free().

---

gtk_selection_data_set_pixbuf ()

```
gboolean    gtk_selection_data_set_pixbuf   (GtkSelectionData *selection_data,
                                             GdkPixbuf *pixbuf);
```

Sets the contents of the selection from a GdkPixbuf The pixbuf is converted to the form determined by *selection_data->target*.

selection_data : a GtkSelectionData

pixbuf : a GdkPixbuf

Returns : TRUE if the selection was successfully set, otherwise FALSE.

Since 2.6

---

gtk_selection_data_get_pixbuf ()

```
GdkPixbuf*  gtk_selection_data_get_pixbuf   (GtkSelectionData *selection_data);
```

Gets the contents of the selection data as a GdkPixbuf.

selection_data : a GtkSelectionData

Returns : if the selection data contained a recognized image type and it could be converted to a GdkPixbuf, a newly allocated pixbuf is returned, otherwise NULL. If the result is non-NULL it must be freed with g_object_unref().

Since 2.6

---

gtk_selection_data_set_uris ()

```
gboolean    gtk_selection_data_set_uris     (GtkSelectionData *selection_data,
                                             gchar **uris);
```

Sets the contents of the selection from a list of URIs. The string is converted to the form determined by *selection_data->target*.

selection_data : a GtkSelectionData

uris : a NULL-terminated array of strings hilding URIs

Returns : TRUE if the selection was successfully set, otherwise FALSE.

Since 2.6

---

gtk_selection_data_get_uris ()

```
gchar**     gtk_selection_data_get_uris     (GtkSelectionData *selection_data);
```

Gets the contents of the selection data as array of URIs.

selection_data : a GtkSelectionData

Returns : if the selection data contains a list of URIs, a newly allocated NULL-terminated string array containing the URIs, otherwise NULL. If the result is non-NULL it must be freed with g_strfreev().

Since 2.6

---

gtk_selection_data_get_targets ()

```
gboolean    gtk_selection_data_get_targets  (GtkSelectionData *selection_data,
                                             GdkAtom **targets,
                                             gint *n_atoms);
```

Gets the contents of *selection_data* as an array of targets. This can be used to interpret the results of getting the standard TARGETS target that is always supplied for any selection.

selection_data : a GtkSelectionData object

targets : location to store an array of targets. The result stored here must be freed with g_free().

n_atoms : location to store number of items in *targets*.

Returns : TRUE if *selection_data* contains a valid array of targets, otherwise FALSE.

---

gtk_selection_data_targets_include_text ()

```
gboolean    gtk_selection_data_targets_include_text
                                            (GtkSelectionData *selection_data);
```

Given a GtkSelectionData object holding a list of targets, determines if any of the targets in *targets* can be used to provide text.

selection_data : a GtkSelectionData object

Returns : TRUE if *selection_data* holds a list of targets, and a suitable target for text is included, otherwise FALSE.

---

gtk_selection_remove_all ()

```
void        gtk_selection_remove_all        (GtkWidget *widget);
```

Removes all handlers and unsets ownership of all selections for a widget. Called when widget is being destroyed. This function will not generally be called by applications.

widget : a GtkWidget

---

gtk_selection_clear ()

```
gboolean    gtk_selection_clear             (GtkWidget *widget,
                                             GdkEventSelection *event);
```

Warning

`gtk_selection_clear` is deprecated and should not be used in newly-written code. Instead of calling this function, chain up from your selection_clear_event handler. Calling this function from any other context is illegal.

The default handler for the GtkWidget::selection_clear_event signal.

`widget` : a GtkWidget
`event` : the event
Returns : TRUE if the event was handled, otherwise false

Since 2.2

---

gtk_selection_data_copy ()

```
GtkSelectionData* gtk_selection_data_copy   (GtkSelectionData *data);
```

Makes a copy of a GtkSelectionData structure and its data.

`data` : a pointer to a GtkSelectionData structure.
Returns : a pointer to a copy of *`data`*.

---

gtk_selection_data_free ()

```
void        gtk_selection_data_free         (GtkSelectionData *data);
```

Frees a GtkSelectionData structure returned from `gtk_selection_data_copy()`.

`data` : a pointer to a GtkSelectionData structure.

See Also

GtkWidget

Much of the operation of selections happens via signals for GtkWidget. In particular, if you are using the functions in this section, you may need to pay attention to ::selection_get, ::selection_received, and :selection_clear_event signals.

<< Styles **Version Information >>**

Version Information

Version Information — Variables and functions to check the GTK+ version

Synopsis

```
#include <gtk/gtk.h>

extern      const guint gtk_major_version;
extern      const guint gtk_minor_version;
extern      const guint gtk_micro_version;
extern      const guint gtk_binary_age;
extern      const guint gtk_interface_age;
gchar*      gtk_check_version               (guint required_major,
                                             guint required_minor,
                                             guint required_micro);

#define     GTK_MAJOR_VERSION
#define     GTK_MINOR_VERSION
#define     GTK_MICRO_VERSION
#define     GTK_BINARY_AGE
#define     GTK_INTERFACE_AGE
#define     GTK_CHECK_VERSION               (major,minor,micro)
```

Description

GTK+ provides version information, primarily useful in configure checks for builds that have a configure script. Applications will not typically use the features described here.

Details

gtk_major_version

```
extern const guint gtk_major_version;
```

The major version number of the GTK+ library. (e.g. in GTK+ version 1.2.5 this is 1.)

This variable is in the library, so represents the GTK+ library you have linked against. Contrast with the GTK_MAJOR_VERSION macro, which represents the major version of the GTK+ headers you have included.

---

gtk_minor_version

```
extern const guint gtk_minor_version;
```

The minor version number of the GTK+ library. (e.g. in GTK+ version 1.2.5 this is 2.)

This variable is in the library, so represents the GTK+ library you have linked against. Contrast with the GTK_MINOR_VERSION macro, which represents the minor version of the GTK+ headers you have included.

---

gtk_micro_version

```
extern const guint gtk_micro_version;
```

The micro version number of the GTK+ library. (e.g. in GTK+ version 1.2.5 this is 5.)

This variable is in the library, so represents the GTK+ library you have linked against. Contrast with the GTK_MICRO_VERSION macro, which represents the micro version of the GTK+ headers you have included.

---

gtk_binary_age

```
extern const guint gtk_binary_age;
```

This is the binary age passed to libtool. If libtool means nothing to you, don't worry about it. ;-)

---

gtk_interface_age

```
extern const guint gtk_interface_age;
```

This is the interface age passed to libtool. If libtool means nothing to you, don't worry about it. ;-)

---

gtk_check_version ()

```
gchar*      gtk_check_version                (guint required_major,
                                              guint required_minor,
                                              guint required_micro);
```

Checks that the GTK+ library in use is compatible with the given version. Generally you would pass in the constants GTK_MAJOR_VERSION, GTK_MINOR_VERSION, GTK_MICRO_VERSION as the three arguments to this function; that produces a check that the library in use is compatible with the version of GTK+ the application or module was compiled against.

Compatibility is defined by two things: first the version of the running library is newer than the version *required_major.required_minor.required_micro*. Second the running library must be binary compatible with the version *required_major.required_minor.required_micro* (same major version.)

This function is primarily for GTK+ modules; the module can call this function to check that it wasn't loaded into an incompatible version of GTK+. However, such a a check isn't completely reliable, since the module may be linked against an old version of GTK+ and calling the old version of `gtk_check_version()`, but still get loaded into an application using a newer version of GTK+.

required_major : the required major version.

required_minor : the required major version.

required_micro : the required major version.

Returns : `NULL` if the GTK+ library is compatible with the given version, or a string describing the version mismatch. The returned string is owned by GTK+ and should not be modified or freed.

GTK_MAJOR_VERSION

```
#define GTK_MAJOR_VERSION                                (2)
```

Like gtk_major_version, but from the headers used at application compile time, rather than from the library linked against at application run time.

GTK_MINOR_VERSION

```
#define GTK_MINOR_VERSION                                (5)
```

Like gtk_minor_version, but from the headers used at application compile time, rather than from the library linked against at application run time.

GTK_MICRO_VERSION

```
#define GTK_MICRO_VERSION                                (4)
```

Like gtk_micro_version, but from the headers used at application compile time, rather than from the library linked against at application run time.

GTK_BINARY_AGE

```
#define GTK_BINARY_AGE                                  (504)
```

Like gtk_binary_age, but from the headers used at application compile time, rather than from the library linked against at application run time.

GTK_INTERFACE_AGE

```
#define GTK_INTERFACE_AGE                               (0)
```

Like gtk_interface_age, but from the headers used at application compile time, rather than from the library linked against at application run time.

GTK_CHECK_VERSION()

```
#define     GTK_CHECK_VERSION(major,minor,micro)
```

Returns TRUE if the version of the GTK+ header files is the same as the passed-in version.

major : major version (e.g. 1 for version 1.2.5)
minor : minor version (e.g. 2 for version 1.2.5)
micro : micro version (e.g. 5 for version 1.2.5)

<< Selections **Signals >>**

Signals

Signals — Object methods and callbacks

Synopsis

```
#include <gtk/gtk.h>

#define     GTK_SIGNAL_OFFSET
enum        GtkSignalRunType;
guint       gtk_signal_new                  (const gchar *name,
                                             GtkSignalRunType signal_flags,
                                             GtkType object_type,
                                             guint function_offset,
                                             GtkSignalMarshaller marshaller,
                                             GtkType return_val,
                                             guint n_args,
                                             ...);
guint       gtk_signal_newv                 (const gchar *name,
                                             GtkSignalRunType signal_flags,
                                             GtkType object_type,
                                             guint function_offset,
                                             GtkSignalMarshaller marshaller,
                                             GtkType return_val,
                                             guint n_args,
                                             GtkType *args);
#define     gtk_signal_lookup               (name,object_type)
#define     gtk_signal_name                 (signal_id)
void        gtk_signal_emit                 (GtkObject *object,
                                             guint signal_id,
                                             ...);
void        gtk_signal_emit_by_name         (GtkObject *object,
                                             const gchar *name,
                                             ...);
void        gtk_signal_emitv                (GtkObject *object,
                                             guint signal_id,
```

```
                                                GtkArg *args);
void        gtk_signal_emitv_by_name            (GtkObject *object,
                                                 const gchar *name,
                                                 GtkArg *args);
#define     gtk_signal_emit_stop                (object,signal_id)
void        gtk_signal_emit_stop_by_name        (GtkObject *object,
                                                 const gchar *name);
#define     gtk_signal_connect                  (object,name,func,func_data)
#define     gtk_signal_connect_after            (object,name,func,func_data)
#define     gtk_signal_connect_object           (object,name,func,slot_object)
#define     gtk_signal_connect_object_after     (object,name,func,slot_object)
gulong      gtk_signal_connect_full             (GtkObject *object,
                                                 const gchar *name,
                                                 GtkSignalFunc func,
                                                 GtkCallbackMarshal unsupported,
                                                 gpointer data,
                                                 GtkDestroyNotify destroy_func,
                                                 gint object_signal,
                                                 gint after);
void        gtk_signal_connect_while_alive      (GtkObject *object,
                                                 const gchar *name,
                                                 GtkSignalFunc func,
                                                 gpointer func_data,
                                                 GtkObject *alive_object);
void        gtk_signal_connect_object_while_alive
                                                (GtkObject *object,
                                                 const gchar *name,
                                                 GtkSignalFunc func,
                                                 GtkObject *alive_object);
#define     gtk_signal_disconnect               (object,handler_id)
#define     gtk_signal_disconnect_by_func       (object,func,data)
#define     gtk_signal_disconnect_by_data       (object,data)
#define     gtk_signal_handler_block            (object,handler_id)
#define     gtk_signal_handler_block_by_func    (object,func,data)
#define     gtk_signal_handler_block_by_data    (object,data)
#define     gtk_signal_handler_unblock          (object,handler_id)
#define     gtk_signal_handler_unblock_by_func  (object,func,data)
#define     gtk_signal_handler_unblock_by_data  (object,data)
#define     gtk_signal_handler_pending          (object,signal_id,may_be_blocked)
#define
gtk_signal_handler_pending_by_func(object,signal_id,may_be_blocked,func,data)
#define     gtk_signal_default_marshaller
```

Description

The GTK+ signal system merely proxies the GLib signal system now. For future usage, direct use of the GSignal API is recommended, this avoids significant performance hits where GtkArg structures have to be converted into GValues.

What are signals?

Signals are a way to get notification when something happens and to customize object behavior according to the user's needs. Every *signal* is uniquely identified by a name, "class_name::signal_name", where signal_name might be something like "clicked" and class_name might be "GtkButton". Note that some other class may also define a "clicked" callback, so long as it doesn't derive from GtkButton.

When they are created, they are also assigned a unique positive integer, the signal id (1 is the first signal id- 0 is used to flag an error). Each is also tied to an array of types that describes the prototype of the function pointer(s) (handlers) you may connect to the signal. Finally, every signal has a default handler that is given by a function pointer in its class structure: it is run by default whenever the signal is emitted. (It is possible that a signal will be emitted and a user-defined handler will prevent the default handler from being run.)

Signals are used by everyone, but they are only created on a per class basis -- so you should not call call `gtk_signal_new()` unless you are writing a new GtkObject type. However, if you want to make a new signal for an existing type, you may use `gtk_object_class_user_signal_new()` to create a signal that doesn't correspond to a class's builtin methods.

How are signals used?

There are two basic actions in the signal handling game. If you want notification of an event, you must *connect* a function pointer and a data pointer to that signal; the data pointer will be passed as the last argument to the function (so long as you are using the default marshalling functions). You will receive a connection id, a unique positive integer corresponding to that attachment.

Functions that want to notify the user of certain actions, *emit* signals.

Basic Terminology

signal	A class method, e.g. GtkButton::clicked. More precisely it is a unique class-branch/signal-name pair. This means you may not define a signal handler for a class which derives from GtkButton that is called clicked, but it is okay to share signals names if they are separate in the class tree.
default handler	The object's internal method which is invoked when the signal is emitted.
user-defined handler	A function pointer and data connected to a signal (for a particular object). There are really two types: those which are connected normally, and those which are connected by one of the connect_after functions. The connect_after handlers are always run after the default handler. Many toolkits refer to these as *callbacks*.
emission	the whole process of emitting a signal, including the invocation of all the different handler types mentioned above.
signal id	The unique positive (nonzero) integer used to identify a signal. It can be used instead of a name to many functions for a slight performance improvement.
connection id	The unique positive (nonzero) integer used to identify the connection of a user-defined handler to a signal. Notice that it is allowed to connect the same function-pointer/user-data pair twice, so there is no guarantee that a function-pointer/user-data maps to a unique connection id.

A brief note on how they work.

The functions responsible for translating an array of GtkArgs to your C compiler's normal semantics are called Marshallers. They are identified by gtk_marshal_*`return_value__parameter_list`*() for example a C function returning a gboolean and taking a gint can be invoked by using `gtk_marshal_BOOL__INT()`. Not all possibly combinations of return/params are available, of course, so if you are writing a GtkObject with parameters you might have to write a marshaller.

Details

GTK_SIGNAL_OFFSET

```
#define GTK_SIGNAL_OFFSET                                   GTK_STRUCT_OFFSET
```

Warning

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

Use in place of `offsetof()`, which is used if it exists.

enum GtkSignalRunType

```
typedef enum                    /*< flags >*/
{
  GTK_RUN_FIRST      = G_SIGNAL_RUN_FIRST,
  GTK_RUN_LAST       = G_SIGNAL_RUN_LAST,
  GTK_RUN_BOTH       = (GTK_RUN_FIRST | GTK_RUN_LAST),
  GTK_RUN_NO_RECURSE = G_SIGNAL_NO_RECURSE,
  GTK_RUN_ACTION     = G_SIGNAL_ACTION,
  GTK_RUN_NO_HOOKS   = G_SIGNAL_NO_HOOKS
} GtkSignalRunType;
```

Warning

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

These configure the signal's emission. They control whether the signal can be emitted recursively on an object and whether to run the default method before or after the user-defined handlers.

GTK_RUN_FIRST	Run the default handler before the connected user-defined handlers.
GTK_RUN_LAST	Run the default handler after the connected user-defined handlers. (Handlers registered as "after" always run after the default handler though)
GTK_RUN_BOTH	Run the default handler twice, once before the user-defined handlers, and once after.
GTK_RUN_NO_RECURSE	Whether to prevent a handler or hook from reemitting the signal from within itself. Attempts to emit the signal while it is running will result in the signal emission being restarted once it is done with the current processing. You must be careful to avoid having two handlers endlessly reemitting signals, `gtk_signal_n_emissions()` can be helpful.
GTK_RUN_ACTION	The signal is an action you can invoke without any particular setup or cleanup. The signal is treated no differently, but some other code can determine if the signal is appropriate to delegate to user control. For example, key binding sets only allow bindings of ACTION signals to keystrokes.
GTK_RUN_NO_HOOKS	This prevents the connection of emission hooks to the signal.

gtk_signal_new ()

```
guint       gtk_signal_new                  (const gchar *name,
                                             GtkSignalRunType signal_flags,
                                             GtkType object_type,
                                             guint function_offset,
                                             GtkSignalMarshaller marshaller,
                                             GtkType return_val,
                                             guint n_args,
                                             ...);
```

Warning

`gtk_signal_new` is deprecated and should not be used in newly-written code. Use `g_signal_new()` instead.

Creates a new signal type. (This is usually done in the class initializer.)

name :	the event name for the signal, e.g. "clicked".
signal_flags :	a combination of GTK_RUN flags specifying detail of when the default handler is to be invoked. You should at least specify GTK_RUN_FIRST or GTK_RUN_LAST.
object_type :	the type of object this signal pertains to. It will also pertain to derivers of this type automatically.
function_offset :	How many bytes the function pointer is in the class structure for this type. Used to invoke a class method generically.
marshaller :	the function to translate between an array of GtkArgs and the native calling convention. Usually they are identified just by the type of arguments they take: for example, `gtk_marshal_BOOL__STRING()` describes a marshaller which takes a string and returns a boolean value.
return_val :	the type of return value, or GTK_TYPE_NONE for a signal without a return value.
n_args :	the number of parameter the handlers may take.
... :	a list of GTK_TYPE_*, one for each parameter.
Returns :	the signal id.

---

gtk_signal_newv ()

```
guint       gtk_signal_newv                 (const gchar *name,
                                             GtkSignalRunType signal_flags,
```

```
                                                GtkType object_type,
                                                guint function_offset,
                                                GtkSignalMarshaller marshaller,
                                                GtkType return_val,
                                                guint n_args,
                                                GtkType *args);
```

Warning

`gtk_signal_newv` is deprecated and should not be used in newly-written code. Use `g_signal_newv()` instead.

Creates a new signal type. (This is usually done in a class initializer.)

This function take the types as an array, instead of a list following the arguments. Otherwise the same as `gtk_signal_new()`.

name :	the name of the signal to create.
signal_flags :	see `gtk_signal_new()`.
object_type :	the type of GtkObject to associate the signal with.
function_offset :	how many bytes the function pointer is in the class structure for this type.
marshaller :	
return_val :	the type of the return value, or GTK_TYPE_NONE if you don't want a return value.
n_args :	the number of parameters to the user-defined handlers.
args :	an array of GtkTypes, describing the prototype to the callbacks.
Returns :	the signal id.

---

gtk_signal_lookup()

```
#define     gtk_signal_lookup(name,object_type)
```

Warning

`gtk_signal_lookup` is deprecated and should not be used in newly-written code. Use `g_signal_lookup()` instead.

Given the name of the signal and the type of object it connects to, get the signal's identifying integer. Emitting the signal by number is somewhat faster than using the name each time.

It also tries the ancestors of the given type.

name :	the signal's name, e.g. clicked.
object_type :	the type that the signal operates on, e.g. GTK_TYPE_BUTTON.
Returns :	the signal's identifying number, or 0 if no signal was found.

---

gtk_signal_name()

```
#define     gtk_signal_name(signal_id)
```

Warning

`gtk_signal_name` is deprecated and should not be used in newly-written code. Use `g_signal_name()` instead.

Given the signal's identifier, finds its name.

Two different signals may have the same name, if they have differing types.

signal_id :	the signal's identifying number.
Returns :	the signal name, or `NULL` if the signal number was invalid.

---

gtk_signal_emit ()

```
void        gtk_signal_emit                     (GtkObject *object,
                                                 guint signal_id,
                                                 ...);
```

Warning

`gtk_signal_emit` is deprecated and should not be used in newly-written code. Use `g_signal_emit()` instead.

Emits a signal. This causes the default handler and user-defined handlers to be run.

Here is what `gtk_signal_emit()` does:

1. Calls the default handler and the user-connected handlers. The default handler will be called first if GTK_RUN_FIRST is set, and last if GTK_RUN_LAST is set.

2. Calls all handlers connected with the "after" flag set.

object : the object that emits the signal.
signal_id : the signal identifier.
... : the parameters to the function, followed by a pointer to the return type, if any.

---

gtk_signal_emit_by_name ()

```
void        gtk_signal_emit_by_name         (GtkObject *object,
                                             const gchar *name,
                                             ...);
```

Warning

`gtk_signal_emit_by_name` is deprecated and should not be used in newly-written code. Use `g_signal_emit_by_name()` instead.

Emits a signal. This causes the default handler and user-connected handlers to be run.

object : the object that emits the signal.
name : the name of the signal.
... : the parameters to the function, followed by a pointer to the return type, if any.

---

gtk_signal_emitv ()

```
void        gtk_signal_emitv                (GtkObject *object,
                                             guint signal_id,
                                             GtkArg *args);
```

Warning

`gtk_signal_emitv` is deprecated and should not be used in newly-written code. Use `g_signal_emitv()` instead.

Emits a signal. This causes the default handler and user-connected handlers to be run. This differs from `gtk_signal_emit()` by taking an array of GtkArgs instead of using C's varargs mechanism.

object :	the object to emit the signal to.
signal_id :	the signal identifier.
args :	an array of GtkArgs, one for each parameter, followed by one which is a pointer to the return type.

---

gtk_signal_emitv_by_name ()

```
void        gtk_signal_emitv_by_name        (GtkObject *object,
                                             const gchar *name,
                                             GtkArg *args);
```

Warning

`gtk_signal_emitv_by_name` is deprecated and should not be used in newly-written code. Use `g_signal_emitv()` and `g_signal_lookup()` instead.

Emits a signal by name. This causes the default handler and user-connected handlers to be run. This differs from `gtk_signal_emit()` by taking an array of GtkArgs instead of using C's varargs mechanism.

object :	the object to emit the signal to.
name :	the name of the signal.
args :	an array of GtkArgs, one for each parameter, followed by one which is a pointer to the return type.

---

gtk_signal_emit_stop()

```
#define     gtk_signal_emit_stop(object,signal_id)
```

Warning

`gtk_signal_emit_stop` is deprecated and should not be used in newly-written code. Use `g_signal_stop_emission()` instead.

This function aborts a signal's current emission.

It will prevent the default method from running, if the signal was GTK_RUN_LAST and you connected normally (i.e. without the "after" flag).

It will print a warning if used on a signal which isn't being emitted.

object : the object whose signal handlers you wish to stop.
signal_id : the signal identifier, as returned by g_signal_lookup().

---

gtk_signal_emit_stop_by_name ()

```
void        gtk_signal_emit_stop_by_name    (GtkObject *object,
                                             const gchar *name);
```

Warning

gtk_signal_emit_stop_by_name is deprecated and should not be used in newly-written code. Use g_signal_stop_emission_by_name() instead.

This function aborts a signal's current emission.

It is just like gtk_signal_emit_stop() except it will lookup the signal id for you.

object : the object whose signal handlers you wish to stop.
name : the name of the signal you wish to stop.

---

gtk_signal_connect()

```
#define     gtk_signal_connect(object,name,func,func_data)
```

Warning

gtk_signal_connect is deprecated and should not be used in newly-written code. Use g_signal_connect() instead.

Attaches a function pointer and user data to a signal for a particular object.

The GtkSignalFunction takes a GtkObject as its first parameter. It will be the same object as the one you're connecting the hook to. The *func_data* will be passed as the last parameter to the hook.

All else being equal, signal handlers are invoked in the order connected (see gtk_signal_emit() for the other details of which order things are called in).

Here is how one passes an integer as user data, for when you just want to specify a constant int as parameter to your function:

```
static void button_clicked_int (GtkButton* button, gpointer func_data)
{
        g_print ("button pressed: %d\n", GPOINTER_TO_INT (func_data));
}

/* By calling this function, you will make the g_print above
 * execute, printing the number passed as `to_print'. */
static void attach_print_signal (GtkButton* button, gint to_print)
{
        gtk_signal_connect (GTK_OBJECT (button), "clicked",
                GTK_SIGNAL_FUNC (button_clicked_int),
                GINT_TO_POINTER (to_print));
}
```

object :	the object associated with the signal, e.g. if a button is getting pressed, this is that button.
name :	name of the signal.
func :	function pointer to attach to the signal.
func_data :	value to pass as to your function (through the marshaller).
Returns :	the connection id.

---

gtk_signal_connect_after()

```
#define     gtk_signal_connect_after(object,name,func,func_data)
```

Warning

gtk_signal_connect_after is deprecated and should not be used in newly-written code. Use g_signal_connect_after() instead.

Attaches a function pointer and user data to a signal so that this handler will be called after the other handlers.

object :	the object associated with the signal.
name :	name of the signal.
func :	function pointer to attach to the signal.
func_data :	value to pass as to your function (through the marshaller).
Returns :	the unique identifier for this attachment: the connection id.

---

gtk_signal_connect_object()

```
#define     gtk_signal_connect_object(object,name,func,slot_object)
```

Warning

`gtk_signal_connect_object` is deprecated and should not be used in newly-written code. Use `g_signal_connect_swapped()` instead.

This function is for registering a callback that will call another object's callback. That is, instead of passing the object which is responsible for the event as the first parameter of the callback, it is switched with the user data (so the object which emits the signal will be the last parameter, which is where the user data usually is).

This is useful for passing a standard function in as a callback. For example, if you wanted a button's press to `gtk_widget_show()` some widget, you could write:

```
gtk_signal_connect_object (button, "clicked", gtk_widget_show, window);
```

object :	the object which emits the signal.
name :	the name of the signal.
func :	the function to callback.
slot_object :	the object to pass as the first parameter to func. (Though it pretends to take an object, you can really pass any gpointer as the slot_object .)
Returns :	the connection id.

---

gtk_signal_connect_object_after()

```
#define     gtk_signal_connect_object_after(object,name,func,slot_object)
```

Warning

`gtk_signal_connect_object_after` is deprecated and should not be used in newly-written code. Use `g_signal_connect_data()` instead, passing `G_CONNECT_AFTER|G_CONNECT_SWAPPED` as *`connect_flags`*.

Attaches a signal hook to a signal, passing in an alternate object as the first parameter, and guaranteeing that the default handler and all normal handlers are called first.

`object` :	the object associated with the signal.
`name` :	name of the signal.
`func` :	function pointer to attach to the signal.
`slot_object` :	the object to pass as the first parameter to func.
Returns :	the connection id.

---

gtk_signal_connect_full ()

```
gulong      gtk_signal_connect_full         (GtkObject *object,
                                             const gchar *name,
                                             GtkSignalFunc func,
                                             GtkCallbackMarshal unsupported,
                                             gpointer data,
                                             GtkDestroyNotify destroy_func,
                                             gint object_signal,
                                             gint after);
```

Warning

`gtk_signal_connect_full` is deprecated and should not be used in newly-written code. Use `g_signal_connect_data()` instead.

Attaches a function pointer and user data to a signal with more control.

`object` :	the object which emits the signal. For example, a button in the button press signal.
`name` :	the name of the signal.
`func` :	function pointer to attach to the signal.
`unsupported` :	
`data` :	the user data associated with the function.

destroy_func :	function to call when this particular hook is disconnected.
object_signal :	whether this is an object signal-- basically an "object signal" is one that wants its user_data and object fields switched, which is useful for calling functions which operate on another object primarily.
after :	whether to invoke the user-defined handler after the signal, or to let the signal's default behavior preside (i.e. depending on GTK_RUN_FIRST and GTK_RUN_LAST).
Returns :	the connection id.

---

gtk_signal_connect_while_alive ()

```
void        gtk_signal_connect_while_alive  (GtkObject *object,
                                             const gchar *name,
                                             GtkSignalFunc func,
                                             gpointer func_data,
                                             GtkObject *alive_object);
```

Warning

`gtk_signal_connect_while_alive` is deprecated and should not be used in newly-written code. Use `g_signal_connect_object()` instead.

Attaches a function pointer and another GtkObject to a signal.

This function takes an object whose "destroy" signal should be trapped. That way, you don't have to clean up the signal handler when you destroy the object. It is a little less efficient though.

(Instead you may call `gtk_signal_disconnect_by_data()`, if you want to explicitly delete all attachments to this object. This is perhaps not recommended since it could be confused with an integer masquerading as a pointer (through `GINT_TO_POINTER()`).)

object :	the object that emits the signal.
name :	name of the signal.
func :	function pointer to attach to the signal.
func_data :	pointer to pass to func.
alive_object :	object whose death should cause the handler connection to be destroyed.

---

gtk_signal_connect_object_while_alive ()

```
void        gtk_signal_connect_object_while_alive
                                            (GtkObject *object,
                                             const gchar *name,
                                             GtkSignalFunc func,
                                             GtkObject *alive_object);
```

Warning

`gtk_signal_connect_object_while_alive` is deprecated and should not be used in newly-written code. Use `g_signal_connect_object()` instead, passing `G_CONNECT_SWAPPED` as *`connect_flags`*.

These signal connectors are for signals which refer to objects, so they must not be called after the object is deleted.

Unlike `gtk_signal_connect_while_alive()`, this swaps the object and user data, making it suitable for use with functions which primarily operate on the user data.

This function acts just like `gtk_signal_connect_object()` except it traps the "destroy" signal to prevent you from having to clean up the handler.

`object` :	the object associated with the signal.
`name` :	name of the signal.
`func` :	function pointer to attach to the signal.
`alive_object` :	the user data, which must be an object, whose destruction should signal the removal of this signal.

---

gtk_signal_disconnect()

```
#define     gtk_signal_disconnect(object,handler_id)
```

Warning

`gtk_signal_disconnect` is deprecated and should not be used in newly-written code. Use `g_signal_handler_disconnect()` instead.

Destroys a user-defined handler connection.

`object` :	the object which the handler pertains to.

handler_id : the connection id.

---

gtk_signal_disconnect_by_func()

```
#define     gtk_signal_disconnect_by_func(object,func,data)
```

Warning

`gtk_signal_disconnect_by_func` is deprecated and should not be used in newly-written code. Use `g_signal_handlers_disconnect_by_func()` instead.

Destroys all connections for a particular object, with the given function-pointer and user-data.

object : the object which emits the signal.
func : the function pointer to search for.
data : the user data to search for.

---

gtk_signal_disconnect_by_data()

```
#define     gtk_signal_disconnect_by_data(object,data)
```

Warning

`gtk_signal_disconnect_by_data` is deprecated and should not be used in newly-written code. Use `g_signal_handlers_disconnect_matched()` instead.

Destroys all connections for a particular object, with the given user-data.

object : the object which emits the signal.
data : the user data to search for.

---

gtk_signal_handler_block()

```
#define     gtk_signal_handler_block(object,handler_id)
```

Warning

`gtk_signal_handler_block` is deprecated and should not be used in newly-written code. Use `g_signal_handler_block()` instead.

Prevents a user-defined handler from being invoked. All other signal processing will go on as normal, but this particular handler will ignore it.

`object` : the object which emits the signal to block.
`handler_id` : the connection id.

---

gtk_signal_handler_block_by_func()

```
#define     gtk_signal_handler_block_by_func(object,func,data)
```

Warning

`gtk_signal_handler_block_by_func` is deprecated and should not be used in newly-written code. Use `g_signal_handlers_block_by_func()` instead.

Prevents a user-defined handler from being invoked, by reference to the user-defined handler's function pointer and user data. (It may result in multiple hooks being blocked, if you've called connect multiple times.)

`object` : the object which emits the signal to block.
`func` : the function pointer of the handler to block.
`data` : the user data of the handler to block.

---

gtk_signal_handler_block_by_data()

```
#define     gtk_signal_handler_block_by_data(object,data)
```

Warning

`gtk_signal_handler_block_by_data` is deprecated and should not be used in newly-written code. Use `g_signal_handlers_block_matched()` instead.

Prevents all user-defined handlers with a certain user data from being invoked.

object : the object which emits the signal we want to block.
data : the user data of the handlers to block.

---

gtk_signal_handler_unblock()

```
#define     gtk_signal_handler_unblock(object,handler_id)
```

Warning

`gtk_signal_handler_unblock` is deprecated and should not be used in newly-written code. Use `g_signal_handler_unblock()` instead.

Undoes a block, by connection id. Note that undoing a block doesn't necessarily make the hook callable, because if you block a hook twice, you must unblock it twice.

object : the object which emits the signal we want to unblock.
handler_id : the emission handler identifier, as returned by `gtk_signal_connect()`, etc.

---

gtk_signal_handler_unblock_by_func()

```
#define     gtk_signal_handler_unblock_by_func(object,func,data)
```

Warning

`gtk_signal_handler_unblock_by_func` is deprecated and should not be used in newly-written code. Use `g_signal_handlers_unblock_by_func()` instead.

Undoes a block, by function pointer and data. Note that undoing a block doesn't necessarily make the hook callable, because if you block a hook twice, you must unblock it twice.

object : the object which emits the signal we want to unblock.
func : the function pointer to search for.
data : the user data to search for.

---

gtk_signal_handler_unblock_by_data()

```
#define     gtk_signal_handler_unblock_by_data(object,data)
```

Warning

`gtk_signal_handler_unblock_by_data` is deprecated and should not be used in newly-written code. Use `g_signal_handlers_unblock_matched()` instead.

Undoes block(s), to all signals for a particular object with a particular user-data pointer

`object` : the object which emits the signal we want to unblock.
`data` : the user data to search for.

---

gtk_signal_handler_pending()

```
#define     gtk_signal_handler_pending(object,signal_id,may_be_blocked)
```

Warning

`gtk_signal_handler_pending` is deprecated and should not be used in newly-written code. Use `g_signal_has_handler_pending()` instead.

Returns a connection id corresponding to a given signal id and object.

One example of when you might use this is when the arguments to the signal are difficult to compute. A class implementor may opt to not emit the signal if no one is attached anyway, thus saving the cost of building the arguments.

`object` : the object to search for the desired user-defined handler.
`signal_id` : the number of the signal to search for.
`may_be_blocked` : whether it is acceptable to return a blocked handler.
Returns : the connection id, if a connection was found. 0 otherwise.

---

gtk_signal_handler_pending_by_func()

```
#define
gtk_signal_handler_pending_by_func(object,signal_id,may_be_blocked,func,data)
```

Warning

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

Returns a connection id corresponding to a given signal id, object, function pointer and user data.

`object` :	the object to search for the desired handler.
`signal_id` :	the number of the signal to search for.
`may_be_blocked` :	whether it is acceptable to return a blocked handler.
`func` :	the function pointer to search for.
`data` :	the user data to search for.
Returns :	the connection id, if a handler was found. 0 otherwise.

---

gtk_signal_default_marshaller

```
#define gtk_signal_default_marshaller   g_cclosure_marshal_VOID__VOID
```

Warning

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

A marshaller that returns void and takes no extra parameters.

See Also

GtkObject The base class for things which emit signals.

GSignal The GLib signal system.

<< Version Information **Types >>**

Types

Types — Handle run-time type creation

Synopsis

```
#include <gtk/gtk.h>

typedef     GtkType;
typedef     GtkFundamentalType;
#define     GTK_CLASS_NAME                  (class)
#define     GTK_CLASS_TYPE                  (class)
#define     GTK_TYPE_IS_OBJECT              (type)
#define     GTK_TYPE_FUNDAMENTAL_LAST
#define     GTK_TYPE_FUNDAMENTAL_MAX
#define     GTK_STRUCT_OFFSET
#define     GTK_CHECK_CAST
#define     GTK_CHECK_CLASS_CAST
#define     GTK_CHECK_TYPE
#define     GTK_CHECK_CLASS_TYPE
#define     GTK_CHECK_GET_CLASS
#define     GTK_FUNDAMENTAL_TYPE
#define     GTK_SIGNAL_FUNC                 (f)
typedef     GtkClassInitFunc;
typedef     GtkObjectInitFunc;
void        (*GtkSignalFunc)                (void);
gboolean    (*GtkFunction)                  (gpointer data);
void        (*GtkDestroyNotify)             (gpointer data);
void        (*GtkCallbackMarshal)           (GtkObject *object,
                                             gpointer data,
                                             guint n_args,
                                             GtkArg *args);
typedef     GtkSignalMarshaller;
typedef     GtkTypeObject;
            GtkArg;
#define     GTK_VALUE_CHAR                  (a)
#define     GTK_VALUE_UCHAR                 (a)
#define     GTK_VALUE_BOOL                  (a)
```

```
#define     GTK_VALUE_INT                   (a)
#define     GTK_VALUE_UINT                  (a)
#define     GTK_VALUE_LONG                  (a)
#define     GTK_VALUE_ULONG                 (a)
#define     GTK_VALUE_FLOAT                 (a)
#define     GTK_VALUE_DOUBLE                (a)
#define     GTK_VALUE_STRING                (a)
#define     GTK_VALUE_ENUM                  (a)
#define     GTK_VALUE_FLAGS                 (a)
#define     GTK_VALUE_BOXED                 (a)
#define     GTK_VALUE_POINTER               (a)
#define     GTK_VALUE_OBJECT                (a)
#define     GTK_VALUE_SIGNAL                (a)
#define     GTK_RETLOC_CHAR                 (a)
#define     GTK_RETLOC_UCHAR                (a)
#define     GTK_RETLOC_BOOL                 (a)
#define     GTK_RETLOC_INT                  (a)
#define     GTK_RETLOC_UINT                 (a)
#define     GTK_RETLOC_LONG                 (a)
#define     GTK_RETLOC_ULONG                (a)
#define     GTK_RETLOC_FLOAT                (a)
#define     GTK_RETLOC_DOUBLE               (a)
#define     GTK_RETLOC_STRING               (a)
#define     GTK_RETLOC_ENUM                 (a)
#define     GTK_RETLOC_FLAGS                (a)
#define     GTK_RETLOC_BOXED                (a)
#define     GTK_RETLOC_POINTER              (a)
#define     GTK_RETLOC_OBJECT               (a)
            GtkTypeInfo;
typedef     GtkTypeClass;
typedef     GtkEnumValue;
typedef     GtkFlagValue;
void        gtk_type_init                   (GTypeDebugFlags debug_flags);
GtkType     gtk_type_unique                 (GtkType parent_type,
                                             const GtkTypeInfo *gtkinfo);
#define     gtk_type_name                   (type)
#define     gtk_type_from_name              (name)
#define     gtk_type_parent                 (type)
gpointer    gtk_type_class                  (GtkType type);
gpointer    gtk_type_new                    (GtkType type);
#define     gtk_type_is_a                   (type, is_a_type)
GtkEnumValue* gtk_type_enum_get_values      (GtkType enum_type);
GtkFlagValue* gtk_type_flags_get_values     (GtkType flags_type);
GtkEnumValue* gtk_type_enum_find_value      (GtkType enum_type,
                                             const gchar *value_name);
```

```
GtkFlagValue* gtk_type_flags_find_value     (GtkType flags_type,
                                             const gchar *value_name);
```

Description

The GTK+ type system is extensible. Because of that, types have to be managed at runtime.

Details

GtkType

```
typedef GType                   GtkType;
```

GtkType is unique integer identifying the type. The guts of the information about the type is held in a private struct named GtkTypeNode.

GtkFundamentalType

```
typedef GType GtkFundamentalType;
```

Warning

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

GtkFundamentalType is an enumerated type which lists all the possible fundamental types (e.g. char, uchar, int, long, float, etc).

GTK_CLASS_NAME()

```
#define GTK_CLASS_NAME(class)           (g_type_name (G_TYPE_FROM_CLASS (class)))
```

Warning

`GTK_CLASS_NAME` is deprecated and should not be used in newly-written code. Use `g_type_name()` and `G_TYPE_FROM_CLASS()` instead.

Returns the type name of *class*.

class : a GtkTypeClass.

GTK_CLASS_TYPE()

```
#define GTK_CLASS_TYPE(class)           (G_TYPE_FROM_CLASS (class))
```

Warning

GTK_CLASS_TYPE is deprecated and should not be used in newly-written code. Use G_TYPE_FROM_CLASS() instead.

Returns the type of *class*.

class : a GtkTypeClass.

GTK_TYPE_IS_OBJECT()

```
#define GTK_TYPE_IS_OBJECT(type)        (g_type_is_a ((type), GTK_TYPE_OBJECT))
```

Warning

GTK_TYPE_IS_OBJECT is deprecated and should not be used in newly-written code. Use G_TYPE_IS_OBJECT() instead.

Returns TRUE if *type* is a GTK_TYPE_OBJECT.

type : a GtkType.

GTK_TYPE_FUNDAMENTAL_LAST

```
#define GTK_TYPE_FUNDAMENTAL_LAST       (G_TYPE_LAST_RESERVED_FUNDAMENTAL - 1)
```

Warning

GTK_TYPE_FUNDAMENTAL_LAST is deprecated and should not be used in newly-written code. Use G_TYPE_LAST_RESERVED_FUNDAMENTAL - 1 instead.

The highest-numbered structured or flat enumerated type value.

GTK_TYPE_FUNDAMENTAL_MAX

```
#define GTK_TYPE_FUNDAMENTAL_MAX          (G_TYPE_FUNDAMENTAL_MAX)
```

Warning

GTK_TYPE_FUNDAMENTAL_MAX is deprecated and should not be used in newly-written code. Use G_TYPE_FUNDAMENTAL_MAX instead.

The maximum fundamental enumerated type value.

GTK_STRUCT_OFFSET

```
#define GTK_STRUCT_OFFSET       G_STRUCT_OFFSET
```

Warning

GTK_STRUCT_OFFSET is deprecated and should not be used in newly-written code. Use G_STRUCT_OFFSET() instead.

Use in place of offsetof(), which is used if it exists.

GTK_CHECK_CAST

```
#define GTK_CHECK_CAST          G_TYPE_CHECK_INSTANCE_CAST
```

Casts the object in *tobj* into *cast*. If G_DISABLE_CAST_CHECKS is defined, just cast it. Otherwise, check to see if we can cast *tobj* into a *cast*.

GTK_CHECK_CLASS_CAST

```
#define GTK_CHECK_CLASS_CAST    G_TYPE_CHECK_CLASS_CAST
```

Casts the object in *tobj* into *cast*. If G_DISABLE_CAST_CHECKS is defined, just cast it. Otherwise, check to see if we can cast *tobj* into a *cast*.

GTK_CHECK_TYPE

```
#define GTK_CHECK_TYPE          G_TYPE_CHECK_INSTANCE_TYPE
```

Determines whether *type_object* is a type of *otype*.

GTK_CHECK_CLASS_TYPE

```
#define GTK_CHECK_CLASS_TYPE    G_TYPE_CHECK_CLASS_TYPE
```

Determines whether *type_class* is a type of *otype*.

GTK_CHECK_GET_CLASS

```
#define GTK_CHECK_GET_CLASS     G_TYPE_INSTANCE_GET_CLASS
```

Gets the class of *tobj*.

GTK_FUNDAMENTAL_TYPE

```
#define GTK_FUNDAMENTAL_TYPE    G_TYPE_FUNDAMENTAL
```

Warning

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

Converts a GTK+ type into a fundamental type.

GTK_SIGNAL_FUNC()

```
#define GTK_SIGNAL_FUNC(f)          ((GtkSignalFunc) (f))
```

Just a macroized cast into a GtkSignalFunc.

f :

GtkClassInitFunc

```
typedef GBaseInitFunc           GtkClassInitFunc;
```

Warning

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

Defines a function pointer.

GtkObjectInitFunc

```
typedef GInstanceInitFunc       GtkObjectInitFunc;
```

Warning

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

Defines a function pointer.

GtkSignalFunc ()

```
```

```
void        (*GtkSignalFunc)                (void);
```

Defines a function pointer.

GtkFunction ()

```
gboolean    (*GtkFunction)                  (gpointer data);
```

Defines a function pointer.

data : gpointer
Returns : gint

GtkDestroyNotify ()

```
void        (*GtkDestroyNotify)             (gpointer data);
```

Defines a function pointer.

data : gpointer

GtkCallbackMarshal ()

```
void        (*GtkCallbackMarshal)           (GtkObject *object,
                                             gpointer data,
                                             guint n_args,
                                             GtkArg *args);
```

Defines a function pointer.

object : GtkObject*
data : gpointer
n_args : guint
args : GtkArg*

GtkSignalMarshaller

```
typedef GSignalCMarshaller          GtkSignalMarshaller;
```

Warning

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

Defines a function pointer.

GtkTypeObject

```
typedef GTypeInstance           GtkTypeObject;
```

Warning

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

A GtkTypeObject defines the minimum structure requirements for type instances. Type instances returned from `gtk_type_new()` and initialized through a GtkObjectInitFunc need to directly inherit from this structure or at least copy its fields one by one.

GtkArg

```
typedef struct {
  GtkType type;
  gchar *name;

  /* this union only defines the required storage types for
   * the possibile values, thus there is no gint enum_data field,
   * because that would just be a mere alias for gint int_data.
   * use the GTK_VALUE_*() and GTK_RETLOC_*() macros to access
   * the discrete memebers.
   */
  union {
    /* flat values */
    gchar char_data;
    guchar uchar_data;
```

```
    gboolean bool_data;
    gint int_data;
    guint uint_data;
    glong long_data;
    gulong ulong_data;
    gfloat float_data;
    gdouble double_data;
    gchar *string_data;
    GtkObject *object_data;
    gpointer pointer_data;

    /* structured values */
    struct {
      GtkSignalFunc f;
      gpointer d;
    } signal_data;
  } d;
} GtkArg;
```

Warning

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

This is a structure that we use to pass in typed values (and names).

GTK_VALUE_CHAR()

```
#define GTK_VALUE_CHAR(a)       ((a).d.char_data)
```

Warning

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

Gets the value of a GtkArg whose GtkType is `GTK_TYPE_CHAR`.

a : a GtkArg.

GTK_VALUE_UCHAR()

```
#define GTK_VALUE_UCHAR(a)      ((a).d.uchar_data)
```

Warning

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

Gets the value of a GtkArg whose GtkType is GTK_TYPE_UCHAR.

a : a GtkArg.

GTK_VALUE_BOOL()

```
#define GTK_VALUE_BOOL(a)       ((a).d.bool_data)
```

Warning

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

Gets the value of a GtkArg whose GtkType is GTK_TYPE_BOOL.

a : a GtkArg.

GTK_VALUE_INT()

```
#define GTK_VALUE_INT(a)        ((a).d.int_data)
```

Warning

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

Gets the value of a GtkArg whose GtkType is GTK_TYPE_INT.

a : a GtkArg.

GTK_VALUE_UINT()

```
#define GTK_VALUE_UINT(a)       ((a).d.uint_data)
```

Warning

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

Gets the value of a GtkArg whose GtkType is `GTK_TYPE_UINT`.

a : a GtkArg.

GTK_VALUE_LONG()

```
#define GTK_VALUE_LONG(a)       ((a).d.long_data)
```

Warning

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

Gets the value of a GtkArg whose GtkType is `GTK_TYPE_LONG`.

a : a GtkArg.

GTK_VALUE_ULONG()

```
#define GTK_VALUE_ULONG(a)      ((a).d.ulong_data)
```

Warning

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

Gets the value of a GtkArg whose GtkType is `GTK_TYPE_ULONG`.

a : a GtkArg.

GTK_VALUE_FLOAT()

```
#define GTK_VALUE_FLOAT(a)      ((a).d.float_data)
```

Warning

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

Gets the value of a GtkArg whose GtkType is GTK_TYPE_FLOAT.

a : a GtkArg.

GTK_VALUE_DOUBLE()

```
#define GTK_VALUE_DOUBLE(a)     ((a).d.double_data)
```

Warning

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

Gets the value of a GtkArg whose GtkType is GTK_TYPE_DOUBLE.

a : a GtkArg.

GTK_VALUE_STRING()

```
#define GTK_VALUE_STRING(a)     ((a).d.string_data)
```

Warning

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

Gets the value of a GtkArg whose GtkType is GTK_TYPE_STRING.

a : a GtkArg.

GTK_VALUE_ENUM()

```
#define GTK_VALUE_ENUM(a)       ((a).d.int_data)
```

Warning

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

Gets the value of a GtkArg whose GtkType is GTK_TYPE_ENUM.

a : a GtkArg.

GTK_VALUE_FLAGS()

```
#define GTK_VALUE_FLAGS(a)      ((a).d.uint_data)
```

Warning

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

Gets the value of a GtkArg whose GtkType is GTK_TYPE_FLAGS.

a : a GtkArg.

GTK_VALUE_BOXED()

```
#define GTK_VALUE_BOXED(a)      ((a).d.pointer_data)
```

Warning

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

Gets the value of a GtkArg whose GtkType is GTK_TYPE_BOXED.

a : a GtkArg.

GTK_VALUE_POINTER()

```
#define GTK_VALUE_POINTER(a)    ((a).d.pointer_data)
```

Warning

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

Gets the value of a GtkArg whose GtkType is GTK_TYPE_POINTER.

a : a GtkArg.

GTK_VALUE_OBJECT()

```
#define GTK_VALUE_OBJECT(a)     ((a).d.object_data)
```

Warning

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

Gets the value of a GtkArg whose GtkType is GTK_TYPE_OBJECT.

a : a GtkArg.

GTK_VALUE_SIGNAL()

```
#define GTK_VALUE_SIGNAL(a)     ((a).d.signal_data)
```

Warning

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

Gets the value of a GtkArg whose GtkType is GTK_TYPE_SIGNAL.

a : a GtkArg.

GTK_RETLOC_CHAR()

```
#define GTK_RETLOC_CHAR(a)      ((gchar*)       (a).d.pointer_data)
```

Warning

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

If the GtkArg contains a pointer to the value, this macro will be a pointer to a GTK_TYPE_CHAR.

a : a GtkArg.

GTK_RETLOC_UCHAR()

```
#define GTK_RETLOC_UCHAR(a)     ((guchar*)      (a).d.pointer_data)
```

Warning

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

If the GtkArg contains a pointer to the value, this macro will be a pointer to a GTK_TYPE_UCHAR.

a : a GtkArg.

GTK_RETLOC_BOOL()

```
#define GTK_RETLOC_BOOL(a)      ((gboolean*)    (a).d.pointer_data)
```

Warning

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

If the GtkArg contains a pointer to the value, this macro will be a pointer to a GTK_TYPE_BOOL.

a : a GtkArg.

GTK_RETLOC_INT()

```
#define GTK_RETLOC_INT(a)       ((gint*)        (a).d.pointer_data)
```

Warning

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

If the GtkArg contains a pointer to the value, this macro will be a pointer to a GTK_TYPE_INT.

a : a GtkArg.

GTK_RETLOC_UINT()

```
#define GTK_RETLOC_UINT(a)      ((guint*)       (a).d.pointer_data)
```

Warning

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

If the GtkArg contains a pointer to the value, this macro will be a pointer to a GTK_TYPE_UINT.

a : a GtkArg.

GTK_RETLOC_LONG()

```
#define GTK_RETLOC_LONG(a)      ((glong*)       (a).d.pointer_data)
```

Warning

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

If the GtkArg contains a pointer to the value, this macro will be a pointer to a GTK_TYPE_LONG.

a : a GtkArg.

GTK_RETLOC_ULONG()

```
#define GTK_RETLOC_ULONG(a)     ((gulong*)      (a).d.pointer_data)
```

Warning

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

If the GtkArg contains a pointer to the value, this macro will be a pointer to a GTK_TYPE_ULONG.

a : a GtkArg.

GTK_RETLOC_FLOAT()

```
#define GTK_RETLOC_FLOAT(a)     ((gfloat*)      (a).d.pointer_data)
```

Warning

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

If the GtkArg contains a pointer to the value, this macro will be a pointer to a GTK_TYPE_FLOAT.

a : a GtkArg.

GTK_RETLOC_DOUBLE()

```
#define GTK_RETLOC_DOUBLE(a)    ((gdouble*)     (a).d.pointer_data)
```

Warning

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

If the GtkArg contains a pointer to the value, this macro will be a pointer to a GTK_TYPE_DOUBLE.

a : a GtkArg.

GTK_RETLOC_STRING()

```
#define GTK_RETLOC_STRING(a)    ((gchar**)      (a).d.pointer_data)
```

Warning

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

If the GtkArg contains a pointer to the value, this macro will be a pointer to a GTK_TYPE_STRING.

a : a GtkArg.

GTK_RETLOC_ENUM()

```
#define GTK_RETLOC_ENUM(a)      ((gint*)        (a).d.pointer_data)
```

Warning

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

If the GtkArg contains a pointer to the value, this macro will be a pointer to a GTK_TYPE_ENUM.

a : a GtkArg.

GTK_RETLOC_FLAGS()

```
#define GTK_RETLOC_FLAGS(a)     ((guint*)       (a).d.pointer_data)
```

Warning

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

If the GtkArg contains a pointer to the value, this macro will be a pointer to a GTK_TYPE_FLAGS.

a : a GtkArg.

GTK_RETLOC_BOXED()

```
#define GTK_RETLOC_BOXED(a)     ((gpointer*)    (a).d.pointer_data)
```

Warning

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

If the GtkArg contains a pointer to the value, this macro will be a pointer to a GTK_TYPE_BOXED.

a : a GtkArg.

GTK_RETLOC_POINTER()

```
#define GTK_RETLOC_POINTER(a)   ((gpointer*)    (a).d.pointer_data)
```

Warning

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

If the GtkArg contains a pointer to the value, this macro will be a pointer to a GTK_TYPE_POINTER.

a : a GtkArg.

GTK_RETLOC_OBJECT()

```
#define GTK_RETLOC_OBJECT(a)    ((GtkObject**)  (a).d.pointer_data)
```

Warning

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

If the GtkArg contains a pointer to the value, this macro will be a pointer to a GTK_TYPE_OBJECT.

a : a GtkArg.

GtkTypeInfo

```
typedef struct {
  gchar                 *type_name;
  guint                  object_size;
```

```
  guint                 class_size;
  GtkClassInitFunc      class_init_func;
  GtkObjectInitFunc     object_init_func;
  gpointer              reserved_1;
  gpointer              reserved_2;
  GtkClassInitFunc      base_class_init_func;
} GtkTypeInfo;
```

Warning

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

Holds information about the type. `gtk_type_name()` returns the name. *`object_size`* is somehow set to the number of bytes that an instance of the object will occupy. *`class_init_func`* holds the type's initialization function. *`object_init_func`* holds the initialization function for an instance of the object. *`reserved_1`* is used for GtkEnumValue to hold the enumerated values.

GtkTypeClass

```
typedef GTypeClass              GtkTypeClass;
```

Warning

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

The base structure for a GTK+ type. Every type inherits this as a base structure.

GtkEnumValue

```
typedef GEnumValue  GtkEnumValue;
```

Warning

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

A structure which contains a single enum value, and its name, and its nickname.

GtkFlagValue

```
typedef GFlagsValue GtkFlagValue;
```

Warning

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

---

gtk_type_init ()

```
void        gtk_type_init                       (GTypeDebugFlags debug_flags);
```

Warning

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

Initializes the data structures associated with GTK+ types.

`debug_flags` :

---

gtk_type_unique ()

```
GtkType     gtk_type_unique                     (GtkType parent_type,
                                                 const GtkTypeInfo *gtkinfo);
```

Warning

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

Creates a new, unique type.

`parent_type` : if zero, a fundamental type is created.
`gtkinfo` :
Returns : the new GtkType.

---

gtk_type_name()

```
#define gtk_type_name(type)             g_type_name (type)
```

Warning

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

Returns a pointer to the name of a type, or `NULL` if it has none.

type : a GtkType.
Returns : a pointer to the name of a type, or `NULL` if it has none.

---

gtk_type_from_name()

```
#define gtk_type_from_name(name)        g_type_from_name (name)
```

Warning

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

Gets the internal representation of a type, given its name.

name : the name of a GTK+ type
Returns : a GtkType.

---

gtk_type_parent()

```
#define gtk_type_parent(type)           g_type_parent (type)
```

Warning

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

Returns the parent type of a GtkType.

type : a GtkType.
Returns : the GtkType of the parent.

gtk_type_class ()

```
gpointer    gtk_type_class                  (GtkType type);
```

Returns a pointer pointing to the class of *type* or NULL if there was any trouble identifying *type*. Initializes the class if necessary.

type : a GtkType.

Returns : pointer to the class.

gtk_type_new ()

```
gpointer    gtk_type_new                    (GtkType type);
```

Warning

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

Creates a new object of a given type, and return a pointer to it. Returns NULL if you give it an invalid type. It allocates the object out of the type's memory chunk if there is a memory chunk. The object has all the proper initializers called.

type : a GtkType.

Returns : pointer to a GtkTypeObject.

gtk_type_is_a()

```
#define gtk_type_is_a(type, is_a_type)   g_type_is_a ((type), (is_a_type))
```

Warning

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

Looks in the type hierarchy to see if *type* has *is_a_type* among its ancestors. Do so with a simple lookup, not a loop.

type : a GtkType.

is_a_type : another GtkType.

Returns : TRUE if *type* is a *is_a_type*.

---

gtk_type_enum_get_values ()

```
GtkEnumValue* gtk_type_enum_get_values      (GtkType enum_type);
```

Warning

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

If *enum_type* has values, then return a pointer to all of them.

enum_type : a GtkType.

Returns : GtkEnumValue*

---

gtk_type_flags_get_values ()

```
GtkFlagValue* gtk_type_flags_get_values     (GtkType flags_type);
```

Warning

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

If *flags_type* has values, then return a pointer to all of them.

flags_type : a GtkType.

Returns : GtkFlagValue*

---

gtk_type_enum_find_value ()

```
GtkEnumValue* gtk_type_enum_find_value      (GtkType enum_type,
                                             const gchar *value_name);
```

Warning

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

Returns a pointer to one of *`enum_type`*'s GtkEnumValues's whose name (or nickname) matches *`value_name`*.

`enum_type` : a GtkType.
`value_name` : the name to look for.
Returns : GtkEnumValue*

---

gtk_type_flags_find_value ()

```
GtkFlagValue* gtk_type_flags_find_value     (GtkType flags_type,
                                             const gchar *value_name);
```

Warning

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

Returns a pointer to one of *`flag_type`*'s GtkFlagValue's whose name (or nickname) matches *`value_name`*.

`flags_type` : a GtkType.
`value_name` : the name to look for.
Returns : GtkFlagValue*

GTK+ Widgets and Objects

<< Types Object Hierarchy >>

Object Hierarchy

- GObject
  - GtkObject
    - GtkWidget
      - GtkContainer
        - GtkBin
          - GtkWindow
            - GtkDialog
              - GtkAboutDialog
              - GtkColorSelectionDialog
              - GtkFileChooserDialog
              - GtkFileSelection
              - GtkFontSelectionDialog
              - GtkInputDialog
              - GtkMessageDialog
            - GtkPlug
          - GtkAlignment
          - GtkFrame
            - GtkAspectFrame
          - GtkButton
            - GtkToggleButton
              - GtkCheckButton
                - GtkRadioButton
            - GtkColorButton
            - GtkFontButton
            - GtkOptionMenu
          - GtkItem
            - GtkMenuItem
              - GtkCheckMenuItem
                - GtkRadioMenuItem
              - GtkImageMenuItem
              - GtkSeparatorMenuItem
              - GtkTearoffMenuItem
            - GtkListItem

- GtkTreeItem
- GtkComboBox
  - GtkComboBoxEntry
- GtkEventBox
- GtkExpander
- GtkHandleBox
- GtkToolItem
  - GtkToolButton
    - GtkMenuToolButton
    - GtkToggleToolButton
      - GtkRadioToolButton
  - GtkSeparatorToolItem
- GtkScrolledWindow
- GtkViewport

GtkBox

- GtkButtonBox
  - GtkHButtonBox
  - GtkVButtonBox
- GtkVBox
  - GtkColorSelection
  - GtkFileChooserWidget
  - GtkFontSelection
  - GtkGammaCurve
- GtkHBox
  - GtkCombo
  - GtkFileChooserButton
  - GtkStatusbar

GtkCList

- GtkCTree

GtkFixed

GtkPaned

- GtkHPaned
- GtkVPaned

GtkIconView

GtkLayout

GtkList

GtkMenuShell

- GtkMenuBar

```
                GtkMenu
            GtkNotebook
            GtkSocket
            GtkTable
            GtkTextView
            GtkToolbar
            GtkTree
            GtkTreeView
        GtkMisc
            GtkLabel
                GtkAccelLabel
                GtkTipsQuery
            GtkArrow
            GtkImage
            GtkPixmap
        GtkCalendar
        GtkCellView
        GtkDrawingArea
            GtkCurve
        GtkEntry
            GtkSpinButton
        GtkRuler
            GtkHRuler
            GtkVRuler
        GtkRange
            GtkScale
                GtkHScale
                GtkVScale
            GtkScrollbar
                GtkHScrollbar
                GtkVScrollbar
        GtkSeparator
            GtkHSeparator
            GtkVSeparator
        GtkInvisible
        GtkOldEditable
            GtkText
        GtkPreview
```

- GtkProgress
  - GtkProgressBar
- GtkAdjustment
- GtkCellRenderer
  - GtkCellRendererText
    - GtkCellRendererCombo
  - GtkCellRendererPixbuf
  - GtkCellRendererToggle
  - GtkCellRendererProgress
- GtkFileFilter
- GtkItemFactory
- GtkTooltips
- GtkTreeViewColumn

GtkAccelGroup

GtkAccelMap

AtkObject

- GtkAccessible

GtkAction

- GtkToggleAction
  - GtkRadioAction

GtkActionGroup

GtkEntryCompletion

GtkIconFactory

GtkIconTheme

GtkIMContext

- GtkIMContextSimple
- GtkIMMulticontext

GtkListStore

GtkRcStyle

GtkSettings

GtkSizeGroup

GtkStyle

GtkTextBuffer

GtkTextChildAnchor

GtkTextMark

GtkTextTag

GtkTextTagTable

GtkTreeModelFilter

```
        GtkTreeModelSort
        GtkTreeSelection
        GtkTreeStore
        GtkUIManager
        GtkWindowGroup
    GInterface
        GtkCellLayout
        GtkCellEditable
        GtkEditable
        GtkFileChooser
        GtkTreeModel
        GtkTreeDragSource
        GtkTreeDragDest
        GtkTreeSortable
```

<< Part III. GTK+ Widgets and Objects **Widget Gallery >>**

Widget Gallery















<< Object Hierarchy **Windows >>**

Windows

GtkDialog - Create popup windows
GtkInvisible - A widget which is not displayed
GtkMessageDialog - A convenient message window
GtkWindow - Toplevel which can contain other widgets
GtkWindowGroup - Limit the effect of grabs
GtkAboutDialog - Display information about an application

GtkDialog

GtkDialog — Create popup windows

Synopsis

```
#include <gtk/gtk.h>

            GtkDialog;
enum        GtkDialogFlags;
enum        GtkResponseType;
GtkWidget*  gtk_dialog_new                  (void);
GtkWidget*  gtk_dialog_new_with_buttons     (const gchar *title,
                                             GtkWindow *parent,
                                             GtkDialogFlags flags,
                                             const gchar *first_button_text,
                                             ...);
gint        gtk_dialog_run                  (GtkDialog *dialog);
void        gtk_dialog_response             (GtkDialog *dialog,
                                             gint response_id);
GtkWidget*  gtk_dialog_add_button           (GtkDialog *dialog,
                                             const gchar *button_text,
                                             gint response_id);
void        gtk_dialog_add_buttons          (GtkDialog *dialog,
                                             const gchar *first_button_text,
                                             ...);
void        gtk_dialog_add_action_widget    (GtkDialog *dialog,
                                             GtkWidget *child,
                                             gint response_id);
gboolean    gtk_dialog_get_has_separator    (GtkDialog *dialog);
void        gtk_dialog_set_default_response (GtkDialog *dialog,
                                             gint response_id);
void        gtk_dialog_set_has_separator    (GtkDialog *dialog,
                                             gboolean setting);
void        gtk_dialog_set_response_sensitive
                                            (GtkDialog *dialog,
                                             gint response_id,
                                             gboolean setting);

gboolean    gtk_alternative_dialog_button_order
                                            (GdkScreen *screen);
```

```
void        gtk_dialog_set_alternative_button_order
                                            (GtkDialog *dialog,
                                             gint first_response_id,
                                             ...);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkWidget
               +----GtkContainer
                     +----GtkBin
                           +----GtkWindow
                                 +----GtkDialog
                                       +----GtkAboutDialog
                                       +----GtkColorSelectionDialog
                                       +----GtkFileChooserDialog
                                       +----GtkFileSelection
                                       +----GtkFontSelectionDialog
                                       +----GtkInputDialog
                                       +----GtkMessageDialog
```

Implemented Interfaces

GtkDialog implements AtkImplementorIface.

Properties

```
  "has-separator"        gboolean             : Read / Write
```

Style Properties

```
  "action-area-border"   gint                 : Read
  "button-spacing"       gint                 : Read
  "content-area-border"  gint                 : Read
```

Signal Prototypes

```
"close"     void        user_function      (GtkDialog *dialog,
                                            gpointer user_data);
"response"  void        user_function      (GtkDialog *dialog,
                                            gint arg1,
                                            gpointer user_data);
```

Description

Dialog boxes are a convenient way to prompt the user for a small amount of input, e.g. to display a message, ask a question, or anything else that does not require extensive effort on the user's part.

GTK+ treats a dialog as a window split vertically. The top section is a GtkVBox, and is where widgets such as a GtkLabel or a GtkEntry should be packed. The bottom area is known as the *action_area*. This is generally used for packing buttons into the dialog which may perform functions such as cancel, ok, or apply. The two areas are separated by a GtkHSeparator.

GtkDialog boxes are created with a call to `gtk_dialog_new()` or `gtk_dialog_new_with_buttons()`. `gtk_dialog_new_with_buttons()` is recommended; it allows you to set the dialog title, some convenient flags, and add simple buttons.

If 'dialog' is a newly created dialog, the two primary areas of the window can be accessed as `GTK_DIALOG(dialog)->vbox` and `GTK_DIALOG(dialog)->action_area`, as can be seen from the example, below.

A 'modal' dialog (that is, one which freezes the rest of the application from user input), can be created by calling `gtk_window_set_modal()` on the dialog. Use the `GTK_WINDOW()` macro to cast the widget returned from `gtk_dialog_new()` into a GtkWindow. When using `gtk_dialog_new_with_buttons()` you can also pass the GTK_DIALOG_MODAL flag to make a dialog modal.

If you add buttons to GtkDialog using `gtk_dialog_new_with_buttons()`, `gtk_dialog_add_button()`, `gtk_dialog_add_buttons()`, or `gtk_dialog_add_action_widget()`, clicking the button will emit a signal called "response" with a response ID that you specified. GTK+ will never assign a meaning to positive response IDs; these are entirely user-defined. But for convenience, you can use the response IDs in the GtkResponseType enumeration (these all have values less than zero). If a dialog receives a delete event, the "response" signal will be emitted with a response ID of GTK_RESPONSE_DELETE_EVENT.

If you want to block waiting for a dialog to return before returning control flow to your code, you can call `gtk_dialog_run()`. This function enters a recursive main loop and waits for the user to respond to the dialog, returning the response ID corresponding to the button the user clicked.

For the simple dialog in the following example, in reality you'd probably use GtkMessageDialog to save yourself some effort. But you'd need to create the dialog contents manually if you had more than a simple message in the dialog.

Example 1. Simple GtkDialog usage.

```
/* Function to open a dialog box displaying the message provided. */
```

```
void quick_message (gchar *message) {

   GtkWidget *dialog, *label;

   /* Create the widgets */

   dialog = gtk_dialog_new_with_buttons ("Message",
                                         main_application_window,
                                         GTK_DIALOG_DESTROY_WITH_PARENT,
                                         GTK_STOCK_OK,
                                         GTK_RESPONSE_NONE,
                                         NULL);
   label = gtk_label_new (message);

   /* Ensure that the dialog box is destroyed when the user responds. */

   g_signal_connect_swapped (dialog,
                             "response",
                             G_CALLBACK (gtk_widget_destroy),
                             dialog);

   /* Add the label, and show everything we've added to the dialog. */

   gtk_container_add (GTK_CONTAINER (GTK_DIALOG(dialog)->vbox),
                      label);
   gtk_widget_show_all (dialog);
}
```

Details

GtkDialog

```
typedef struct {
  GtkWidget *vbox;
  GtkWidget *action_area;
} GtkDialog;
```

vbox is a GtkVBox - the main part of the dialog box.

action_area is a GtkHButtonBox packed below the dividing GtkHSeparator in the dialog. It is treated exactly the same as any other GtkHButtonBox.

---

enum GtkDialogFlags

```
typedef enum
```

```
{
  GTK_DIALOG_MODAL              = 1 << 0, /* call gtk_window_set_modal (win, TRUE)
*/
  GTK_DIALOG_DESTROY_WITH_PARENT = 1 << 1, /* call gtk_window_set_destroy_with_parent
() */
  GTK_DIALOG_NO_SEPARATOR       = 1 << 2  /* no separator bar above buttons */
} GtkDialogFlags;
```

Flags used to influence dialog construction.

GTK_DIALOG_MODAL	Make the constructed dialog modal, see gtk_widget_set_modal().
GTK_DIALOG_DESTROY_WITH_PARENT	Destroy the dialog when its parent is destroyed, see gtk_window_set_destroy_with_parent().
GTK_DIALOG_NO_SEPARATOR	Don't put a separator between the action area and the dialog content.

---

enum GtkResponseType

```
typedef enum
{
  /* GTK returns this if a response widget has no response_id,
   * or if the dialog gets programmatically hidden or destroyed.
   */
  GTK_RESPONSE_NONE = -1,

  /* GTK won't return these unless you pass them in
   * as the response for an action widget. They are
   * for your convenience.
   */
  GTK_RESPONSE_REJECT = -2,
  GTK_RESPONSE_ACCEPT = -3,

  /* If the dialog is deleted. */
  GTK_RESPONSE_DELETE_EVENT = -4,

  /* These are returned from GTK dialogs, and you can also use them
   * yourself if you like.
   */
  GTK_RESPONSE_OK     = -5,
  GTK_RESPONSE_CANCEL = -6,
  GTK_RESPONSE_CLOSE  = -7,
  GTK_RESPONSE_YES    = -8,
  GTK_RESPONSE_NO     = -9,
  GTK_RESPONSE_APPLY  = -10,
  GTK_RESPONSE_HELP   = -11
} GtkResponseType;
```

Predefined values for use as response ids in gtk_dialog_add_button(). All predefined values are negative, GTK+ leaves

positive values for application-defined response ids.

GTK_RESPONSE_NONE	Returned if an action widget has no response id, or if the dialog gets programmatically hidden or destroyed.
GTK_RESPONSE_REJECT	Generic response id, not used by GTK+ dialogs.
GTK_RESPONSE_ACCEPT	Generic response id, not used by GTK+ dialogs.
GTK_RESPONSE_DELETE_EVENT	Returned if the dialog is deleted.
GTK_RESPONSE_OK	Returned by OK buttons in GTK+ dialogs.
GTK_RESPONSE_CANCEL	Returned by Cancel buttons in GTK+ dialogs.
GTK_RESPONSE_CLOSE	Returned by Close buttons in GTK+ dialogs.
GTK_RESPONSE_YES	Returned by Yes buttons in GTK+ dialogs.
GTK_RESPONSE_NO	Returned by No buttons in GTK+ dialogs.
GTK_RESPONSE_APPLY	Returned by Apply buttons in GTK+ dialogs.
GTK_RESPONSE_HELP	Returned by Help buttons in GTK+ dialogs.

---

gtk_dialog_new ()

```
GtkWidget*  gtk_dialog_new                  (void);
```

Creates a new dialog box. Widgets should not be packed into this GtkWindow directly, but into the *vbox* and *action_area*, as described above.

Returns : a new GtkDialog.

---

gtk_dialog_new_with_buttons ()

```
GtkWidget*  gtk_dialog_new_with_buttons     (const gchar *title,
                                             GtkWindow *parent,
                                             GtkDialogFlags flags,
                                             const gchar *first_button_text,
                                             ...);
```

Creates a new GtkDialog with title *title* (or NULL for the default title; see gtk_window_set_title()) and transient parent *parent* (or NULL for none; see gtk_window_set_transient_for()). The *flags* argument can be used to make the dialog modal (GTK_DIALOG_MODAL) and/or to have it destroyed along with its transient parent (GTK_DIALOG_DESTROY_WITH_PARENT). After *flags*, button text/response ID pairs should be listed, with a NULL pointer ending the list. Button text can be either a stock ID such as GTK_STOCK_OK, or some arbitrary text. A response ID can be any positive number, or one of the values in the GtkResponseType enumeration. If the user clicks one of these dialog buttons, GtkDialog will emit the "response" signal with the corresponding response ID. If a GtkDialog receives the "delete_event" signal, it will emit "response" with a response ID of GTK_RESPONSE_DELETE_EVENT. However, destroying a dialog does not emit the "response" signal; so be careful relying on "response" when using the GTK_DIALOG_DESTROY_WITH_PARENT flag.

Buttons are from left to right, so the first button in the list will be the leftmost button in the dialog.

Here's a simple example:

```
 GtkWidget *dialog = gtk_dialog_new_with_buttons ("My dialog",
                                                  main_app_window,
                                                  GTK_DIALOG_MODAL |
GTK_DIALOG_DESTROY_WITH_PARENT,
                                                  GTK_STOCK_OK,
                                                  GTK_RESPONSE_ACCEPT,
                                                  GTK_STOCK_CANCEL,
                                                  GTK_RESPONSE_REJECT,
                                                  NULL);
```

title :	Title of the dialog, or NULL
parent :	Transient parent of the dialog, or NULL
flags :	from GtkDialogFlags
first_button_text :	stock ID or text to go in first button, or NULL
... :	response ID for first button, then additional buttons, ending with NULL
Returns :	a new GtkDialog

---

gtk_dialog_run ()

```
gint        gtk_dialog_run                  (GtkDialog *dialog);
```

Blocks in a recursive main loop until the *dialog* either emits the response signal, or is destroyed. If the dialog is destroyed during the call to gtk_dialog_run(), gtk_dialog_returns GTK_RESPONSE_NONE. Otherwise, it returns the response ID from the "response" signal emission. Before entering the recursive main loop, gtk_dialog_run() calls gtk_widget_show() on the dialog for you. Note that you still need to show any children of the dialog yourself.

During gtk_dialog_run(), the default behavior of "delete_event" is disabled; if the dialog receives "delete_event", it will not be destroyed as windows usually are, and gtk_dialog_run() will return GTK_RESPONSE_DELETE_EVENT. Also, during gtk_dialog_run() the dialog will be modal. You can force gtk_dialog_run() to return at any time by calling gtk_dialog_response() to emit the "response" signal. Destroying the dialog during gtk_dialog_run() is a very bad idea, because your post-run code won't know whether the dialog was destroyed or not.

After gtk_dialog_run() returns, you are responsible for hiding or destroying the dialog if you wish to do so.

Typical usage of this function might be:

```
  gint result = gtk_dialog_run (GTK_DIALOG (dialog));
  switch (result)
    {
      case GTK_RESPONSE_ACCEPT:
```

```
        do_application_specific_something ();
        break;
      default:
        do_nothing_since_dialog_was_cancelled ();
        break;
    }
  gtk_widget_destroy (dialog);
```

dialog : a GtkDialog

Returns : response ID

---

gtk_dialog_response ()

```
void        gtk_dialog_response             (GtkDialog *dialog,
                                             gint response_id);
```

Emits the "response" signal with the given response ID. Used to indicate that the user has responded to the dialog in some way; typically either you or `gtk_dialog_run()` will be monitoring the "response" signal and take appropriate action.

dialog : a GtkDialog

response_id : response ID

---

gtk_dialog_add_button ()

```
GtkWidget*  gtk_dialog_add_button           (GtkDialog *dialog,
                                             const gchar *button_text,
                                             gint response_id);
```

Adds a button with the given text (or a stock button, if *button_text* is a stock ID) and sets things up so that clicking the button will emit the "response" signal with the given *response_id*. The button is appended to the end of the dialog's action area. The button widget is returned, but usually you don't need it.

dialog : a GtkDialog

button_text : text of button, or stock ID

response_id : response ID for the button

Returns : the button widget that was added

---

gtk_dialog_add_buttons ()

```
void        gtk_dialog_add_buttons          (GtkDialog *dialog,
                                             const gchar *first_button_text,
                                             ...);
```

Adds more buttons, same as calling `gtk_dialog_add_button()` repeatedly. The variable argument list should be `NULL`-terminated as with `gtk_dialog_new_with_buttons()`. Each button must have both text and response ID.

`dialog` :	a GtkDialog
`first_button_text` :	button text or stock ID
`...` :	response ID for first button, then more text-response_id pairs

---

gtk_dialog_add_action_widget ()

```
void        gtk_dialog_add_action_widget    (GtkDialog *dialog,
                                             GtkWidget *child,
                                             gint response_id);
```

Adds an activatable widget to the action area of a GtkDialog, connecting a signal handler that will emit the "response" signal on the dialog when the widget is activated. The widget is appended to the end of the dialog's action area. If you want to add a non-activatable widget, simply pack it into the `action_area` field of the GtkDialog struct.

`dialog` :	a GtkDialog
`child` :	an activatable widget
`response_id` :	response ID for *`child`*

---

gtk_dialog_get_has_separator ()

```
gboolean    gtk_dialog_get_has_separator    (GtkDialog *dialog);
```

Accessor for whether the dialog has a separator.

`dialog` :	a GtkDialog
Returns :	`TRUE` if the dialog has a separator

---

gtk_dialog_set_default_response ()

```
void        gtk_dialog_set_default_response (GtkDialog *dialog,
                                             gint response_id);
```

Sets the last widget in the dialog's action area with the given *response_id* as the default widget for the dialog. Pressing "Enter" normally activates the default widget.

dialog : a GtkDialog
response_id : a response ID

---

gtk_dialog_set_has_separator ()

```
void        gtk_dialog_set_has_separator    (GtkDialog *dialog,
                                             gboolean setting);
```

Sets whether the dialog has a separator above the buttons. TRUE by default.

dialog : a GtkDialog
setting : TRUE to have a separator

---

gtk_dialog_set_response_sensitive ()

```
void        gtk_dialog_set_response_sensitive
                                            (GtkDialog *dialog,
                                             gint response_id,
                                             gboolean setting);
```

Calls `gtk_widget_set_sensitive (widget, setting)` for each widget in the dialog's action area with the given *response_id*. A convenient way to sensitize/desensitize dialog buttons.

dialog : a GtkDialog
response_id : a response ID
setting : TRUE for sensitive

---

gtk_alternative_dialog_button_order ()

```
gboolean    gtk_alternative_dialog_button_order
                                            (GdkScreen *screen);
```

Returns TRUE if dialogs are expected to use an alternative button order on the screen *screen*. See `gtk_dialog_set_alternative_button_order()` for more details about alternative button order.

If you need to use this function, you should probably connect to the ::notify:gtk-alternative-button-order signal on the GtkSettings object associated to *screen*, in order to be notified if the button order setting changes.

screen : a GdkScreen, or NULL to use the default screen

Returns : Whether the alternative button order should be used

Since 2.6

---

gtk_dialog_set_alternative_button_order ()

```
void        gtk_dialog_set_alternative_button_order
                                            (GtkDialog *dialog,
                                             gint first_response_id,
                                             ...);
```

Sets an alternative button order. If the gtk-alternative-button-order setting is set to TRUE, the dialog buttons are reordered according to the order of the response ids passed to this function.

By default, GTK+ dialogs use the button order advocated by the Gnome Human Interface Guidelines with the affirmative button at the far right, and the cancel button left of it. But the builtin GTK+ dialogs and GtkMessageDialogs do provide an alternative button order, which is more suitable on some platforms, e.g. Windows.

Use this function after adding all the buttons to your dialog, as the following example shows:

```
cancel_button = gtk_dialog_add_button (GTK_DIALOG (dialog),
                                       GTK_STOCK_CANCEL,
                                       GTK_RESPONSE_CANCEL);

ok_button = gtk_dialog_add_button (GTK_DIALOG (dialog),
                                   GTK_STOCK_OK,
                                   GTK_RESPONSE_OK);

gtk_widget_grab_default (ok_button);

help_button = gtk_dialog_add_button (GTK_DIALOG (dialog),
                                     GTK_STOCK_HELP,
                                     GTK_RESPONSE_HELP);

gtk_dialog_set_alternative_button_order (GTK_DIALOG (dialog),
                                         GTK_RESPONSE_OK,
                                         GTK_RESPONSE_CANCEL,
                                         GTK_RESPONSE_HELP,
                                         -1);
```

dialog :	a GtkDialog
first_response_id :	a response id used by one *dialog*'s buttons
... :	a list of more response ids of *dialog*'s buttons, terminated by -1

Since 2.6

Properties

The "has-separator" property

```
"has-separator"        gboolean              : Read / Write
```

The dialog has a separator bar above its buttons.

Default value: TRUE

Style Properties

The "action-area-border" style property

```
"action-area-border"   gint                  : Read
```

Width of border around the button area at the bottom of the dialog.

Allowed values: >= 0

Default value: 5

The "button-spacing" style property

```
"button-spacing"       gint                  : Read
```

Spacing between buttons.

Allowed values: >= 0

Default value: 10

The "content-area-border" style property

```
  "content-area-border"  gint                  : Read
```

Width of border around the main dialog area.

Allowed values: >= 0

Default value: 2

Signals

The "close" signal

```
void        user_function                  (GtkDialog *dialog,
                                            gpointer user_data);
```

dialog : the object which received the signal.
user_data : user data set when the signal handler was connected.

The "response" signal

```
void        user_function                  (GtkDialog *dialog,
                                            gint arg1,
                                            gpointer user_data);
```

Emitted when an action widget is clicked, the dialog receives a delete event, or the application programmer calls `gtk_dialog_response()`. On a delete event, the response ID is GTK_RESPONSE_NONE. Otherwise, it depends on which action widget was clicked.

dialog : the object which received the signal.
arg1 : the response ID
user_data : user data set when the signal handler was connected.

See Also

GtkVBox Pack widgets vertically.
GtkWindow Alter the properties of your dialog box.
GtkButton Add them to the *action_area* to get a response from the user.

<< Windows **GtkInvisible >>**

GtkInvisible

GtkInvisible — A widget which is not displayed

Synopsis

```
#include <gtk/gtk.h>

            GtkInvisible;
GtkWidget*  gtk_invisible_new               (void);
GtkWidget*  gtk_invisible_new_for_screen    (GdkScreen *screen);
void        gtk_invisible_set_screen        (GtkInvisible *invisible,
                                             GdkScreen *screen);
GdkScreen*  gtk_invisible_get_screen        (GtkInvisible *invisible);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkWidget
               +----GtkInvisible
```

Implemented Interfaces

GtkInvisible implements AtkImplementorIface.

Properties

```
  "screen"                GdkScreen             : Read / Write
```

Description

The GtkInvisible widget is used internally in GTK+, and is probably not very useful for application developers.

It is used for reliable pointer grabs and selection handling in the code for drag-and-drop.

Details

GtkInvisible

```
typedef struct _GtkInvisible GtkInvisible;
```

The GtkInvisible-struct struct contains no public fields.

---

gtk_invisible_new ()

```
GtkWidget*  gtk_invisible_new               (void);
```

Creates a new GtkInvisible.

Returns : a new GtkInvisible.

---

gtk_invisible_new_for_screen ()

```
GtkWidget*  gtk_invisible_new_for_screen    (GdkScreen *screen);
```

Creates a new GtkInvisible object for a specified screen

screen : a GdkScreen which identifies on which the new GtkInvisible will be created.

Returns : a newly created GtkInvisible object

Since 2.2

---

gtk_invisible_set_screen ()

```
void        gtk_invisible_set_screen        (GtkInvisible *invisible,
                                             GdkScreen *screen);
```

Sets the GdkScreen where the GtkInvisible object will be displayed.

invisible : a GtkInvisible.

screen : a GdkScreen.

Since 2.2

---

gtk_invisible_get_screen ()

```
GdkScreen*  gtk_invisible_get_screen        (GtkInvisible *invisible);
```

Returns the GdkScreen object associated with *invisible*

invisible : a GtkInvisible.

Returns : the associated GdkScreen.

Since 2.2

Properties

The "screen" property

"screen"	GdkScreen	: Read / Write

The screen where this window will be displayed.

<< GtkDialog **GtkMessageDialog >>**

GtkMessageDialog

GtkMessageDialog — A convenient message window

Synopsis

```
#include <gtk/gtk.h>

            GtkMessageDialog;
enum        GtkMessageType;
enum        GtkButtonsType;
GtkWidget*  gtk_message_dialog_new          (GtkWindow *parent,
                                             GtkDialogFlags flags,
                                             GtkMessageType type,
                                             GtkButtonsType buttons,
                                             const gchar *message_format,
                                             ...);
GtkWidget*  gtk_message_dialog_new_with_markup
                                            (GtkWindow *parent,
                                             GtkDialogFlags flags,
                                             GtkMessageType type,
                                             GtkButtonsType buttons,
                                             const gchar *message_format,
                                             ...);
void        gtk_message_dialog_set_markup   (GtkMessageDialog *message_dialog,
                                             const gchar *str);
void        gtk_message_dialog_format_secondary_text
                                            (GtkMessageDialog *message_dialog,
                                             const gchar *message_format,
                                             ...);
void        gtk_message_dialog_format_secondary_markup
                                            (GtkMessageDialog *message_dialog,
                                             const gchar *message_format,
                                             ...);
```

Object Hierarchy

```
GObject
 +----GtkObject
       +----GtkWidget
             +----GtkContainer
                   +----GtkBin
                         +----GtkWindow
                               +----GtkDialog
                                     +----GtkMessageDialog
```

Implemented Interfaces

GtkMessageDialog implements AtkImplementorIface.

Properties

```
"buttons"             GtkButtonsType        : Write / Construct Only
"message-type"        GtkMessageType        : Read / Write / Construct
```

Style Properties

```
"message-border"      gint                  : Read
"use-separator"       gboolean              : Read
```

Description

GtkMessageDialog presents a dialog with an image representing the type of message (Error, Question, etc.) alongside some message text. It's simply a convenience widget; you could construct the equivalent of GtkMessageDialog from GtkDialog without too much effort, but GtkMessageDialog saves typing.

The easiest way to do a modal message dialog is to use `gtk_dialog_run()`, though you can also pass in the `GTK_DIALOG_MODAL` flag, `gtk_dialog_run()` automatically makes the dialog modal and waits for the user to respond to it. `gtk_dialog_run()` returns when any dialog button is clicked.

Example 2. A modal dialog.

```
dialog = gtk_message_dialog_new (main_application_window,
                                 GTK_DIALOG_DESTROY_WITH_PARENT,
                                 GTK_MESSAGE_ERROR,
                                 GTK_BUTTONS_CLOSE,
                                 "Error loading file '%s': %s",
                                 filename, g_strerror (errno));
gtk_dialog_run (GTK_DIALOG (dialog));
gtk_widget_destroy (dialog);
```

You might do a non-modal GtkMessageDialog as follows:

Example 3. A non-modal dialog.

```
dialog = gtk_message_dialog_new (main_application_window,
                                 GTK_DIALOG_DESTROY_WITH_PARENT,
                                 GTK_MESSAGE_ERROR,
                                 GTK_BUTTONS_CLOSE,
                                 "Error loading file '%s': %s",
                                 filename, g_strerror (errno));

/* Destroy the dialog when the user responds to it (e.g. clicks a button) */
g_signal_connect_swapped (dialog, "response",
                          G_CALLBACK (gtk_widget_destroy),
                          dialog);
```

Details

GtkMessageDialog

```
typedef struct _GtkMessageDialog GtkMessageDialog;
```

---

enum GtkMessageType

```
typedef enum
{
  GTK_MESSAGE_INFO,
  GTK_MESSAGE_WARNING,
```

```
  GTK_MESSAGE_QUESTION,
  GTK_MESSAGE_ERROR
} GtkMessageType;
```

The type of message being displayed in the dialog.

GTK_MESSAGE_INFO	Informational message
GTK_MESSAGE_WARNING	Nonfatal warning message
GTK_MESSAGE_QUESTION	Question requiring a choice
GTK_MESSAGE_ERROR	Fatal error message

---

enum GtkButtonsType

```
typedef enum
{
  GTK_BUTTONS_NONE,
  GTK_BUTTONS_OK,
  GTK_BUTTONS_CLOSE,
  GTK_BUTTONS_CANCEL,
  GTK_BUTTONS_YES_NO,
  GTK_BUTTONS_OK_CANCEL
} GtkButtonsType;
```

Prebuilt sets of buttons for the dialog. If none of these choices are appropriate, simply use GTK_BUTTONS_NONE then call gtk_dialog_add_buttons().

GTK_BUTTONS_NONE	no buttons at all
GTK_BUTTONS_OK	an OK button
GTK_BUTTONS_CLOSE	a Close button
GTK_BUTTONS_CANCEL	a Cancel button
GTK_BUTTONS_YES_NO	Yes and No buttons
GTK_BUTTONS_OK_CANCEL	OK and Cancel buttons

---

gtk_message_dialog_new ()

```
GtkWidget*  gtk_message_dialog_new          (GtkWindow *parent,
                                             GtkDialogFlags flags,
                                             GtkMessageType type,
```

```
                                                GtkButtonsType buttons,
                                                const gchar *message_format,
                                                ...);
```

Creates a new message dialog, which is a simple dialog with an icon indicating the dialog type (error, warning, etc.) and some text the user may want to see. When the user clicks a button a "response" signal is emitted with response IDs from GtkResponseType. See GtkDialog for more details.

parent :	transient parent, or NULL for none
flags :	flags
type :	type of message
buttons :	set of buttons to use
message_format :	printf()-style format string, or NULL
... :	arguments for *message_format*
Returns :	a new GtkMessageDialog

---

gtk_message_dialog_new_with_markup ()

```
GtkWidget*  gtk_message_dialog_new_with_markup
                                               (GtkWindow *parent,
                                                GtkDialogFlags flags,
                                                GtkMessageType type,
                                                GtkButtonsType buttons,
                                                const gchar *message_format,
                                                ...);
```

Creates a new message dialog, which is a simple dialog with an icon indicating the dialog type (error, warning, etc.) and some text which is marked up with the Pango text markup language. When the user clicks a button a "response" signal is emitted with response IDs from GtkResponseType. See GtkDialog for more details.

Special XML characters in the printf() arguments passed to this function will automatically be escaped as necessary. (See g_markup_printf_escaped() for how this is implemented.) Usually this is what you want, but if you have an existing Pango markup string that you want to use literally as the label, then you need to use gtk_message_dialog_set_markup() instead, since you can't pass the markup string either as the format (it might contain '%' characters) or as a string argument.

```
 GtkWidget *dialog;
 dialog = gtk_message_dialog_new (main_application_window,
                                  GTK_DIALOG_DESTROY_WITH_PARENT,
                                  GTK_MESSAGE_ERROR,
```

```
                                    GTK_BUTTON_CLOSE,
                                    NULL);
gtk_message_dialog_set_markup (GTK_MESSAGE_DIALOG (dialog),
                               markup);
```

parent :	transient parent, or NULL for none
flags :	flags
type :	type of message
buttons :	set of buttons to use
message_format :	printf()-style format string, or NULL
... :	arguments for *message_format*
Returns :	a new GtkMessageDialog

Since 2.4

---

gtk_message_dialog_set_markup ()

```
void        gtk_message_dialog_set_markup   (GtkMessageDialog *message_dialog,
                                             const gchar *str);
```

Sets the text of the message dialog to be *str*, which is marked up with the Pango text markup language.

message_dialog :	a GtkMessageDialog
str :	markup string (see Pango markup format)

Since 2.4

---

gtk_message_dialog_format_secondary_text ()

```
void        gtk_message_dialog_format_secondary_text
                                            (GtkMessageDialog *message_dialog,
                                             const gchar *message_format,
                                             ...);
```

Sets the secondary text of the message dialog to be *message_format* (with printf()-style).

Note that setting a secondary text makes the primary text become bold, unless you have provided explicit markup.

message_dialog : a GtkMessageDialog
message_format : printf()-style format string, or NULL
... : arguments for *message_format*

Since 2.6

---

gtk_message_dialog_format_secondary_markup ()

```
void        gtk_message_dialog_format_secondary_markup
                                            (GtkMessageDialog *message_dialog,
                                             const gchar *message_format,
                                             ...);
```

Sets the secondary text of the message dialog to be *message_format* (with printf()-style), which is marked up with the Pango text markup language.

Note that setting a secondary text makes the primary text become bold, unless you have provided explicit markup.

message_dialog : a GtkMessageDialog
message_format : printf()-style markup string (see Pango markup format), or NULL
... : arguments for *message_format*

Since 2.6

Properties

The "buttons" property

```
  "buttons"             GtkButtonsType        : Write / Construct Only
```

The buttons shown in the message dialog.

Default value: GTK_BUTTONS_NONE

The "message-type" property

```
"message-type"          GtkMessageType          : Read / Write / Construct
```

The type of message.

Default value: GTK_MESSAGE_INFO

Style Properties

The "message-border" style property

```
"message-border"        gint                    : Read
```

Width of border around the label and image in the message dialog.

Allowed values: >= 0

Default value: 12

The "use-separator" style property

```
"use-separator"         gboolean                : Read
```

Whether to put a separator between the message dialog's text and the buttons.

Default value: FALSE

See Also

GtkDialog

<< GtkInvisible **GtkWindow >>**

GtkWindow

GtkWindow — Toplevel which can contain other widgets

Synopsis

```
#include <gtk/gtk.h>

            GtkWindow;
GtkWidget*  gtk_window_new                  (GtkWindowType type);
void        gtk_window_set_title            (GtkWindow *window,
                                             const gchar *title);
void        gtk_window_set_wmclass          (GtkWindow *window,
                                             const gchar *wmclass_name,
                                             const gchar *wmclass_class);
void        gtk_window_set_policy           (GtkWindow *window,
                                             gint allow_shrink,
                                             gint allow_grow,
                                             gint auto_shrink);
void        gtk_window_set_resizable        (GtkWindow *window,
                                             gboolean resizable);
gboolean    gtk_window_get_resizable        (GtkWindow *window);
void        gtk_window_add_accel_group      (GtkWindow *window,
                                             GtkAccelGroup *accel_group);
void        gtk_window_remove_accel_group   (GtkWindow *window,
                                             GtkAccelGroup *accel_group);
#define     gtk_window_position
gboolean    gtk_window_activate_focus       (GtkWindow *window);
gboolean    gtk_window_activate_default     (GtkWindow *window);
void        gtk_window_set_modal            (GtkWindow *window,
                                             gboolean modal);
void        gtk_window_set_default_size     (GtkWindow *window,
                                             gint width,
                                             gint height);
void        gtk_window_set_geometry_hints   (GtkWindow *window,
                                             GtkWidget *geometry_widget,
```

```
                                                GdkGeometry *geometry,
                                                GdkWindowHints geom_mask);
void        gtk_window_set_gravity          (GtkWindow *window,
                                                GdkGravity gravity);
GdkGravity  gtk_window_get_gravity          (GtkWindow *window);
void        gtk_window_set_position         (GtkWindow *window,
                                                GtkWindowPosition position);
void        gtk_window_set_transient_for    (GtkWindow *window,
                                                GtkWindow *parent);
void        gtk_window_set_destroy_with_parent
                                            (GtkWindow *window,
                                                gboolean setting);
void        gtk_window_set_screen           (GtkWindow *window,
                                                GdkScreen *screen);
GdkScreen*  gtk_window_get_screen           (GtkWindow *window);
gboolean    gtk_window_is_active            (GtkWindow *window);
gboolean    gtk_window_has_toplevel_focus   (GtkWindow *window);
GList*      gtk_window_list_toplevels       (void);
void        gtk_window_add_mnemonic         (GtkWindow *window,
                                                guint keyval,
                                                GtkWidget *target);
void        gtk_window_remove_mnemonic      (GtkWindow *window,
                                                guint keyval,
                                                GtkWidget *target);
gboolean    gtk_window_mnemonic_activate    (GtkWindow *window,
                                                guint keyval,
                                                GdkModifierType modifier);
gboolean    gtk_window_activate_key         (GtkWindow *window,
                                                GdkEventKey *event);
gboolean    gtk_window_propagate_key_event  (GtkWindow *window,
                                                GdkEventKey *event);
GtkWidget*  gtk_window_get_focus            (GtkWindow *window);
void        gtk_window_set_focus            (GtkWindow *window,
                                                GtkWidget *focus);
void        gtk_window_set_default          (GtkWindow *window,
                                                GtkWidget *default_widget);
void        gtk_window_present              (GtkWindow *window);
void        gtk_window_iconify              (GtkWindow *window);
void        gtk_window_deiconify            (GtkWindow *window);
void        gtk_window_stick                (GtkWindow *window);
void        gtk_window_unstick              (GtkWindow *window);
void        gtk_window_maximize             (GtkWindow *window);
void        gtk_window_unmaximize           (GtkWindow *window);
```

```
void        gtk_window_fullscreen           (GtkWindow *window);
void        gtk_window_unfullscreen         (GtkWindow *window);
void        gtk_window_set_keep_above       (GtkWindow *window,
                                             gboolean setting);
void        gtk_window_set_keep_below       (GtkWindow *window,
                                             gboolean setting);
void        gtk_window_begin_resize_drag    (GtkWindow *window,
                                             GdkWindowEdge edge,
                                             gint button,
                                             gint root_x,
                                             gint root_y,
                                             guint32 timestamp);
void        gtk_window_begin_move_drag      (GtkWindow *window,
                                             gint button,
                                             gint root_x,
                                             gint root_y,
                                             guint32 timestamp);
void        gtk_window_set_decorated        (GtkWindow *window,
                                             gboolean setting);
void        gtk_window_set_frame_dimensions (GtkWindow *window,
                                             gint left,
                                             gint top,
                                             gint right,
                                             gint bottom);
void        gtk_window_set_has_frame        (GtkWindow *window,
                                             gboolean setting);
void        gtk_window_set_mnemonic_modifier
                                            (GtkWindow *window,
                                             GdkModifierType modifier);
void        gtk_window_set_role             (GtkWindow *window,
                                             const gchar *role);
void        gtk_window_set_type_hint        (GtkWindow *window,
                                             GdkWindowTypeHint hint);
void        gtk_window_set_skip_taskbar_hint
                                            (GtkWindow *window,
                                             gboolean setting);
void        gtk_window_set_skip_pager_hint  (GtkWindow *window,
                                             gboolean setting);
void        gtk_window_set_accept_focus     (GtkWindow *window,
                                             gboolean setting);
void        gtk_window_set_focus_on_map     (GtkWindow *window,
                                             gboolean setting);
gboolean    gtk_window_get_decorated        (GtkWindow *window);
```

```
GList*      gtk_window_get_default_icon_list
                                            (void);
void        gtk_window_get_default_size     (GtkWindow *window,
                                             gint *width,
                                             gint *height);
gboolean    gtk_window_get_destroy_with_parent
                                            (GtkWindow *window);
void        gtk_window_get_frame_dimensions (GtkWindow *window,
                                             gint *left,
                                             gint *top,
                                             gint *right,
                                             gint *bottom);
gboolean    gtk_window_get_has_frame        (GtkWindow *window);
GdkPixbuf*  gtk_window_get_icon             (GtkWindow *window);
GList*      gtk_window_get_icon_list        (GtkWindow *window);
gchar*      gtk_window_get_icon_name        (GtkWindow *window);
GdkModifierType gtk_window_get_mnemonic_modifier
                                            (GtkWindow *window);
gboolean    gtk_window_get_modal            (GtkWindow *window);
void        gtk_window_get_position         (GtkWindow *window,
                                             gint *root_x,
                                             gint *root_y);
G_CONST_RETURN gchar* gtk_window_get_role   (GtkWindow *window);
void        gtk_window_get_size             (GtkWindow *window,
                                             gint *width,
                                             gint *height);
G_CONST_RETURN gchar* gtk_window_get_title  (GtkWindow *window);
GtkWindow*  gtk_window_get_transient_for    (GtkWindow *window);
GdkWindowTypeHint gtk_window_get_type_hint  (GtkWindow *window);
gboolean    gtk_window_get_skip_taskbar_hint
                                            (GtkWindow *window);
gboolean    gtk_window_get_skip_pager_hint  (GtkWindow *window);
gboolean    gtk_window_get_accept_focus     (GtkWindow *window);
gboolean    gtk_window_get_focus_on_map     (GtkWindow *window);
void        gtk_window_move                 (GtkWindow *window,
                                             gint x,
                                             gint y);
gboolean    gtk_window_parse_geometry       (GtkWindow *window,
                                             const gchar *geometry);
void        gtk_window_reshow_with_initial_size
                                            (GtkWindow *window);
void        gtk_window_resize               (GtkWindow *window,
                                             gint width,
```

```
                                                gint height);
void        gtk_window_set_default_icon_list
                                               (GList *list);
void        gtk_window_set_default_icon        (GdkPixbuf *icon);
gboolean    gtk_window_set_default_icon_from_file
                                               (const gchar *filename,
                                                GError **err);
void        gtk_window_set_default_icon_name
                                               (const gchar *name);
void        gtk_window_set_icon                (GtkWindow *window,
                                                GdkPixbuf *icon);
void        gtk_window_set_icon_list           (GtkWindow *window,
                                                GList *list);
gboolean    gtk_window_set_icon_from_file      (GtkWindow *window,
                                                const gchar *filename,
                                                GError **err);
void        gtk_window_set_icon_name           (GtkWindow *window,
                                                const gchar *name);
void        gtk_window_set_auto_startup_notification
                                               (gboolean setting);


void        gtk_decorated_window_init          (GtkWindow *window);
void        gtk_decorated_window_calculate_frame_size
                                               (GtkWindow *window);
void        gtk_decorated_window_set_title     (GtkWindow *window,
                                                const gchar *title);
void        gtk_decorated_window_move_resize_window
                                               (GtkWindow *window,
                                                gint x,
                                                gint y,
                                                gint width,
                                                gint height);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkWidget
               +----GtkContainer
                     +----GtkBin
```

```
                        +----GtkWindow
                              +----GtkDialog
                              +----GtkPlug
```

Implemented Interfaces

GtkWindow implements AtkImplementorIface.

Properties

```
  "accept-focus"         gboolean              : Read / Write
  "allow-grow"           gboolean              : Read / Write
  "allow-shrink"         gboolean              : Read / Write
  "decorated"            gboolean              : Read / Write
  "default-height"       gint                  : Read / Write
  "default-width"        gint                  : Read / Write
  "destroy-with-parent"  gboolean              : Read / Write
  "focus-on-map"         gboolean              : Read / Write
  "gravity"              GdkGravity            : Read / Write
  "has-toplevel-focus"   gboolean              : Read
  "icon"                 GdkPixbuf             : Read / Write
  "icon-name"            gchararray            : Read / Write
  "is-active"            gboolean              : Read
  "modal"                gboolean              : Read / Write
  "resizable"            gboolean              : Read / Write
  "role"                 gchararray            : Read / Write
  "screen"               GdkScreen             : Read / Write
  "skip-pager-hint"      gboolean              : Read / Write
  "skip-taskbar-hint"    gboolean              : Read / Write
  "title"                gchararray            : Read / Write
  "type"                 GtkWindowType         : Read / Write / Construct Only
  "type-hint"            GdkWindowTypeHint     : Read / Write
  "window-position"      GtkWindowPosition     : Read / Write
```

Signal Prototypes

```
"activate-default"
            void        user_function      (GtkWindow *window,
```

```
                                                gpointer user_data);
"activate-focus"
            void        user_function      (GtkWindow *window,
                                                gpointer user_data);
"frame-event"
            gboolean    user_function      (GtkWindow *window,
                                                GdkEvent *event,
                                                gpointer user_data);
"keys-changed"
            void        user_function      (GtkWindow *window,
                                                gpointer user_data);
"move-focus"
            void        user_function      (GtkWindow *window,
                                                GtkDirectionType arg1,
                                                gpointer user_data);
"set-focus" void        user_function      (GtkWindow *window,
                                                GtkWidget *widget,
                                                gpointer user_data);
```

Description

Details

GtkWindow

```
typedef struct _GtkWindow GtkWindow;
```

---

gtk_window_new ()

```
GtkWidget*  gtk_window_new                  (GtkWindowType type);
```

Creates a new GtkWindow, which is a toplevel window that can contain other widgets. Nearly always, the type of the window should be GTK_WINDOW_TOPLEVEL. If you're implementing something like a popup menu from scratch (which is a bad idea, just use GtkMenu), you might use GTK_WINDOW_POPUP. GTK_WINDOW_POPUP is not for dialogs, though in some other toolkits dialogs are called "popups". In GTK+, GTK_WINDOW_POPUP means a pop-up menu or pop-up tooltip. On X11, popup windows are not controlled by the window manager.

If you simply want an undecorated window (no window borders), use `gtk_window_set_decorated()`, don't use GTK_WINDOW_POPUP.

type : type of window

Returns : a new GtkWindow.

---

gtk_window_set_title ()

```
void        gtk_window_set_title            (GtkWindow *window,
                                             const gchar *title);
```

Sets the title of the GtkWindow. The title of a window will be displayed in its title bar; on the X Window System, the title bar is rendered by the window manager, so exactly how the title appears to users may vary according to a user's exact configuration. The title should help a user distinguish this window from other windows they may have open. A good title might include the application name and current document filename, for example.

window : a GtkWindow

title : title of the window

---

gtk_window_set_wmclass ()

```
void        gtk_window_set_wmclass          (GtkWindow *window,
                                             const gchar *wmclass_name,
                                             const gchar *wmclass_class);
```

Don't use this function. It sets the X Window System "class" and "name" hints for a window. According to the ICCCM, you should always set these to the same value for all windows in an application, and GTK+ sets them to that value by default, so calling this function is sort of pointless. However, you may want to call `gtk_window_set_role()` on each window in your application, for the benefit of the session manager. Setting the role allows the window manager to restore window positions when loading a saved session.

window : a GtkWindow

wmclass_name : window name hint

wmclass_class : window class hint

---

gtk_window_set_policy ()

```
void        gtk_window_set_policy           (GtkWindow *window,
                                             gint allow_shrink,
                                             gint allow_grow,
                                             gint auto_shrink);
```

Warning

`gtk_window_set_policy` is deprecated and should not be used in newly-written code. Use `gtk_window_set_resizable()` instead.

Changes how a toplevel window deals with its size request and user resize attempts. There are really only two reasonable ways to call this function:

1. `gtk_window_set_policy (GTK_WINDOW (window), FALSE, TRUE, FALSE)` means that the window is user-resizable.
2. `gtk_window_set_policy (GTK_WINDOW (window), FALSE, FALSE, TRUE)` means that the window's size is program-controlled, and should simply match the current size request of the window's children.

The first policy is the default, that is, by default windows are designed to be resized by users.

The basic ugly truth of this function is that it should be simply: `void gtk_window_set_resizable (GtkWindow* window, gboolean setting);` ...which is why GTK+ 2.0 introduces `gtk_window_set_resizable()`, which you should use instead of `gtk_window_set_policy()`.

If set to TRUE, the *`allow_grow`* parameter allows the user to expand the window beyond the size request of its child widgets. If *`allow_grow`* is TRUE, be sure to check that your child widgets work properly as the window is resized.

A toplevel window will always change size to ensure its child widgets receive their requested size. This means that if you add child widgets, the toplevel window will expand to contain them. However, normally the toplevel will not shrink to fit the size request of its children if it's too large; the *`auto_shrink`* parameter causes the window to shrink when child widgets have too much space. *`auto_shrink`* is normally used with the second of the two window policies mentioned above. That is, set *`auto_shrink`* to TRUE if you want the window to have a fixed, always-optimal size determined by your program.

Note that *`auto_shrink`* doesn't do anything if *`allow_shrink`* and *`allow_grow`* are both set to FALSE.

Neither of the two suggested window policies set the *`allow_shrink`* parameter to TRUE. If *`allow_shrink`* is TRUE, the user can shrink the window so that its children do not receive their full size request; this is basically a bad thing, because most widgets will look wrong if this happens. Furthermore GTK+ has a tendency to re-expand the window if size is recalculated for any reason. The upshot is that *`allow_shrink`* should always be set to FALSE.

Sometimes when you think you want to use *`allow_shrink`*, the real problem is that some specific child widget is requesting too much space, so the user can't shrink the window sufficiently. Perhaps you are calling

gtk_widget_set_size_request() on a child widget, and forcing its size request to be too large. Instead of setting the child's usize, consider using gtk_window_set_default_size() so that the child gets a larger allocation than it requests.

window : the window
allow_shrink : whether the user can shrink the window below its size request
allow_grow : whether the user can grow the window larger than its size request
auto_shrink : whether the window automatically snaps back to its size request if it's larger

---

gtk_window_set_resizable ()

```
void        gtk_window_set_resizable        (GtkWindow *window,
                                             gboolean resizable);
```

Sets whether the user can resize a window. Windows are user resizable by default.

window : a GtkWindow
resizable : TRUE if the user can resize this window

---

gtk_window_get_resizable ()

```
gboolean    gtk_window_get_resizable        (GtkWindow *window);
```

Gets the value set by gtk_window_set_resizable().

window : a GtkWindow
Returns : TRUE if the user can resize the window

---

gtk_window_add_accel_group ()

```
void        gtk_window_add_accel_group      (GtkWindow *window,
                                             GtkAccelGroup *accel_group);
```

Associate *accel_group* with *window*, such that calling gtk_accel_groups_activate() on *window* will

activate accelerators in *accel_group*.

window : window to attach accelerator group to
accel_group : a GtkAccelGroup

---

gtk_window_remove_accel_group ()

```
void        gtk_window_remove_accel_group   (GtkWindow *window,
                                             GtkAccelGroup *accel_group);
```

Reverses the effects of gtk_window_add_accel_group().

window : a GtkWindow
accel_group : a GtkAccelGroup

---

gtk_window_position

```
#define gtk_window_position                     gtk_window_set_position
```

Warning

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

Deprecated alias for gtk_window_set_position().

---

gtk_window_activate_focus ()

```
gboolean    gtk_window_activate_focus       (GtkWindow *window);
```

Activates the current focused widget within the window.

window : a GtkWindow
Returns : TRUE if a widget got activated.

---

gtk_window_activate_default ()

```
gboolean    gtk_window_activate_default     (GtkWindow *window);
```

Activates the default widget for the window, unless the current focused widget has been configured to receive the default action (see GTK_RECEIVES_DEFAULT in GtkWidgetFlags), in which case the focused widget is activated.

window : a GtkWindow

Returns : TRUE if a widget got activated.

---

gtk_window_set_modal ()

```
void        gtk_window_set_modal            (GtkWindow *window,
                                             gboolean modal);
```

Sets a window modal or non-modal. Modal windows prevent interaction with other windows in the same application. To keep modal dialogs on top of main application windows, use `gtk_window_set_transient_for()` to make the dialog transient for the parent; most window managers will then disallow lowering the dialog below the parent.

window : a GtkWindow

modal : whether the window is modal

---

gtk_window_set_default_size ()

```
void        gtk_window_set_default_size     (GtkWindow *window,
                                             gint width,
                                             gint height);
```

Sets the default size of a window. If the window's "natural" size (its size request) is larger than the default, the default will be ignored. More generally, if the default size does not obey the geometry hints for the window (`gtk_window_set_geometry_hints()` can be used to set these explicitly), the default size will be clamped to the nearest permitted size.

Unlike `gtk_widget_set_size_request()`, which sets a size request for a widget and thus would keep users from shrinking the window, this function only sets the initial size, just as if the user had resized the window

themselves. Users can still shrink the window again as they normally would. Setting a default size of -1 means to use the "natural" default size (the size request of the window).

For more control over a window's initial size and how resizing works, investigate `gtk_window_set_geometry_hints()`.

For some uses, `gtk_window_resize()` is a more appropriate function. `gtk_window_resize()` changes the current size of the window, rather than the size to be used on initial display. `gtk_window_resize()` always affects the window itself, not the geometry widget.

The default size of a window only affects the first time a window is shown; if a window is hidden and re-shown, it will remember the size it had prior to hiding, rather than using the default size.

Windows can't actually be 0x0 in size, they must be at least 1x1, but passing 0 for *`width`* and *`height`* is OK, resulting in a 1x1 default size.

`window` : a GtkWindow
`width` : width in pixels, or -1 to unset the default width
`height` : height in pixels, or -1 to unset the default height

---

gtk_window_set_geometry_hints ()

```
void        gtk_window_set_geometry_hints   (GtkWindow *window,
                                             GtkWidget *geometry_widget,
                                             GdkGeometry *geometry,
                                             GdkWindowHints geom_mask);
```

This function sets up hints about how a window can be resized by the user. You can set a minimum and maximum size; allowed resize increments (e.g. for xterm, you can only resize by the size of a character); aspect ratios; and more. See the GdkGeometry struct.

`window` : a GtkWindow
`geometry_widget` : widget the geometry hints will be applied to
`geometry` : struct containing geometry information
`geom_mask` : mask indicating which struct fields should be paid attention to

---

gtk_window_set_gravity ()

```
```

```
void        gtk_window_set_gravity          (GtkWindow *window,
                                             GdkGravity gravity);
```

Window gravity defines the meaning of coordinates passed to `gtk_window_move()`. See `gtk_window_move()` and GdkGravity for more details.

The default window gravity is GDK_GRAVITY_NORTH_WEST which will typically "do what you mean."

window : a GtkWindow
gravity : window gravity

---

gtk_window_get_gravity ()

```
GdkGravity  gtk_window_get_gravity          (GtkWindow *window);
```

Gets the value set by `gtk_window_set_gravity()`.

window : a GtkWindow
Returns : window gravity

---

gtk_window_set_position ()

```
void        gtk_window_set_position         (GtkWindow *window,
                                             GtkWindowPosition position);
```

Sets a position constraint for this window. If the old or new constraint is `GTK_WIN_POS_CENTER_ALWAYS`, this will also cause the window to be repositioned to satisfy the new constraint.

window : a GtkWindow.
position : a position constraint.

---

gtk_window_set_transient_for ()

```
void        gtk_window_set_transient_for    (GtkWindow *window,
```

```
                                                GtkWindow *parent);
```

Dialog windows should be set transient for the main application window they were spawned from. This allows window managers to e.g. keep the dialog on top of the main window, or center the dialog over the main window. `gtk_dialog_new_with_buttons()` and other convenience functions in GTK+ will sometimes call `gtk_window_set_transient_for()` on your behalf.

On Windows, this function will and put the child window on top of the parent, much as the window manager would have done on X.

window : a GtkWindow

parent : parent window

---

gtk_window_set_destroy_with_parent ()

```
void        gtk_window_set_destroy_with_parent
                                            (GtkWindow *window,
                                             gboolean setting);
```

If *setting* is TRUE, then destroying the transient parent of *window* will also destroy *window* itself. This is useful for dialogs that shouldn't persist beyond the lifetime of the main window they're associated with, for example.

window : a GtkWindow

setting : whether to destroy *window* with its transient parent

---

gtk_window_set_screen ()

```
void        gtk_window_set_screen           (GtkWindow *window,
                                             GdkScreen *screen);
```

Sets the GdkScreen where the *window* is displayed; if the window is already mapped, it will be unmapped, and then remapped on the new screen.

window : a GtkWindow.

screen : a GdkScreen.

Since 2.2

---

gtk_window_get_screen ()

```
GdkScreen*  gtk_window_get_screen           (GtkWindow *window);
```

Returns the GdkScreen associated with *window*.

window : a GtkWindow.

Returns : a GdkScreen.

Since 2.2

---

gtk_window_is_active ()

```
gboolean    gtk_window_is_active            (GtkWindow *window);
```

Returns whether the window is part of the current active toplevel. (That is, the toplevel window receiving keystrokes.) The return value is TRUE if the window is active toplevel itself, but also if it is, say, a GtkPlug embedded in the active toplevel. You might use this function if you wanted to draw a widget differently in an active window from a widget in an inactive window. See `gtk_window_has_toplevel_focus()`

window : a GtkWindow

Returns : TRUE if the window part of the current active window.

Since 2.4

---

gtk_window_has_toplevel_focus ()

```
gboolean    gtk_window_has_toplevel_focus   (GtkWindow *window);
```

Returns whether the input focus is within this GtkWindow. For real toplevel windows, this is identical to

`gtk_window_is_active()`, but for embedded windows, like GtkPlug, the results will differ.

`window` : a GtkWindow

Returns : TRUE if the the input focus is within this GtkWindow

Since 2.4

---

gtk_window_list_toplevels ()

```
GList*      gtk_window_list_toplevels       (void);
```

Returns a list of all existing toplevel windows. The widgets in the list are not individually referenced. If you want to iterate through the list and perform actions involving callbacks that might destroy the widgets, you *must* call `g_list_foreach (result, (GFunc)g_object_ref, NULL)` first, and then unref all the widgets afterwards.

Returns : list of toplevel widgets

---

gtk_window_add_mnemonic ()

```
void        gtk_window_add_mnemonic         (GtkWindow *window,
                                             guint keyval,
                                             GtkWidget *target);
```

Adds a mnemonic to this window.

`window` : a GtkWindow

`keyval` : the mnemonic

`target` : the widget that gets activated by the mnemonic

---

gtk_window_remove_mnemonic ()

```
void        gtk_window_remove_mnemonic      (GtkWindow *window,
                                             guint keyval,
```

```
                                                GtkWidget *target);
```

Removes a mnemonic from this window.

window : a GtkWindow
keyval : the mnemonic
target : the widget that gets activated by the mnemonic

---

gtk_window_mnemonic_activate ()

```
gboolean    gtk_window_mnemonic_activate    (GtkWindow *window,
                                             guint keyval,
                                             GdkModifierType modifier);
```

Activates the targets associated with the mnemonic.

window : a GtkWindow
keyval : the mnemonic
modifier : the modifiers
Returns : TRUE if the activation is done.

---

gtk_window_activate_key ()

```
gboolean    gtk_window_activate_key         (GtkWindow *window,
                                             GdkEventKey *event);
```

Activates mnemonics and accelerators for this GtkWindow. This is normally called by the default ::key_press_event handler for toplevel windows, however in some cases it may be useful to call this directly when overriding the standard key handling for a toplevel window.

window : a GtkWindow
event : a GdkEventKey
Returns : TRUE if a mnemonic or accelerator was found and activated.

---

gtk_window_propagate_key_event ()

```
gboolean    gtk_window_propagate_key_event  (GtkWindow *window,
                                             GdkEventKey *event);
```

Propagate a key press or release event to the focus widget and up the focus container chain until a widget handles *event*. This is normally called by the default ::key_press_event and ::key_release_event handlers for toplevel windows, however in some cases it may be useful to call this directly when overriding the standard key handling for a toplevel window.

window : a GtkWindow

event : a GdkEventKey

Returns : TRUE if a widget in the focus chain handled the event.

---

gtk_window_get_focus ()

```
GtkWidget*  gtk_window_get_focus            (GtkWindow *window);
```

Retrieves the current focused widget within the window. Note that this is the widget that would have the focus if the toplevel window focused; if the toplevel window is not focused then GTK_WIDGET_HAS_FOCUS (widget) will not be TRUE for the widget.

window : a GtkWindow

Returns : the currently focused widget, or NULL if there is none.

---

gtk_window_set_focus ()

```
void        gtk_window_set_focus            (GtkWindow *window,
                                             GtkWidget *focus);
```

If *focus* is not the current focus widget, and is focusable, sets it as the focus widget for the window. If *focus* is NULL, unsets the focus widget for this window. To set the focus to a particular widget in the toplevel, it is usually more convenient to use gtk_widget_grab_focus() instead of this function.

window : a GtkWindow

focus : widget to be the new focus widget, or NULL to unset any focus widget for the toplevel window.

---

gtk_window_set_default ()

```
void        gtk_window_set_default          (GtkWindow *window,
                                             GtkWidget *default_widget);
```

The default widget is the widget that's activated when the user presses Enter in a dialog (for example). This function sets or unsets the default widget for a GtkWindow about. When setting (rather than unsetting) the default widget it's generally easier to call `gtk_widget_grab_focus()` on the widget. Before making a widget the default widget, you must set the GTK_CAN_DEFAULT flag on the widget you'd like to make the default using `GTK_WIDGET_SET_FLAGS()`.

window : a GtkWindow

default_widget : widget to be the default, or NULL to unset the default widget for the toplevel.

---

gtk_window_present ()

```
void        gtk_window_present              (GtkWindow *window);
```

Presents a window to the user. This may mean raising the window in the stacking order, deiconifying it, moving it to the current desktop, and/or giving it the keyboard focus, possibly dependent on the user's platform, window manager, and preferences.

If *window* is hidden, this function calls `gtk_widget_show()` as well.

This function should be used when the user tries to open a window that's already open. Say for example the preferences dialog is currently open, and the user chooses Preferences from the menu a second time; use `gtk_window_present()` to move the already-open dialog where the user can see it.

window : a GtkWindow

---

gtk_window_iconify ()

```
void        gtk_window_iconify              (GtkWindow *window);
```

Asks to iconify (i.e. minimize) the specified *window*. Note that you shouldn't assume the window is definitely

iconified afterward, because other entities (e.g. the user or window manager) could deiconify it again, or there may not be a window manager in which case iconification isn't possible, etc. But normally the window will end up iconified. Just don't write code that crashes if not.

It's permitted to call this function before showing a window, in which case the window will be iconified before it ever appears onscreen.

You can track iconification via the "window_state_event" signal on GtkWidget.

window : a GtkWindow

---

gtk_window_deiconify ()

```
void        gtk_window_deiconify            (GtkWindow *window);
```

Asks to deiconify (i.e. unminimize) the specified *window*. Note that you shouldn't assume the window is definitely deiconified afterward, because other entities (e.g. the user or window manager) could iconify it again before your code which assumes deiconification gets to run.

You can track iconification via the "window_state_event" signal on GtkWidget.

window : a GtkWindow

---

gtk_window_stick ()

```
void        gtk_window_stick                (GtkWindow *window);
```

Asks to stick *window*, which means that it will appear on all user desktops. Note that you shouldn't assume the window is definitely stuck afterward, because other entities (e.g. the user or window manager) could unstick it again, and some window managers do not support sticking windows. But normally the window will end up stuck. Just don't write code that crashes if not.

It's permitted to call this function before showing a window.

You can track stickiness via the "window_state_event" signal on GtkWidget.

window : a GtkWindow

---

gtk_window_unstick ()

```
void        gtk_window_unstick              (GtkWindow *window);
```

Asks to unstick *window*, which means that it will appear on only one of the user's desktops. Note that you shouldn't assume the window is definitely unstuck afterward, because other entities (e.g. the user or window manager) could stick it again. But normally the window will end up stuck. Just don't write code that crashes if not.

You can track stickiness via the "window_state_event" signal on GtkWidget.

window : a GtkWindow

---

gtk_window_maximize ()

```
void        gtk_window_maximize             (GtkWindow *window);
```

Asks to maximize *window*, so that it becomes full-screen. Note that you shouldn't assume the window is definitely maximized afterward, because other entities (e.g. the user or window manager) could unmaximize it again, and not all window managers support maximization. But normally the window will end up maximized. Just don't write code that crashes if not.

It's permitted to call this function before showing a window, in which case the window will be maximized when it appears onscreen initially.

You can track maximization via the "window_state_event" signal on GtkWidget.

window : a GtkWindow

---

gtk_window_unmaximize ()

```
void        gtk_window_unmaximize           (GtkWindow *window);
```

Asks to unmaximize *window*. Note that you shouldn't assume the window is definitely unmaximized afterward, because other entities (e.g. the user or window manager) could maximize it again, and not all window managers honor requests to unmaximize. But normally the window will end up unmaximized. Just don't write code that crashes if not.

You can track maximization via the "window_state_event" signal on GtkWidget.

window : a GtkWindow

---

gtk_window_fullscreen ()

```
void        gtk_window_fullscreen           (GtkWindow *window);
```

Asks to place *window* in the fullscreen state. Note that you shouldn't assume the window is definitely full screen afterward, because other entities (e.g. the user or window manager) could unfullscreen it again, and not all window managers honor requests to fullscreen windows. But normally the window will end up fullscreen. Just don't write code that crashes if not.

You can track the fullscreen state via the "window_state_event" signal on GtkWidget.

window : a GtkWindow

Since 2.2

---

gtk_window_unfullscreen ()

```
void        gtk_window_unfullscreen         (GtkWindow *window);
```

Asks to toggle off the fullscreen state for *window*. Note that you shouldn't assume the window is definitely not full screen afterward, because other entities (e.g. the user or window manager) could fullscreen it again, and not all window managers honor requests to unfullscreen windows. But normally the window will end up restored to its normal state. Just don't write code that crashes if not.

You can track the fullscreen state via the "window_state_event" signal on GtkWidget.

window : a GtkWindow

Since 2.2

---

gtk_window_set_keep_above ()

```
void        gtk_window_set_keep_above       (GtkWindow *window,
                                             gboolean setting);
```

Asks to keep *window* above, so that it stays on top. Note that you shouldn't assume the window is definitely above afterward, because other entities (e.g. the user or window manager) could not keep it above, and not all window managers support keeping windows above. But normally the window will end kept above. Just don't write code that crashes if not.

It's permitted to call this function before showing a window, in which case the window will be kept above when it appears onscreen initially.

You can track the above state via the "window_state_event" signal on GtkWidget.

Note that, according to the Extended Window Manager Hints specification, the above state is mainly meant for user preferences and should not be used by applications e.g. for drawing attention to their dialogs.

window : a GtkWindow

setting : whether to keep *window* above other windows

Since 2.4

---

gtk_window_set_keep_below ()

```
void        gtk_window_set_keep_below       (GtkWindow *window,
                                             gboolean setting);
```

Asks to keep *window* below, so that it stays in bottom. Note that you shouldn't assume the window is definitely below afterward, because other entities (e.g. the user or window manager) could not keep it below, and not all window managers support putting windows below. But normally the window will be kept below. Just don't write code that crashes if not.

It's permitted to call this function before showing a window, in which case the window will be kept below when it appears onscreen initially.

You can track the below state via the "window_state_event" signal on GtkWidget.

Note that, according to the Extended Window Manager Hints specification, the above state is mainly meant for user

preferences and should not be used by applications e.g. for drawing attention to their dialogs.

window : a GtkWindow
setting : whether to keep *window* below other windows

Since 2.4

---

gtk_window_begin_resize_drag ()

```
void        gtk_window_begin_resize_drag    (GtkWindow *window,
                                             GdkWindowEdge edge,
                                             gint button,
                                             gint root_x,
                                             gint root_y,
                                             guint32 timestamp);
```

Starts resizing a window. This function is used if an application has window resizing controls. When GDK can support it, the resize will be done using the standard mechanism for the window manager or windowing system. Otherwise, GDK will try to emulate window resizing, potentially not all that well, depending on the windowing system.

window : a GtkWindow
edge : position of the resize control
button : mouse button that initiated the drag
root_x : X position where the user clicked to initiate the drag, in root window coordinates
root_y : Y position where the user clicked to initiate the drag
timestamp : timestamp from the click event that initiated the drag

---

gtk_window_begin_move_drag ()

```
void        gtk_window_begin_move_drag      (GtkWindow *window,
                                             gint button,
                                             gint root_x,
                                             gint root_y,
                                             guint32 timestamp);
```

Starts moving a window. This function is used if an application has window movement grips. When GDK can support

it, the window movement will be done using the standard mechanism for the window manager or windowing system. Otherwise, GDK will try to emulate window movement, potentially not all that well, depending on the windowing system.

window : a GtkWindow
button : mouse button that initiated the drag
root_x : X position where the user clicked to initiate the drag, in root window coordinates
root_y : Y position where the user clicked to initiate the drag
timestamp : timestamp from the click event that initiated the drag

---

gtk_window_set_decorated ()

```
void        gtk_window_set_decorated        (GtkWindow *window,
                                             gboolean setting);
```

By default, windows are decorated with a title bar, resize controls, etc. Some window managers allow GTK+ to disable these decorations, creating a borderless window. If you set the decorated property to FALSE using this function, GTK+ will do its best to convince the window manager not to decorate the window. Depending on the system, this function may not have any effect when called on a window that is already visible, so you should call it before calling `gtk_window_show()`.

On Windows, this function always works, since there's no window manager policy involved.

window : a GtkWindow
setting : TRUE to decorate the window

---

gtk_window_set_frame_dimensions ()

```
void        gtk_window_set_frame_dimensions (GtkWindow *window,
                                             gint left,
                                             gint top,
                                             gint right,
                                             gint bottom);
```

(Note: this is a special-purpose function intended for the framebuffer port; see `gtk_window_set_has_frame()`. It will have no effect on the window border drawn by the window manager, which is the normal case when using the X Window system.)

For windows with frames (see `gtk_window_set_has_frame()`) this function can be used to change the size of the frame border.

window : a GtkWindow that has a frame
left : The width of the left border
top : The height of the top border
right : The width of the right border
bottom : The height of the bottom border

---

gtk_window_set_has_frame ()

```
void        gtk_window_set_has_frame        (GtkWindow *window,
                                             gboolean setting);
```

(Note: this is a special-purpose function for the framebuffer port, that causes GTK+ to draw its own window border. For most applications, you want `gtk_window_set_decorated()` instead, which tells the window manager whether to draw the window border.)

If this function is called on a window with setting of TRUE, before it is realized or showed, it will have a "frame" window around *window->window*, accessible in *window->frame*. Using the signal frame_event you can receive all events targeted at the frame.

This function is used by the linux-fb port to implement managed windows, but it could concievably be used by X-programs that want to do their own window decorations.

window : a GtkWindow
setting : a boolean

---

gtk_window_set_mnemonic_modifier ()

```
void        gtk_window_set_mnemonic_modifier
                                            (GtkWindow *window,
                                             GdkModifierType modifier);
```

Sets the mnemonic modifier for this window.

window : a GtkWindow

modifier : the modifier mask used to activate mnemonics on this window.

---

gtk_window_set_role ()

```
void        gtk_window_set_role             (GtkWindow *window,
                                             const gchar *role);
```

This function is only useful on X11, not with other GTK+ targets.

In combination with the window title, the window role allows a window manager to identify "the same" window when an application is restarted. So for example you might set the "toolbox" role on your app's toolbox window, so that when the user restarts their session, the window manager can put the toolbox back in the same place.

If a window already has a unique title, you don't need to set the role, since the WM can use the title to identify the window when restoring the session.

window : a GtkWindow
role : unique identifier for the window to be used when restoring a session

---

gtk_window_set_type_hint ()

```
void        gtk_window_set_type_hint        (GtkWindow *window,
                                             GdkWindowTypeHint hint);
```

By setting the type hint for the window, you allow the window manager to decorate and handle the window in a way which is suitable to the function of the window in your application.

This function should be called before the window becomes visible.

`gtk_dialog_new_with_buttons()` and other convenience functions in GTK+ will sometimes call `gtk_window_set_type_hint()` on your behalf.

window : a GtkWindow
hint : the window type

---

gtk_window_set_skip_taskbar_hint ()

```
void        gtk_window_set_skip_taskbar_hint
                                            (GtkWindow *window,
                                             gboolean setting);
```

Windows may set a hint asking the desktop environment not to display the window in the task bar. This function sets this hint.

window : a GtkWindow

setting : TRUE to keep this window from appearing in the task bar

Since 2.2

---

gtk_window_set_skip_pager_hint ()

```
void        gtk_window_set_skip_pager_hint  (GtkWindow *window,
                                             gboolean setting);
```

Windows may set a hint asking the desktop environment not to display the window in the pager. This function sets this hint. (A "pager" is any desktop navigation tool such as a workspace switcher that displays a thumbnail representation of the windows on the screen.)

window : a GtkWindow

setting : TRUE to keep this window from appearing in the pager

Since 2.2

---

gtk_window_set_accept_focus ()

```
void        gtk_window_set_accept_focus     (GtkWindow *window,
                                             gboolean setting);
```

Windows may set a hint asking the desktop environment not to receive the input focus. This function sets this hint.

window : a GtkWindow

setting : TRUE to let this window receive input focus

Since 2.4

---

gtk_window_set_focus_on_map ()

```
void        gtk_window_set_focus_on_map     (GtkWindow *window,
                                             gboolean setting);
```

Windows may set a hint asking the desktop environment not to receive the input focus when the window is mapped. This function sets this hint.

window : a GtkWindow
setting : TRUE to let this window receive input focus on map

Since 2.6

---

gtk_window_get_decorated ()

```
gboolean    gtk_window_get_decorated        (GtkWindow *window);
```

Returns whether the window has been set to have decorations such as a title bar via gtk_window_set_decorated().

window : a GtkWindow
Returns : TRUE if the window has been set to have decorations

---

gtk_window_get_default_icon_list ()

```
GList*      gtk_window_get_default_icon_list
                                            (void);
```

Gets the value set by gtk_window_set_default_icon_list(). The list is a copy and should be freed with

g_list_free(), but the pixbufs in the list have not had their reference count incremented.

Returns : copy of default icon list

---

gtk_window_get_default_size ()

```
void        gtk_window_get_default_size     (GtkWindow *window,
                                             gint *width,
                                             gint *height);
```

Gets the default size of the window. A value of -1 for the width or height indicates that a default size has not been explicitly set for that dimension, so the "natural" size of the window will be used.

window : a GtkWindow

width : location to store the default width, or NULL

height : location to store the default height, or NULL

---

gtk_window_get_destroy_with_parent ()

```
gboolean    gtk_window_get_destroy_with_parent
                                             (GtkWindow *window);
```

Returns whether the window will be destroyed with its transient parent. See gtk_window_set_destroy_with_parent().

window : a GtkWindow

Returns : TRUE if the window will be destroyed with its transient parent.

---

gtk_window_get_frame_dimensions ()

```
void        gtk_window_get_frame_dimensions (GtkWindow *window,
                                             gint *left,
                                             gint *top,
                                             gint *right,
                                             gint *bottom);
```

(Note: this is a special-purpose function intended for the framebuffer port; see gtk_window_set_has_frame(). It will not return the size of the window border drawn by the window manager, which is the normal case when using a windowing system. See gdk_window_get_frame_extents() to get the standard window border extents.)

Retrieves the dimensions of the frame window for this toplevel. See gtk_window_set_has_frame(), gtk_window_set_frame_dimensions().

window : a GtkWindow
left : location to store the width of the frame at the left, or NULL
top : location to store the height of the frame at the top, or NULL
right : location to store the width of the frame at the returns, or NULL
bottom : location to store the height of the frame at the bottom, or NULL

---

gtk_window_get_has_frame ()

```
gboolean    gtk_window_get_has_frame        (GtkWindow *window);
```

Accessor for whether the window has a frame window exterior to *window->window*. Gets the value set by gtk_window_set_has_frame().

window : a GtkWindow
Returns : TRUE if a frame has been added to the window via gtk_window_set_has_frame().

---

gtk_window_get_icon ()

```
GdkPixbuf*  gtk_window_get_icon             (GtkWindow *window);
```

Gets the value set by gtk_window_set_icon() (or if you've called gtk_window_set_icon_list(), gets the first icon in the icon list).

window : a GtkWindow
Returns : icon for window

---

gtk_window_get_icon_list ()

```
GList*      gtk_window_get_icon_list        (GtkWindow *window);
```

Retrieves the list of icons set by gtk_window_set_icon_list(). The list is copied, but the reference count on each member won't be incremented.

window : a GtkWindow
Returns : copy of window's icon list

---

gtk_window_get_icon_name ()

```
gchar*      gtk_window_get_icon_name        (GtkWindow *window);
```

Returns the name of the themed icon for the window, see gtk_window_set_icon_name().

window : a GtkWindow
Returns : the icon name or NULL if the window has no themed icon

Since 2.6

---

gtk_window_get_mnemonic_modifier ()

```
GdkModifierType gtk_window_get_mnemonic_modifier
                                            (GtkWindow *window);
```

Returns the mnemonic modifier for this window. See gtk_window_set_mnemonic_modifier().

window : a GtkWindow
Returns : the modifier mask used to activate mnemonics on this window.

---

gtk_window_get_modal ()

```
gboolean    gtk_window_get_modal            (GtkWindow *window);
```

Returns whether the window is modal. See gtk_window_set_modal().

window : a GtkWindow

Returns : TRUE if the window is set to be modal and establishes a grab when shown

---

gtk_window_get_position ()

```
void        gtk_window_get_position         (GtkWindow *window,
                                             gint *root_x,
                                             gint *root_y);
```

This function returns the position you need to pass to gtk_window_move() to keep *window* in its current position. This means that the meaning of the returned value varies with window gravity. See gtk_window_move() for more details.

If you haven't changed the window gravity, its gravity will be GDK_GRAVITY_NORTH_WEST. This means that gtk_window_get_position() gets the position of the top-left corner of the window manager frame for the window. gtk_window_move() sets the position of this same top-left corner.

gtk_window_get_position() is not 100% reliable because the X Window System does not specify a way to obtain the geometry of the decorations placed on a window by the window manager. Thus GTK+ is using a "best guess" that works with most window managers.

Moreover, nearly all window managers are historically broken with respect to their handling of window gravity. So moving a window to its current position as returned by gtk_window_get_position() tends to result in moving the window slightly. Window managers are slowly getting better over time.

If a window has gravity GDK_GRAVITY_STATIC the window manager frame is not relevant, and thus gtk_window_get_position() will always produce accurate results. However you can't use static gravity to do things like place a window in a corner of the screen, because static gravity ignores the window manager decorations.

If you are saving and restoring your application's window positions, you should know that it's impossible for applications to do this without getting it somewhat wrong because applications do not have sufficient knowledge of window manager state. The Correct Mechanism is to support the session management protocol (see the "GnomeClient" object in the GNOME libraries for example) and allow the window manager to save your window sizes and positions.

window : a GtkWindow

root_x : return location for X coordinate of gravity-determined reference p\oint

root_y : return location for Y coordinate of gravity-determined reference p\oint

---

gtk_window_get_role ()

```
G_CONST_RETURN gchar* gtk_window_get_role   (GtkWindow *window);
```

Returns the role of the window. See gtk_window_set_role() for further explanation.

window : a GtkWindow

Returns : the role of the window if set, or NULL. The returned is owned by the widget and must not be modified or freed.

---

gtk_window_get_size ()

```
void        gtk_window_get_size             (GtkWindow *window,
                                             gint *width,
                                             gint *height);
```

Obtains the current size of *window*. If *window* is not onscreen, it returns the size GTK+ will suggest to the window manager for the initial window size (but this is not reliably the same as the size the window manager will actually select). The size obtained by gtk_window_get_size() is the last size received in a GdkEventConfigure, that is, GTK+ uses its locally-stored size, rather than querying the X server for the size. As a result, if you call gtk_window_resize() then immediately call gtk_window_get_size(), the size won't have taken effect yet. After the window manager processes the resize request, GTK+ receives notification that the size has changed via a configure event, and the size of the window gets updated.

Note 1: Nearly any use of this function creates a race condition, because the size of the window may change between the time that you get the size and the time that you perform some action assuming that size is the current size. To avoid race conditions, connect to "configure_event" on the window and adjust your size-dependent state to match the size delivered in the GdkEventConfigure.

Note 2: The returned size does *not* include the size of the window manager decorations (aka the window frame or border). Those are not drawn by GTK+ and GTK+ has no reliable method of determining their size.

Note 3: If you are getting a window size in order to position the window onscreen, there may be a better way. The preferred way is to simply set the window's semantic type with gtk_window_set_type_hint(), which allows the window manager to e.g. center dialogs. Also, if you set the transient parent of dialogs with gtk_window_set_transient_for() window managers will often center the dialog over its parent window. It's much preferred to let the window manager handle these things rather than doing it yourself, because all apps will

behave consistently and according to user prefs if the window manager handles it. Also, the window manager can take the size of the window decorations/border into account, while your application cannot.

In any case, if you insist on application-specified window positioning, there's *still* a better way than doing it yourself - `gtk_window_set_position()` will frequently handle the details for you.

window : a GtkWindow
width : return location for width, or `NULL`
height : return location for height, or `NULL`

---

gtk_window_get_title ()

```
G_CONST_RETURN gchar* gtk_window_get_title  (GtkWindow *window);
```

Retrieves the title of the window. See `gtk_window_set_title()`.

window : a GtkWindow
Returns : the title of the window, or `NULL` if none has been set explicitely. The returned string is owned by the widget and must not be modified or freed.

---

gtk_window_get_transient_for ()

```
GtkWindow*  gtk_window_get_transient_for    (GtkWindow *window);
```

Fetches the transient parent for this window. See `gtk_window_set_transient_for()`.

window : a GtkWindow
Returns : the transient parent for this window, or `NULL` if no transient parent has been set.

---

gtk_window_get_type_hint ()

```
GdkWindowTypeHint gtk_window_get_type_hint  (GtkWindow *window);
```

Gets the type hint for this window. See `gtk_window_set_type_hint()`.

window : a GtkWindow

Returns : the type hint for *window*.

---

gtk_window_get_skip_taskbar_hint ()

```
gboolean    gtk_window_get_skip_taskbar_hint
                                            (GtkWindow *window);
```

Gets the value set by gtk_window_set_skip_taskbar_hint()

window : a GtkWindow

Returns : TRUE if window shouldn't be in taskbar

Since 2.2

---

gtk_window_get_skip_pager_hint ()

```
gboolean    gtk_window_get_skip_pager_hint  (GtkWindow *window);
```

Gets the value set by gtk_window_set_skip_pager_hint().

window : a GtkWindow

Returns : TRUE if window shouldn't be in pager

Since 2.2

---

gtk_window_get_accept_focus ()

```
gboolean    gtk_window_get_accept_focus     (GtkWindow *window);
```

Gets the value set by gtk_window_set_accept_focus().

window : a GtkWindow

Returns : TRUE if window should receive the input focus

Since 2.4

---

gtk_window_get_focus_on_map ()

```
gboolean    gtk_window_get_focus_on_map     (GtkWindow *window);
```

Gets the value set by gtk_window_set_focus_on_map().

window : a GtkWindow

Returns : TRUE if window should receive the input focus when mapped.

Since 2.6

---

gtk_window_move ()

```
void        gtk_window_move                 (GtkWindow *window,
                                             gint x,
                                             gint y);
```

Asks the window manager to move *window* to the given position. Window managers are free to ignore this; most window managers ignore requests for initial window positions (instead using a user-defined placement algorithm) and honor requests after the window has already been shown.

Note: the position is the position of the gravity-determined reference point for the window. The gravity determines two things: first, the location of the reference point in root window coordinates; and second, which point on the window is positioned at the reference point.

By default the gravity is GDK_GRAVITY_NORTH_WEST, so the reference point is simply the *x*, *y* supplied to gtk_window_move(). The top-left corner of the window decorations (aka window frame or border) will be placed at *x*, *y*. Therefore, to position a window at the top left of the screen, you want to use the default gravity (which is GDK_GRAVITY_NORTH_WEST) and move the window to 0,0.

To position a window at the bottom right corner of the screen, you would set GDK_GRAVITY_SOUTH_EAST, which means that the reference point is at *x* + the window width and *y* + the window height, and the bottom-right

corner of the window border will be placed at that reference point. So, to place a window in the bottom right corner you would first set gravity to south east, then write: `gtk_window_move (window, gdk_screen_width() - window_width, gdk_screen_height() - window_height)`.

The Extended Window Manager Hints specification at http://www.freedesktop.org/standards/wm-spec has a nice table of gravities in the "implementation notes" section.

The `gtk_window_get_position()` documentation may also be relevant.

window : a GtkWindow
x : X coordinate to move window to
y : Y coordinate to move window to

---

gtk_window_parse_geometry ()

```
gboolean    gtk_window_parse_geometry       (GtkWindow *window,
                                             const gchar *geometry);
```

Parses a standard X Window System geometry string - see the manual page for X (type 'man X') for details on this. `gtk_window_parse_geometry()` does work on all GTK+ ports including Win32 but is primarily intended for an X environment.

If either a size or a position can be extracted from the geometry string, `gtk_window_parse_geometry()` returns TRUE and calls `gtk_window_set_default_size()` and/or `gtk_window_move()` to resize/move the window.

If `gtk_window_parse_geometry()` returns TRUE, it will also set the GDK_HINT_USER_POS and/or GDK_HINT_USER_SIZE hints indicating to the window manager that the size/position of the window was user-specified. This causes most window managers to honor the geometry.

Note that for `gtk_window_parse_geometry()` to work as expected, it has to be called when the window has its "final" size, i.e. after calling `gtk_widget_show_all()` on the contents and `gtk_window_set_geometry_hints()` on the window.

```
int
main (int argc, char *argv[])
{
  GtkWidget *window, vbox;
  GdkGeometry size_hints;

  gtk_init (&argc, &argv);
```

```
  window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
  vbox = gtk_vbox_new (FALSE, 0);

  gtk_container_add (GTK_CONTAINER (window), vbox);
  fill_with_content (vbox);
  gtk_widget_show_all (vbox);

  size_hints = {
    100, 50, 0, 0, 100, 50, 10, 10, 0.0, 0.0, GDK_GRAVITY_NORTH_WEST
  };

  gtk_window_set_geometry_hints (GTK_WINDOW (window),
                                 window,
                                 &size_hints,
                                 GDK_HINT_MIN_SIZE |
                                 GDK_HINT_BASE_SIZE |
                                 GDK_HINT_RESIZE_INC);

  if (argc > 1)
    {
      if (!gtk_window_parse_geometry (GTK_WINDOW (window), argv[1]))
        fprintf (stderr, "Failed to parse '%s'\n", argv[1]);
    }

   gtk_widget_show_all (window);
   gtk_main();

   return 0;
}
```

window : a GtkWindow

geometry : geometry string

Returns : TRUE if string was parsed successfully

---

gtk_window_reshow_with_initial_size ()

```
void        gtk_window_reshow_with_initial_size
                                            (GtkWindow *window);
```

Hides *window*, then reshows it, resetting the default size and position of the window. Used by GUI builders only.

window : a GtkWindow

---

gtk_window_resize ()

```
void        gtk_window_resize               (GtkWindow *window,
                                             gint width,
                                             gint height);
```

Resizes the window as if the user had done so, obeying geometry constraints. The default geometry constraint is that windows may not be smaller than their size request; to override this constraint, call `gtk_widget_set_size_request()` to set the window's request to a smaller value.

If `gtk_window_resize()` is called before showing a window for the first time, it overrides any default size set with `gtk_window_set_default_size()`.

Windows may not be resized smaller than 1 by 1 pixels.

window : a GtkWindow
width : width in pixels to resize the window to
height : height in pixels to resize the window to

---

gtk_window_set_default_icon_list ()

```
void        gtk_window_set_default_icon_list
                                            (GList *list);
```

Sets an icon list to be used as fallback for windows that haven't had `gtk_window_set_icon_list()` called on them to set up a window-specific icon list. This function allows you to set up the icon for all windows in your app at once.

See `gtk_window_set_icon_list()` for more details.

list : a list of GdkPixbuf

---

gtk_window_set_default_icon ()

```
void        gtk_window_set_default_icon     (GdkPixbuf *icon);
```

Sets an icon to be used as fallback for windows that haven't had gtk_window_set_icon() called on them from a pixbuf.

icon : the icon

Since 2.4

---

gtk_window_set_default_icon_from_file ()

```
gboolean    gtk_window_set_default_icon_from_file
                                            (const gchar *filename,
                                             GError **err);
```

Sets an icon to be used as fallback for windows that haven't had gtk_window_set_icon_list() called on them from a file on disk. Warns on failure if *err* is NULL.

filename : location of icon file
err : location to store error, or NULL.
Returns : TRUE if setting the icon succeeded.

Since 2.2

---

gtk_window_set_default_icon_name ()

```
void        gtk_window_set_default_icon_name
                                            (const gchar *name);
```

Sets an icon to be used as fallback for windows that haven't had gtk_window_set_icon_list() called on them from a named themed icon, see gtk_window_set_icon_name().

name : the name of the themed icon

Since 2.6

---

gtk_window_set_icon ()

```
void        gtk_window_set_icon             (GtkWindow *window,
                                             GdkPixbuf *icon);
```

Sets up the icon representing a GtkWindow. This icon is used when the window is minimized (also known as iconified). Some window managers or desktop environments may also place it in the window frame, or display it in other contexts.

The icon should be provided in whatever size it was naturally drawn; that is, don't scale the image before passing it to GTK+. Scaling is postponed until the last minute, when the desired final size is known, to allow best quality.

If you have your icon hand-drawn in multiple sizes, use `gtk_window_set_icon_list()`. Then the best size will be used.

This function is equivalent to calling `gtk_window_set_icon_list()` with a 1-element list.

See also `gtk_window_set_default_icon_list()` to set the icon for all windows in your application in one go.

window : a GtkWindow

icon : icon image, or NULL

---

gtk_window_set_icon_list ()

```
void        gtk_window_set_icon_list        (GtkWindow *window,
                                             GList *list);
```

Sets up the icon representing a GtkWindow. The icon is used when the window is minimized (also known as iconified). Some window managers or desktop environments may also place it in the window frame, or display it in other contexts.

`gtk_window_set_icon_list()` allows you to pass in the same icon in several hand-drawn sizes. The list should contain the natural sizes your icon is available in; that is, don't scale the image before passing it to GTK+. Scaling is postponed until the last minute, when the desired final size is known, to allow best quality.

By passing several sizes, you may improve the final image quality of the icon, by reducing or eliminating automatic image scaling.

Recommended sizes to provide: 16x16, 32x32, 48x48 at minimum, and larger images (64x64, 128x128) if you have them.

See also `gtk_window_set_default_icon_list()` to set the icon for all windows in your application in one go.

Note that transient windows (those who have been set transient for another window using `gtk_window_set_transient_for()`) will inherit their icon from their transient parent. So there's no need to explicitly set the icon on transient windows.

window : a GtkWindow
list : list of GdkPixbuf

---

gtk_window_set_icon_from_file ()

```
gboolean    gtk_window_set_icon_from_file   (GtkWindow *window,
                                             const gchar *filename,
                                             GError **err);
```

Sets the icon for *window*. Warns on failure if *err* is NULL.

This function is equivalent to calling `gtk_window_set_icon()` with a pixbuf created by loading the image from *filename*.

window : a GtkWindow
filename : location of icon file
err : location to store error, or NULL.
Returns : TRUE if setting the icon succeeded.

Since 2.2

---

gtk_window_set_icon_name ()

```
void        gtk_window_set_icon_name        (GtkWindow *window,
                                             const gchar *name);
```

Sets the icon for the window from a named themed icon. See the docs for GtkIconTheme for more details.

Note that this has nothing to do with the WM_ICON_NAME property which is mentioned in the ICCCM.

window : a GtkWindow
name : the name of the themed icon

Since 2.6

---

gtk_window_set_auto_startup_notification ()

```
void        gtk_window_set_auto_startup_notification
                                            (gboolean setting);
```

By default, after showing the first GtkWindow for each GdkScreen, GTK+ calls `gdk_screen_notify_startup_complete()`. Call this function to disable the automatic startup notification. You might do this if your first window is a splash screen, and you want to delay notification until after your real main window has been shown, for example.

In that example, you would disable startup notification temporarily, show your splash screen, then re-enable it so that showing the main window would automatically result in notification.

setting : TRUE to automatically do startup notification

Since 2.2

---

gtk_decorated_window_init ()

```
void        gtk_decorated_window_init       (GtkWindow *window);
```

window :

---

gtk_decorated_window_calculate_frame_size ()

```
void        gtk_decorated_window_calculate_frame_size
```

```
                                                (GtkWindow *window);
```

window:

---

gtk_decorated_window_set_title ()

```
void        gtk_decorated_window_set_title  (GtkWindow *window,
                                             const gchar *title);
```

window:
title:

---

gtk_decorated_window_move_resize_window ()

```
void        gtk_decorated_window_move_resize_window
                                            (GtkWindow *window,
                                             gint x,
                                             gint y,
                                             gint width,
                                             gint height);
```

window:
x:
y:
width:
height:

Properties

The "accept-focus" property

```
  "accept-focus"         gboolean              : Read / Write
```

TRUE if the window should receive the input focus.

Default value: TRUE

The "allow-grow" property

```
"allow-grow"           gboolean              : Read / Write
```

If TRUE, users can expand the window beyond its minimum size.

Default value: TRUE

The "allow-shrink" property

```
"allow-shrink"         gboolean              : Read / Write
```

If TRUE, the window has no mimimum size. Setting this to TRUE is 99% of the time a bad idea.

Default value: FALSE

The "decorated" property

```
"decorated"            gboolean              : Read / Write
```

Whether the window should be decorated by the window manager.

Default value: TRUE

Since 2.4

The "default-height" property

```
"default-height"       gint                  : Read / Write
```

The default height of the window, used when initially showing the window.

Allowed values: >= -1

Default value: -1

The "default-width" property

```
"default-width"        gint                : Read / Write
```

The default width of the window, used when initially showing the window.

Allowed values: >= -1

Default value: -1

The "destroy-with-parent" property

```
"destroy-with-parent"  gboolean            : Read / Write
```

If this window should be destroyed when the parent is destroyed.

Default value: FALSE

The "focus-on-map" property

```
"focus-on-map"         gboolean            : Read / Write
```

TRUE if the window should receive the input focus when mapped.

Default value: TRUE

The "gravity" property

```
  "gravity"             GdkGravity            : Read / Write
```

The window gravity of the window. See gtk_window_move() and GdkGravity for more details about window gravity.

Default value: GDK_GRAVITY_NORTH_WEST

Since 2.4

The "has-toplevel-focus" property

```
  "has-toplevel-focus"  gboolean              : Read
```

Whether the input focus is within this GtkWindow.

Default value: FALSE

The "icon" property

```
  "icon"                GdkPixbuf             : Read / Write
```

Icon for this window.

The "icon-name" property

```
  "icon-name"           gchararray            : Read / Write
```

The :icon-name property specifies the name of the themed icon to use as the window icon. See GtkIconTheme for more details.

Default value: NULL

Since 2.6

The "is-active" property

"is-active"	gboolean	: Read

Whether the toplevel is the current active window.

Default value: FALSE

The "modal" property

"modal"	gboolean	: Read / Write

If TRUE, the window is modal (other windows are not usable while this one is up).

Default value: FALSE

The "resizable" property

"resizable"	gboolean	: Read / Write

If TRUE, users can resize the window.

Default value: TRUE

The "role" property

"role"	gchararray	: Read / Write

Unique identifier for the window to be used when restoring a session.

Default value: NULL

The "screen" property

```
"screen"              GdkScreen             : Read / Write
```

The screen where this window will be displayed.

The "skip-pager-hint" property

```
"skip-pager-hint"     gboolean              : Read / Write
```

TRUE if the window should not be in the pager.

Default value: FALSE

The "skip-taskbar-hint" property

```
"skip-taskbar-hint"   gboolean              : Read / Write
```

TRUE if the window should not be in the task bar.

Default value: FALSE

The "title" property

```
"title"               gchararray            : Read / Write
```

The title of the window.

Default value: NULL

The "type" property

```
  "type"                GtkWindowType         : Read / Write / Construct Only
```

The type of the window.

Default value: GTK_WINDOW_TOPLEVEL

The "type-hint" property

```
  "type-hint"           GdkWindowTypeHint     : Read / Write
```

Hint to help the desktop environment understand what kind of window this is and how to treat it.

Default value: GDK_WINDOW_TYPE_HINT_NORMAL

The "window-position" property

```
  "window-position"     GtkWindowPosition     : Read / Write
```

The initial position of the window.

Default value: GTK_WIN_POS_NONE

Signals

The "activate-default" signal

```
void        user_function                 (GtkWindow *window,
                                           gpointer user_data);
```

window : the object which received the signal.
user_data : user data set when the signal handler was connected.

The "activate-focus" signal

```
void        user_function                   (GtkWindow *window,
                                             gpointer user_data);
```

window : the object which received the signal.
user_data : user data set when the signal handler was connected.

The "frame-event" signal

```
gboolean    user_function                   (GtkWindow *window,
                                             GdkEvent *event,
                                             gpointer user_data);
```

window : the object which received the signal.
event :
user_data : user data set when the signal handler was connected.
Returns :

The "keys-changed" signal

```
void        user_function                   (GtkWindow *window,
                                             gpointer user_data);
```

window : the object which received the signal.
user_data : user data set when the signal handler was connected.

The "move-focus" signal

```
void        user_function                   (GtkWindow *window,
                                             GtkDirectionType arg1,
                                             gpointer user_data);
```

window : the object which received the signal.
arg1 :
user_data : user data set when the signal handler was connected.

The "set-focus" signal

```
void        user_function                   (GtkWindow *window,
                                             GtkWidget *widget,
                                             gpointer user_data);
```

window : the object which received the signal.
widget :
user_data : user data set when the signal handler was connected.

<< GtkMessageDialog **GtkWindowGroup >>**

GtkWindowGroup

GtkWindowGroup — Limit the effect of grabs

Synopsis

```
#include <gtk/gtk.h>

            GtkWindowGroup;
GtkWindowGroup* gtk_window_group_new         (void);
void        gtk_window_group_add_window      (GtkWindowGroup *window_group,
                                              GtkWindow *window);
void        gtk_window_group_remove_window   (GtkWindowGroup *window_group,
                                              GtkWindow *window);
```

Object Hierarchy

```
  GObject
   +----GtkWindowGroup
```

Description

Details

GtkWindowGroup

```
typedef struct _GtkWindowGroup GtkWindowGroup;
```

gtk_window_group_new ()

```
GtkWindowGroup* gtk_window_group_new        (void);
```

Creates a new GtkWindowGroup object. Grabs added with `gtk_grab_add()` only affect windows within the same GtkWindowGroup.

Returns : a new GtkWindowGroup.

gtk_window_group_add_window ()

```
void        gtk_window_group_add_window     (GtkWindowGroup *window_group,
                                             GtkWindow *window);
```

Adds a window to a GtkWindowGroup.

window_group : a GtkWindowGroup
window : the GtkWindow to add

gtk_window_group_remove_window ()

```
void        gtk_window_group_remove_window  (GtkWindowGroup *window_group,
                                             GtkWindow *window);
```

Removes a window from a GtkWindowGroup.

window_group : a GtkWindowGroup
window : the GtkWindow to remove

<< GtkWindow **GtkAboutDialog >>**

GtkAboutDialog

GtkAboutDialog — Display information about an application

Synopsis

```
#include <gtk/gtk.h>

            GtkAboutDialog;
GtkWidget*  gtk_about_dialog_new            (void);
G_CONST_RETURN gchar* gtk_about_dialog_get_name
                                            (GtkAboutDialog *about);
void        gtk_about_dialog_set_name       (GtkAboutDialog *about,
                                             const gchar *name);
G_CONST_RETURN gchar* gtk_about_dialog_get_version
                                            (GtkAboutDialog *about);
void        gtk_about_dialog_set_version    (GtkAboutDialog *about,
                                             const gchar *version);
G_CONST_RETURN gchar* gtk_about_dialog_get_copyright
                                            (GtkAboutDialog *about);
void        gtk_about_dialog_set_copyright  (GtkAboutDialog *about,
                                             const gchar *copyright);
G_CONST_RETURN gchar* gtk_about_dialog_get_comments
                                            (GtkAboutDialog *about);
void        gtk_about_dialog_set_comments   (GtkAboutDialog *about,
                                             const gchar *comments);
G_CONST_RETURN gchar* gtk_about_dialog_get_license
                                            (GtkAboutDialog *about);
void        gtk_about_dialog_set_license    (GtkAboutDialog *about,
                                             const gchar *license);
G_CONST_RETURN gchar* gtk_about_dialog_get_website
                                            (GtkAboutDialog *about);
void        gtk_about_dialog_set_website    (GtkAboutDialog *about,
                                             const gchar *website);
G_CONST_RETURN gchar* gtk_about_dialog_get_website_label
                                            (GtkAboutDialog *about);
void        gtk_about_dialog_set_website_label
                                            (GtkAboutDialog *about,
```

```
                                                const gchar *website_label);
gchar**     gtk_about_dialog_get_authors    (GtkAboutDialog *about);
void        gtk_about_dialog_set_authors    (GtkAboutDialog *about,
                                             gchar **authors);
gchar**     gtk_about_dialog_get_artists    (GtkAboutDialog *about);
void        gtk_about_dialog_set_artists    (GtkAboutDialog *about,
                                             gchar **artists);
gchar**     gtk_about_dialog_get_documenters
                                            (GtkAboutDialog *about);
void        gtk_about_dialog_set_documenters
                                            (GtkAboutDialog *about,
                                             gchar **documenters);
G_CONST_RETURN gchar* gtk_about_dialog_get_translator_credits
                                            (GtkAboutDialog *about);
void        gtk_about_dialog_set_translator_credits
                                            (GtkAboutDialog *about,
                                             const gchar *translator_credits);
GdkPixbuf*  gtk_about_dialog_get_logo       (GtkAboutDialog *about);
void        gtk_about_dialog_set_logo       (GtkAboutDialog *about,
                                             GdkPixbuf *logo);
void        (*GtkAboutDialogActivateLinkFunc)
                                            (GtkAboutDialog *about,
                                             const gchar *link,
                                             gpointer data);
GtkAboutDialogActivateLinkFunc gtk_about_dialog_set_email_hook
                                            (GtkAboutDialogActivateLinkFunc func,
                                             gpointer data,
                                             GDestroyNotify destroy);
GtkAboutDialogActivateLinkFunc gtk_about_dialog_set_url_hook
                                            (GtkAboutDialogActivateLinkFunc func,
                                             gpointer data,
                                             GDestroyNotify destroy);
void        gtk_show_about_dialog           (GtkWindow *parent,
                                             const gchar *first_property_name,
                                             ...);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkWidget
```

```
                +----GtkContainer
                      +----GtkBin
                            +----GtkWindow
                                  +----GtkDialog
                                        +----GtkAboutDialog
```

Implemented Interfaces

GtkAboutDialog implements AtkImplementorIface.

Properties

```
  "artists"              GStrv                : Read / Write
  "authors"              GStrv                : Read / Write
  "comments"             gchararray           : Read / Write
  "copyright"            gchararray           : Read / Write
  "documenters"          GStrv                : Read / Write
  "license"              gchararray           : Read / Write
  "logo"                 GdkPixbuf            : Read / Write
  "logo-icon-name"       gchararray           : Read / Write
  "name"                 gchararray           : Read / Write
  "translator-credits"   gchararray           : Read / Write
  "version"              gchararray           : Read / Write
  "website"              gchararray           : Read / Write
  "website-label"        gchararray           : Read / Write
```

Style Properties

```
  "link-color"           GdkColor             : Read
```

Description

The GtkAboutDialog offers a simple way to display information about a program like its logo, name, copyright, website and license. It is also possible to give credits to the authors, documenters, translators and artists who have worked on the program. An about dialog is typically opened when the user selects the `About` option from the `Help` menu. All parts of the dialog are optional.

About dialog often contain links and email addresses. GtkAboutDialog supports this by offering global hooks, which are called when the user clicks on a link or email address, see `gtk_about_dialog_set_email_hook()` and

gtk_about_dialog_set_url_hook().

Details

GtkAboutDialog

```
typedef struct _GtkAboutDialog GtkAboutDialog;
```

The GtkAboutDialog struct contains only private fields and should not be directly accessed.

---

gtk_about_dialog_new ()

```
GtkWidget*  gtk_about_dialog_new            (void);
```

Creates a new GtkAboutDialog.

Returns : a newly created GtkAboutDialog

Since 2.6

---

gtk_about_dialog_get_name ()

```
G_CONST_RETURN gchar* gtk_about_dialog_get_name
                                            (GtkAboutDialog *about);
```

Returns the program name displayed in the about dialog.

about : a GtkAboutDialog

Returns : The program name. The string is owned by the about dialog and must not be modified.

Since 2.6

---

gtk_about_dialog_set_name ()

```
void        gtk_about_dialog_set_name       (GtkAboutDialog *about,
                                             const gchar *name);
```

Sets the name to display in the about dialog. If this is not set, it defaults to g_get_application_name().

about : a GtkAboutDialog
name : the program name

Since 2.6

---

gtk_about_dialog_get_version ()

```
G_CONST_RETURN gchar* gtk_about_dialog_get_version
                                             (GtkAboutDialog *about);
```

Returns the version string.

about : a GtkAboutDialog
Returns : The version string. The string is owned by the about dialog and must not be modified.

Since 2.6

---

gtk_about_dialog_set_version ()

```
void        gtk_about_dialog_set_version    (GtkAboutDialog *about,
                                             const gchar *version);
```

Sets the version string to display in the about dialog.

about : a GtkAboutDialog
version : the version string

Since 2.6

---

gtk_about_dialog_get_copyright ()

```
G_CONST_RETURN gchar* gtk_about_dialog_get_copyright
                                                (GtkAboutDialog *about);
```

Returns the copyright string.

about : a GtkAboutDialog

Returns : The copyright string. The string is owned by the about dialog and must not be modified.

Since 2.6

---

gtk_about_dialog_set_copyright ()

```
void        gtk_about_dialog_set_copyright  (GtkAboutDialog *about,
                                             const gchar *copyright);
```

Sets the copyright string to display in the about dialog. This should be a short string of one or two lines.

about : a GtkAboutDialog

copyright : the copyright string

Since 2.6

---

gtk_about_dialog_get_comments ()

```
G_CONST_RETURN gchar* gtk_about_dialog_get_comments
                                                (GtkAboutDialog *about);
```

Returns the comments string.

about : a GtkAboutDialog

Returns : The comments. The string is owned by the about dialog and must not be modified.

Since 2.6

---

gtk_about_dialog_set_comments ()

```
void        gtk_about_dialog_set_comments   (GtkAboutDialog *about,
                                             const gchar *comments);
```

Sets the comments string to display in the about dialog. This should be a short string of one or two lines.

about : a GtkAboutDialog

comments : a comments string

Since 2.6

---

gtk_about_dialog_get_license ()

```
G_CONST_RETURN gchar* gtk_about_dialog_get_license
                                             (GtkAboutDialog *about);
```

Returns the license information.

about : a GtkAboutDialog

Returns : The license information. The string is owned by the about dialog and must not be modified.

Since 2.6

---

gtk_about_dialog_set_license ()

```
void        gtk_about_dialog_set_license    (GtkAboutDialog *about,
                                             const gchar *license);
```

Sets the license information to be displayed in the secondary license dialog. If *license* is NULL, the license button is hidden.

about : a GtkAboutDialog

license : the license information or NULL

Since 2.6

---

gtk_about_dialog_get_website ()

```
G_CONST_RETURN gchar* gtk_about_dialog_get_website
                                                (GtkAboutDialog *about);
```

Returns the website URL.

about : a GtkAboutDialog

Returns : The website URL. The string is owned by the about dialog and must not be modified.

Since 2.6

---

gtk_about_dialog_set_website ()

```
void        gtk_about_dialog_set_website    (GtkAboutDialog *about,
                                             const gchar *website);
```

Sets the URL to use for the website link.

about : a GtkAboutDialog

website : a URL string starting with "http://"

Since 2.6

---

gtk_about_dialog_get_website_label ()

```
G_CONST_RETURN gchar* gtk_about_dialog_get_website_label
                                                (GtkAboutDialog *about);
```

Returns the label used for the website link.

about : a GtkAboutDialog

Returns : The label used for the website link. The string is owned by the about dialog and must not be modified.

Since 2.6

---

gtk_about_dialog_set_website_label ()

```
void        gtk_about_dialog_set_website_label
                                            (GtkAboutDialog *about,
                                             const gchar *website_label);
```

Sets the label to be used for the website link. It defaults to the website URL.

about : a GtkAboutDialog

website_label : the label used for the website link

Since 2.6

---

gtk_about_dialog_get_authors ()

```
gchar**     gtk_about_dialog_get_authors    (GtkAboutDialog *about);
```

Returns the string which are displayed in the authors tab of the secondary credits dialog.

about : a GtkAboutDialog

Returns : A NULL-terminated string array containing the authors. The array is owned by the about dialog and must not be modified.

Since 2.6

---

gtk_about_dialog_set_authors ()

```
void        gtk_about_dialog_set_authors    (GtkAboutDialog *about,
                                             gchar **authors);
```

Sets the strings which are displayed in the authors tab of the secondary credits dialog.

about : a GtkAboutDialog
authors : a NULL-terminated array of strings

Since 2.6

---

gtk_about_dialog_get_artists ()

```
gchar**     gtk_about_dialog_get_artists    (GtkAboutDialog *about);
```

Returns the string which are displayed in the artists tab of the secondary credits dialog.

about : a GtkAboutDialog
Returns : A NULL-terminated string array containing the artists. The array is owned by the about dialog and must not be modified.

Since 2.6

---

gtk_about_dialog_set_artists ()

```
void        gtk_about_dialog_set_artists    (GtkAboutDialog *about,
                                             gchar **artists);
```

Sets the strings which are displayed in the artists tab of the secondary credits dialog.

about : a GtkAboutDialog
artists : a NULL-terminated array of strings

Since 2.6

---

gtk_about_dialog_get_documenters ()

```
gchar**     gtk_about_dialog_get_documenters
                                            (GtkAboutDialog *about);
```

Returns the string which are displayed in the documenters tab of the secondary credits dialog.

about : a GtkAboutDialog

Returns : A NULL-terminated string array containing the documenters. The array is owned by the about dialog and must not be modified.

Since 2.6

---

gtk_about_dialog_set_documenters ()

```
void        gtk_about_dialog_set_documenters
                                            (GtkAboutDialog *about,
                                             gchar **documenters);
```

Sets the strings which are displayed in the documenters tab of the secondary credits dialog.

about : a GtkAboutDialog

documenters : a NULL-terminated array of strings

Since 2.6

---

gtk_about_dialog_get_translator_credits ()

```
G_CONST_RETURN gchar* gtk_about_dialog_get_translator_credits
                                            (GtkAboutDialog *about);
```

Returns the translator credits string which is displayed in the translators tab of the secondary credits dialog.

about : a GtkAboutDialog

Returns : The translator credits string. The string is owned by the about dialog and must not be modified.

Since 2.6

---

gtk_about_dialog_set_translator_credits ()

```
void        gtk_about_dialog_set_translator_credits
                                            (GtkAboutDialog *about,
                                             const gchar *translator_credits);
```

Sets the translator credits string which is displayed in the translators tab of the secondary credits dialog.

The intended use for this string is to display the translator of the language which is currently used in the user interface. Using `gettext()`, a simple way to achieve that is to mark the string for translation:

```
gtk_about_dialog_set_translator_credits (about, _("translator-credits"));
```

It is a good idea to use the customary msgid "translator-credits" for this purpose, since translators will already know the purpose of that msgid, and since GtkAboutDialog will detect if "translator-credits" is untranslated and hide the tab.

about : a GtkAboutDialog

translator_credits : the translator credits

Since 2.6

---

gtk_about_dialog_get_logo ()

```
GdkPixbuf*  gtk_about_dialog_get_logo       (GtkAboutDialog *about);
```

Returns the pixbuf displayed as logo in the about dialog.

about : a GtkAboutDialog

Returns : the pixbuf displayed as logo. The pixbuf is owned by the about dialog. If you want to keep a reference to it, you have to call `g_object_ref()` on it.

Since 2.6

---

gtk_about_dialog_set_logo ()

```
void        gtk_about_dialog_set_logo       (GtkAboutDialog *about,
                                             GdkPixbuf *logo);
```

Sets the pixbuf to be displayed as logo in the about dialog. If it is NULL, the default window icon set with gtk_window_set_default_icon() will be used.

about : a GtkAboutDialog
logo : a GdkPixbuf, or NULL

Since 2.6

GtkAboutDialogActivateLinkFunc ()

```
void        (*GtkAboutDialogActivateLinkFunc)
                                            (GtkAboutDialog *about,
                                             const gchar *link,
                                             gpointer data);
```

The type of a function which is called when a URL or email link is activated.

about : the GtkAboutDialog in which the link was activated
link : the URL or email address to whiche the activated link points
data :

---

gtk_about_dialog_set_email_hook ()

```
GtkAboutDialogActivateLinkFunc gtk_about_dialog_set_email_hook
                                            (GtkAboutDialogActivateLinkFunc func,
                                             gpointer data,
                                             GDestroyNotify destroy);
```

Installs a global function to be called whenever the user activates an email link in an about dialog.

func : a function to call when an email link is activated.

data : data to pass to *func*
destroy : GDestroyNotify for *data*
Returns : the previous email hook.

Since 2.6

---

gtk_about_dialog_set_url_hook ()

```
GtkAboutDialogActivateLinkFunc gtk_about_dialog_set_url_hook
                                                    (GtkAboutDialogActivateLinkFunc func,
                                                     gpointer data,
                                                     GDestroyNotify destroy);
```

Installs a global function to be called whenever the user activates a URL link in an about dialog.

func : a function to call when a URL link is activated.
data : data to pass to *func*
destroy : GDestroyNotify for *data*
Returns : the previous URL hook.

Since 2.6

---

gtk_show_about_dialog ()

```
void        gtk_show_about_dialog           (GtkWindow *parent,
                                             const gchar *first_property_name,
                                             ...);
```

This is a convenience function for showing an application's about box.

parent : transient parent, or NULL for none
first_property_name : the name of the first property
... : value of first property, followed by more properties, NULL-terminated

Since 2.6

Properties

The "artists" property

"artists"	GStrv	: Read / Write

List of people who have contributed artwork to the program.

The "authors" property

"authors"	GStrv	: Read / Write

List of authors of the programs.

The "comments" property

"comments"	gchararray	: Read / Write

Comments about the program.

Default value: NULL

The "copyright" property

"copyright"	gchararray	: Read / Write

Copyright information for the program.

Default value: NULL

The "documenters" property

```
"documenters"          GStrv                : Read / Write
```

List of people documenting the program.

The "license" property

```
"license"              gchararray           : Read / Write
```

The license of the program.

Default value: NULL

The "logo" property

```
"logo"                 GdkPixbuf            : Read / Write
```

A logo for the about box. If this is not set, it defaults to gtk_window_get_default_icon_list().

The "logo-icon-name" property

```
"logo-icon-name"       gchararray           : Read / Write
```

A named icon to use as the logo for the about box.

Default value: NULL

The "name" property

```
"name"                 gchararray           : Read / Write
```

The name of the program. If this is not set, it defaults to g_get_application_name().

Default value: NULL

The "translator-credits" property

```
"translator-credits"   gchararray          : Read / Write
```

Credits to the translators. This string should be marked as translatable.

Default value: NULL

The "version" property

```
"version"              gchararray          : Read / Write
```

The version of the program.

Default value: NULL

The "website" property

```
"website"              gchararray          : Read / Write
```

The URL for the link to the website of the program.

Default value: NULL

The "website-label" property

```
"website-label"        gchararray          : Read / Write
```

The label for the link to the website of the program. If this is not set, it defaults to the URL.

Default value: NULL

Style Properties

The "link-color" style property

```
  "link-color"          GdkColor              : Read
```

Color of hyperlinks.

See Also

GTK_STOCK_ABOUT

<< GtkWindowGroup **Display Widgets >>**

Display Widgets

GtkAccelLabel - A label which displays an accelerator key on the right of the text
GtkImage - A widget displaying an image
GtkLabel - A widget that displays a small to medium amount of text
GtkProgressBar - A widget which indicates progress visually
GtkStatusbar - Report messages of minor importance to the user

<< GtkAboutDialog **GtkAccelLabel >>**

GtkAccelLabel

GtkAccelLabel — A label which displays an accelerator key on the right of the text

Synopsis

```
#include <gtk/gtk.h>


            GtkAccelLabel;
GtkWidget*  gtk_accel_label_new             (const gchar *string);
void        gtk_accel_label_set_accel_closure
                                            (GtkAccelLabel *accel_label,
                                             GClosure *accel_closure);
GtkWidget*  gtk_accel_label_get_accel_widget
                                            (GtkAccelLabel *accel_label);
void        gtk_accel_label_set_accel_widget
                                            (GtkAccelLabel *accel_label,
                                             GtkWidget *accel_widget);
guint       gtk_accel_label_get_accel_width (GtkAccelLabel *accel_label);
gboolean    gtk_accel_label_refetch         (GtkAccelLabel *accel_label);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkWidget
               +----GtkMisc
                     +----GtkLabel
                           +----GtkAccelLabel
```

Implemented Interfaces

GtkAccelLabel implements AtkImplementorIface.

Properties

```
"accel-closure"        GClosure              : Read / Write
"accel-widget"         GtkWidget             : Read / Write
```

Description

The GtkAccelLabel widget is a subclass of GtkLabel that also displays an accelerator key on the right of the label text, e.g. 'Ctl+S'. It is commonly used in menus to show the keyboard short-cuts for commands.

The accelerator key to display is not set explicitly. Instead, the GtkAccelLabel displays the accelerators which have been added to a particular widget. This widget is set by calling `gtk_accel_label_set_accel_widget()`.

For example, a GtkMenuItem widget may have an accelerator added to emit the "activate" signal when the 'Ctl+S' key combination is pressed. A GtkAccelLabel is created and added to the GtkMenuItem, and `gtk_accel_label_set_accel_widget()` is called with the GtkMenuItem as the second argument. The GtkAccelLabel will now display 'Ctl+S' after its label.

Note that creating a GtkMenuItem with `gtk_menu_item_new_with_label()` (or one of the similar functions for GtkCheckMenuItem and GtkRadioMenuItem) automatically adds a GtkAccelLabel to the GtkMenuItem and calls `gtk_accel_label_set_accel_widget()` to set it up for you.

A GtkAccelLabel will only display accelerators which have `GTK_ACCEL_VISIBLE` set (see GtkAccelFlags). A GtkAccelLabel can display multiple accelerators and even signal names, though it is almost always used to display just one accelerator key.

Example 1. Creating a simple menu item with an accelerator key.

```
GtkWidget *save_item;
GtkAccelGroup *accel_group;

/* Create a GtkAccelGroup and add it to the window. */
accel_group = gtk_accel_group_new ();
gtk_window_add_accel_group (GTK_WINDOW (window), accel_group);

/* Create the menu item using the convenience function. */
save_item = gtk_menu_item_new_with_label ("Save");
gtk_widget_show (save_item);
gtk_container_add (GTK_CONTAINER (menu), save_item);
```

```
  /* Now add the accelerator to the GtkMenuItem. Note that since we called
     gtk_menu_item_new_with_label() to create the GtkMenuItem the
     GtkAccelLabel is automatically set up to display the GtkMenuItem
     accelerators. We just need to make sure we use GTK_ACCEL_VISIBLE here. */
  gtk_widget_add_accelerator (save_item, "activate", accel_group,
                              GDK_s, GDK_CONTROL_MASK, GTK_ACCEL_VISIBLE);
```

Details

GtkAccelLabel

```
typedef struct _GtkAccelLabel GtkAccelLabel;
```

The GtkAccelLabel-struct struct contains private data only, and should be accessed using the functions below.

---

gtk_accel_label_new ()

```
GtkWidget*  gtk_accel_label_new             (const gchar *string);
```

Creates a new GtkAccelLabel.

string : the label string. Must be non-NULL.

Returns : a new GtkAccelLabel.

---

gtk_accel_label_set_accel_closure ()

```
void        gtk_accel_label_set_accel_closure
                                            (GtkAccelLabel *accel_label,
                                             GClosure *accel_closure);
```

Sets the closure to be monitored by this accelerator label. The closure must be connected to an accelerator group; see gtk_accel_group_connect().

accel_label : a GtkAccelLabel

accel_closure : the closure to monitor for accelerator changes.

---

gtk_accel_label_get_accel_widget ()

```
GtkWidget*  gtk_accel_label_get_accel_widget
                                            (GtkAccelLabel *accel_label);
```

Fetches the widget monitored by this accelerator label. See gtk_accel_label_set_accel_widget().

accel_label : a GtkAccelLabel

Returns : the object monitored by the accelerator label, or NULL.

---

gtk_accel_label_set_accel_widget ()

```
void        gtk_accel_label_set_accel_widget
                                            (GtkAccelLabel *accel_label,
                                             GtkWidget *accel_widget);
```

Sets the widget to be monitored by this accelerator label.

accel_label : a GtkAccelLabel

accel_widget : the widget to be monitored.

---

gtk_accel_label_get_accel_width ()

```
guint       gtk_accel_label_get_accel_width (GtkAccelLabel *accel_label);
```

Returns the width needed to display the accelerator key(s). This is used by menus to align all of the GtkMenuItem widgets, and shouldn't be needed by applications.

accel_label : a GtkAccelLabel.

Returns : the width needed to display the accelerator key(s).

---

gtk_accel_label_refetch ()

```
gboolean    gtk_accel_label_refetch         (GtkAccelLabel *accel_label);
```

Recreates the string representing the accelerator keys. This should not be needed since the string is automatically updated whenever accelerators are added or removed from the associated widget.

accel_label : a GtkAccelLabel.

Returns : always returns FALSE.

Properties

The "accel-closure" property

```
  "accel-closure"         GClosure              : Read / Write
```

The closure to be monitored for accelerator changes.

The "accel-widget" property

```
  "accel-widget"          GtkWidget             : Read / Write
```

The widget to be monitored for accelerator changes.

See Also

Keyboard Accelerators installing and using keyboard short-cuts.

GtkItemFactory an easier way to create menus.

GtkImage

GtkImage — A widget displaying an image

Synopsis

```
#include <gtk/gtk.h>

            GtkImage;
enum        GtkImageType;
void        gtk_image_get_icon_set          (GtkImage *image,
                                             GtkIconSet **icon_set,
                                             GtkIconSize *size);
void        gtk_image_get_image             (GtkImage *image,
                                             GdkImage **gdk_image,
                                             GdkBitmap **mask);
GdkPixbuf*  gtk_image_get_pixbuf            (GtkImage *image);
void        gtk_image_get_pixmap            (GtkImage *image,
                                             GdkPixmap **pixmap,
                                             GdkBitmap **mask);
void        gtk_image_get_stock             (GtkImage *image,
                                             gchar **stock_id,
                                             GtkIconSize *size);
GdkPixbufAnimation* gtk_image_get_animation (GtkImage *image);
void        gtk_image_get_icon_name         (GtkImage *image,
                                             G_CONST_RETURN gchar **icon_name,
                                             GtkIconSize *size);
GtkImageType gtk_image_get_storage_type     (GtkImage *image);
GtkWidget*  gtk_image_new_from_file         (const gchar *filename);
GtkWidget*  gtk_image_new_from_icon_set     (GtkIconSet *icon_set,
                                             GtkIconSize size);
GtkWidget*  gtk_image_new_from_image        (GdkImage *image,
                                             GdkBitmap *mask);
GtkWidget*  gtk_image_new_from_pixbuf       (GdkPixbuf *pixbuf);
GtkWidget*  gtk_image_new_from_pixmap       (GdkPixmap *pixmap,
                                             GdkBitmap *mask);
```

```
GtkWidget*  gtk_image_new_from_stock         (const gchar *stock_id,
                                              GtkIconSize size);
GtkWidget*  gtk_image_new_from_animation     (GdkPixbufAnimation *animation);
GtkWidget*  gtk_image_new_from_icon_name     (const gchar *icon_name,
                                              GtkIconSize size);
void        gtk_image_set_from_file          (GtkImage *image,
                                              const gchar *filename);
void        gtk_image_set_from_icon_set      (GtkImage *image,
                                              GtkIconSet *icon_set,
                                              GtkIconSize size);
void        gtk_image_set_from_image         (GtkImage *image,
                                              GdkImage *gdk_image,
                                              GdkBitmap *mask);
void        gtk_image_set_from_pixbuf        (GtkImage *image,
                                              GdkPixbuf *pixbuf);
void        gtk_image_set_from_pixmap        (GtkImage *image,
                                              GdkPixmap *pixmap,
                                              GdkBitmap *mask);
void        gtk_image_set_from_stock         (GtkImage *image,
                                              const gchar *stock_id,
                                              GtkIconSize size);
void        gtk_image_set_from_animation     (GtkImage *image,
                                              GdkPixbufAnimation *animation);
void        gtk_image_set_from_icon_name     (GtkImage *image,
                                              const gchar *icon_name,
                                              GtkIconSize size);
GtkWidget*  gtk_image_new                    (void);
void        gtk_image_set                    (GtkImage *image,
                                              GdkImage *val,
                                              GdkBitmap *mask);
void        gtk_image_get                    (GtkImage *image,
                                              GdkImage **val,
                                              GdkBitmap **mask);
void        gtk_image_set_pixel_size         (GtkImage *image,
                                              gint pixel_size);
gint        gtk_image_get_pixel_size         (GtkImage *image);
```

Object Hierarchy

```
GObject
 +----GtkObject
       +----GtkWidget
             +----GtkMisc
                   +----GtkImage
```

Implemented Interfaces

GtkImage implements AtkImplementorIface.

Properties

```
"file"                gchararray            : Write
"icon-name"           gchararray            : Read / Write
"icon-set"            GtkIconSet            : Read / Write
"icon-size"           gint                  : Read / Write
"image"               GdkImage              : Read / Write
"mask"                GdkPixmap             : Read / Write
"pixbuf"              GdkPixbuf             : Read / Write
"pixbuf-animation"    GdkPixbufAnimation    : Read / Write
"pixel-size"          gint                  : Read / Write
"pixmap"              GdkPixmap             : Read / Write
"stock"               gchararray            : Read / Write
"storage-type"        GtkImageType          : Read
```

Description

The GtkImage widget displays an image. Various kinds of object can be displayed as an image; most typically, you would load a GdkPixbuf ("pixel buffer") from a file, and then display that. There's a convenience function to do this, `gtk_image_new_from_file()`, used as follows:

```
GtkWidget *image;
image = gtk_image_new_from_file ("myfile.png");
```

If the file isn't loaded successfully, the image will contain a "broken image" icon similar to that used in many web browsers. If you want to handle errors in loading the file yourself, for example by displaying an error message, then load the image with `gdk_pixbuf_new_from_file()`, then create the GtkImage with `gtk_image_new_from_pixbuf()`.

The image file may contain an animation, if so the GtkImage will display an animation (GdkPixbufAnimation) instead of a static image.

GtkImage is a subclass of GtkMisc, which implies that you can align it (center, left, right) and add padding to it, using GtkMisc methods.

GtkImage is a "no window" widget (has no GdkWindow of its own), so by default does not receive events. If you want to receive events on the image, such as button clicks, place the image inside a GtkEventBox, then connect to the event signals on the event box.

Example 2. Handling button press events on a GtkImage.

```
static gboolean
button_press_callback (GtkWidget      *event_box,
                       GdkEventButton *event,
                       gpointer        data)
{
  g_print ("Event box clicked at coordinates %f,%f\n",
           event->x, event->y);

  /* Returning TRUE means we handled the event, so the signal
   * emission should be stopped (don't call any further
   * callbacks that may be connected). Return FALSE
   * to continue invoking callbacks.
   */
  return TRUE;
}

static GtkWidget*
create_image (void)
{
  GtkWidget *image;
  GtkWidget *event_box;

  image = gtk_image_new_from_file ("myfile.png");

  event_box = gtk_event_box_new ();

  gtk_container_add (GTK_CONTAINER (event_box), image);

  g_signal_connect (G_OBJECT (event_box),
                    "button_press_event",
                    G_CALLBACK (button_press_callback),
                    image);

  return image;
}
```

When handling events on the event box, keep in mind that coordinates in the image may be different from event box coordinates due to the alignment and padding settings on the image (see GtkMisc). The simplest way to solve this is to set the alignment to 0.0 (left/top), and set the padding to zero. Then the origin of the image will be the same as the origin of the event box.

Sometimes an application will want to avoid depending on external data files, such as image files. GTK+ comes with a program to avoid this, called gdk-pixbuf-csource. This program allows you to convert an image into a C variable declaration, which can then be loaded into a GdkPixbuf using `gdk_pixbuf_new_from_inline()`.

Details

GtkImage

```
typedef struct _GtkImage GtkImage;
```

This struct contain private data only and should be accessed by the functions below.

---

enum GtkImageType

```
typedef enum
{
  GTK_IMAGE_EMPTY,
  GTK_IMAGE_PIXMAP,
  GTK_IMAGE_IMAGE,
  GTK_IMAGE_PIXBUF,
  GTK_IMAGE_STOCK,
  GTK_IMAGE_ICON_SET,
  GTK_IMAGE_ANIMATION,
  GTK_IMAGE_ICON_NAME
} GtkImageType;
```

Describes the image data representation used by a GtkImage. If you want to get the image from the widget, you can only get the currently-stored representation. e.g. if the `gtk_image_get_storage_type()` returns GTK_IMAGE_PIXBUF, then you can call `gtk_image_get_pixbuf()` but not `gtk_image_get_stock()`. For empty images, you can request any storage type (call any of the "get" functions), but they will all return `NULL` values.

`GTK_IMAGE_EMPTY`	there is no image displayed by the widget
`GTK_IMAGE_PIXMAP`	the widget contains a GdkPixmap

`GTK_IMAGE_IMAGE`	the widget contains a GdkImage
`GTK_IMAGE_PIXBUF`	the widget contains a GdkPixbuf
`GTK_IMAGE_STOCK`	the widget contains a stock icon name (see Stock Items(3))
`GTK_IMAGE_ICON_SET`	the widget contains a GtkIconSet
`GTK_IMAGE_ANIMATION`	the widget contains a GdkPixbufAnimation
`GTK_IMAGE_ICON_NAME`	

---

gtk_image_get_icon_set ()

```
void        gtk_image_get_icon_set          (GtkImage *image,
                                             GtkIconSet **icon_set,
                                             GtkIconSize *size);
```

Gets the icon set and size being displayed by the GtkImage. The storage type of the image must be `GTK_IMAGE_EMPTY` or `GTK_IMAGE_ICON_SET` (see `gtk_image_get_storage_type()`).

`image` :	a GtkImage
`icon_set` :	location to store a GtkIconSet
`size` :	location to store a stock icon size

---

gtk_image_get_image ()

```
void        gtk_image_get_image             (GtkImage *image,
                                             GdkImage **gdk_image,
                                             GdkBitmap **mask);
```

Gets the GdkImage and mask being displayed by the GtkImage. The storage type of the image must be `GTK_IMAGE_EMPTY` or `GTK_IMAGE_IMAGE` (see `gtk_image_get_storage_type()`). The caller of this function does not own a reference to the returned image and mask.

`image` :	a GtkImage
`gdk_image` :	return location for a GtkImage
`mask` :	return location for a GdkBitmap

---

gtk_image_get_pixbuf ()

```
GdkPixbuf*  gtk_image_get_pixbuf            (GtkImage *image);
```

Gets the GdkPixbuf being displayed by the GtkImage. The storage type of the image must be `GTK_IMAGE_EMPTY` or `GTK_IMAGE_PIXBUF` (see `gtk_image_get_storage_type()`). The caller of this function does not own a reference to the returned pixbuf.

image : a GtkImage

Returns : the displayed pixbuf, or `NULL` if the image is empty

---

gtk_image_get_pixmap ()

```
void        gtk_image_get_pixmap            (GtkImage *image,
                                             GdkPixmap **pixmap,
                                             GdkBitmap **mask);
```

Gets the pixmap and mask being displayed by the GtkImage. The storage type of the image must be `GTK_IMAGE_EMPTY` or `GTK_IMAGE_PIXMAP` (see `gtk_image_get_storage_type()`). The caller of this function does not own a reference to the returned pixmap and mask.

image : a GtkImage

pixmap : location to store the pixmap, or `NULL`

mask : location to store the mask, or `NULL`

---

gtk_image_get_stock ()

```
void        gtk_image_get_stock             (GtkImage *image,
                                             gchar **stock_id,
                                             GtkIconSize *size);
```

Gets the stock icon name and size being displayed by the GtkImage. The storage type of the image must be `GTK_IMAGE_EMPTY` or `GTK_IMAGE_STOCK` (see `gtk_image_get_storage_type()`). The returned string is owned by the GtkImage and should not be freed.

image : a GtkImage
stock_id : place to store a stock icon name
size : place to store a stock icon size

---

gtk_image_get_animation ()

```
GdkPixbufAnimation* gtk_image_get_animation (GtkImage *image);
```

Gets the GdkPixbufAnimation being displayed by the GtkImage. The storage type of the image must be GTK_IMAGE_EMPTY or GTK_IMAGE_ANIMATION (see gtk_image_get_storage_type()). The caller of this function does not own a reference to the returned animation.

image : a GtkImage
Returns : the displayed animation, or NULL if the image is empty

---

gtk_image_get_icon_name ()

```
void        gtk_image_get_icon_name         (GtkImage *image,
                                             G_CONST_RETURN gchar **icon_name,
                                             GtkIconSize *size);
```

Gets the icon name and size being displayed by the GtkImage. The storage type of the image must be GTK_IMAGE_EMPTY or GTK_IMAGE_ICON_NAME (see gtk_image_get_storage_type()). The returned string is owned by the GtkImage and should not be freed.

image : a GtkImage
icon_name : place to store an icon name
size : place to store an icon size

Since 2.6

---

gtk_image_get_storage_type ()

```
GtkImageType gtk_image_get_storage_type     (GtkImage *image);
```

Gets the type of representation being used by the GtkImage to store image data. If the GtkImage has no image data, the return value will be GTK_IMAGE_EMPTY.

image : a GtkImage

Returns : image representation being used

---

gtk_image_new_from_file ()

```
GtkWidget*  gtk_image_new_from_file         (const gchar *filename);
```

Creates a new GtkImage displaying the file *filename*. If the file isn't found or can't be loaded, the resulting GtkImage will display a "broken image" icon. This function never returns NULL, it always returns a valid GtkImage widget.

If the file contains an animation, the image will contain an animation.

If you need to detect failures to load the file, use gdk_pixbuf_new_from_file() to load the file yourself, then create the GtkImage from the pixbuf. (Or for animations, use gdk_pixbuf_animation_new_from_file()).

The storage type (gtk_image_get_storage_type()) of the returned image is not defined, it will be whatever is appropriate for displaying the file.

filename : a filename

Returns : a new GtkImage

---

gtk_image_new_from_icon_set ()

```
GtkWidget*  gtk_image_new_from_icon_set     (GtkIconSet *icon_set,
                                             GtkIconSize size);
```

Creates a GtkImage displaying an icon set. Sample stock sizes are GTK_ICON_SIZE_MENU, GTK_ICON_SIZE_SMALL_TOOLBAR. Instead of using this function, usually it's better to create a GtkIconFactory, put your icon sets in the icon factory, add the icon factory to the list of default factories with gtk_icon_factory_add_default(), and then use gtk_image_new_from_stock(). This will allow themes to override the icon you ship with your application.

The GtkImage does not assume a reference to the icon set; you still need to unref it if you own references. GtkImage will add its own reference rather than adopting yours.

icon_set : a GtkIconSet
size : a stock icon size
Returns : a new GtkImage

---

gtk_image_new_from_image ()

```
GtkWidget*  gtk_image_new_from_image        (GdkImage *image,
                                             GdkBitmap *mask);
```

Creates a GtkImage widget displaying a *image* with a *mask*. A GdkImage is a client-side image buffer in the pixel format of the current display. The GtkImage does not assume a reference to the image or mask; you still need to unref them if you own references. GtkImage will add its own reference rather than adopting yours.

image : a GdkImage, or NULL
mask : a GdkBitmap, or NULL
Returns : a new GtkImage

---

gtk_image_new_from_pixbuf ()

```
GtkWidget*  gtk_image_new_from_pixbuf       (GdkPixbuf *pixbuf);
```

Creates a new GtkImage displaying *pixbuf*. The GtkImage does not assume a reference to the pixbuf; you still need to unref it if you own references. GtkImage will add its own reference rather than adopting yours.

Note that this function just creates an GtkImage from the pixbuf. The GtkImage created will not react to state changes. Should you want that, you should use gtk_image_new_from_icon_set().

pixbuf : a GdkPixbuf, or NULL
Returns : a new GtkImage

---

gtk_image_new_from_pixmap ()

```
GtkWidget*  gtk_image_new_from_pixmap       (GdkPixmap *pixmap,
                                             GdkBitmap *mask);
```

Creates a GtkImage widget displaying *pixmap* with a *mask*. A GdkPixmap is a server-side image buffer in the pixel format of the current display. The GtkImage does not assume a reference to the pixmap or mask; you still need to unref them if you own references. GtkImage will add its own reference rather than adopting yours.

pixmap : a GdkPixmap, or NULL
mask : a GdkBitmap, or NULL
Returns : a new GtkImage

---

gtk_image_new_from_stock ()

```
GtkWidget*  gtk_image_new_from_stock        (const gchar *stock_id,
                                             GtkIconSize size);
```

Creates a GtkImage displaying a stock icon. Sample stock icon names are GTK_STOCK_OPEN, GTK_STOCK_EXIT. Sample stock sizes are GTK_ICON_SIZE_MENU, GTK_ICON_SIZE_SMALL_TOOLBAR. If the stock icon name isn't known, a "broken image" icon will be displayed instead. You can register your own stock icon names, see `gtk_icon_factory_add_default()` and `gtk_icon_factory_add()`.

stock_id : a stock icon name
size : a stock icon size
Returns : a new GtkImage displaying the stock icon

---

gtk_image_new_from_animation ()

```
GtkWidget*  gtk_image_new_from_animation    (GdkPixbufAnimation *animation);
```

Creates a GtkImage displaying the given animation. The GtkImage does not assume a reference to the animation; you still need to unref it if you own references. GtkImage will add its own reference rather than adopting yours.

animation : an animation
Returns : a new GtkImage widget

---

gtk_image_new_from_icon_name ()

```
GtkWidget*  gtk_image_new_from_icon_name    (const gchar *icon_name,
                                             GtkIconSize size);
```

Creates a GtkImage displaying an icon from the current icon theme. If the icon name isn't known, a "broken image" icon will be displayed instead. If the current icon theme is changed, the icon will be updated appropriately.

icon_name : an icon name
size : a stock icon size
Returns : a new GtkImage displaying the themed icon

Since 2.6

---

gtk_image_set_from_file ()

```
void        gtk_image_set_from_file         (GtkImage *image,
                                             const gchar *filename);
```

See `gtk_image_new_from_file()` for details.

image : a GtkImage
filename : a filename or `NULL`

---

gtk_image_set_from_icon_set ()

```
void        gtk_image_set_from_icon_set     (GtkImage *image,
                                             GtkIconSet *icon_set,
                                             GtkIconSize size);
```

See `gtk_image_new_from_icon_set()` for details.

image : a GtkImage

icon_set : a GtkIconSet
size : a stock icon size

---

gtk_image_set_from_image ()

```
void        gtk_image_set_from_image        (GtkImage *image,
                                             GdkImage *gdk_image,
                                             GdkBitmap *mask);
```

See `gtk_image_new_from_image()` for details.

image : a GtkImage
gdk_image : a GdkImage or `NULL`
mask : a GdkBitmap or `NULL`

---

gtk_image_set_from_pixbuf ()

```
void        gtk_image_set_from_pixbuf       (GtkImage *image,
                                             GdkPixbuf *pixbuf);
```

See `gtk_image_new_from_pixbuf()` for details.

image : a GtkImage
pixbuf : a GdkPixbuf or `NULL`

---

gtk_image_set_from_pixmap ()

```
void        gtk_image_set_from_pixmap       (GtkImage *image,
                                             GdkPixmap *pixmap,
                                             GdkBitmap *mask);
```

See `gtk_image_new_from_pixmap()` for details.

image : a GtkImage
pixmap : a GdkPixmap or NULL
mask : a GdkBitmap or NULL

---

gtk_image_set_from_stock ()

```
void        gtk_image_set_from_stock        (GtkImage *image,
                                             const gchar *stock_id,
                                             GtkIconSize size);
```

See gtk_image_new_from_stock() for details.

image : a GtkImage
stock_id : a stock icon name
size : a stock icon size

---

gtk_image_set_from_animation ()

```
void        gtk_image_set_from_animation    (GtkImage *image,
                                             GdkPixbufAnimation *animation);
```

Causes the GtkImage to display the given animation (or display nothing, if you set the animation to NULL).

image : a GtkImage
animation : the GdkPixbufAnimation

---

gtk_image_set_from_icon_name ()

```
void        gtk_image_set_from_icon_name    (GtkImage *image,
                                             const gchar *icon_name,
                                             GtkIconSize size);
```

See gtk_image_new_from_icon_name() for details.

image : a GtkImage
icon_name : an icon name
size : an icon size

Since 2.6

---

gtk_image_new ()

```
GtkWidget*  gtk_image_new                   (void);
```

Creates a new empty GtkImage widget.

Returns : a newly created GtkImage widget.

---

gtk_image_set ()

```
void        gtk_image_set                   (GtkImage *image,
                                             GdkImage *val,
                                             GdkBitmap *mask);
```

Warning

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

Sets the GtkImage.

image : a GdkPixmap
val :
mask : a GdkBitmap that indicates which parts of the image should be transparent.

---

gtk_image_get ()

```
void        gtk_image_get                   (GtkImage *image,
```

```
                                                GdkImage **val,
                                                GdkBitmap **mask);
```

Warning

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

Gets the GtkImage.

image : a GdkPixmap
val :
mask : a GdkBitmap that indicates which parts of the image should be transparent.

---

gtk_image_set_pixel_size ()

```
void        gtk_image_set_pixel_size        (GtkImage *image,
                                             gint pixel_size);
```

Sets the pixel size to use for named icons. If the pixel size is set to a value != -1, it is used instead of the icon size set by `gtk_image_set_from_icon_name()`.

image : a GtkImage
pixel_size : the new pixel size

Since 2.6

---

gtk_image_get_pixel_size ()

```
gint        gtk_image_get_pixel_size        (GtkImage *image);
```

Gets the pixel size used for named icons.

image : a GtkImage
Returns : the pixel size used for named icons.

Since 2.6

Properties

The "file" property

```
"file"                gchararray            : Write
```

Filename to load and display.

Default value: NULL

The "icon-name" property

```
"icon-name"           gchararray            : Read / Write
```

The name of the icon in the icon theme. If the icon theme is changed, the image will be updated automatically.

Default value: NULL

Since 2.6

The "icon-set" property

```
"icon-set"            GtkIconSet            : Read / Write
```

Icon set to display.

The "icon-size" property

```
"icon-size"           gint                  : Read / Write
```

Symbolic size to use for stock icon, icon set or named icon.

Allowed values: >= 0

Default value: 4

The "image" property

```
  "image"                GdkImage              : Read / Write
```

A GdkImage to display.

The "mask" property

```
  "mask"                 GdkPixmap             : Read / Write
```

Mask bitmap to use with GdkImage or GdkPixmap.

The "pixbuf" property

```
  "pixbuf"               GdkPixbuf             : Read / Write
```

A GdkPixbuf to display.

The "pixbuf-animation" property

```
  "pixbuf-animation"     GdkPixbufAnimation    : Read / Write
```

GdkPixbufAnimation to display.

The "pixel-size" property

"pixel-size"	gint	: Read / Write

The :pixel-size property can be used to specify a fixed size overriding the :icon-size property for images of type GTK_IMAGE_ICON_NAME.

Allowed values: >= -1

Default value: -1

Since 2.6

The "pixmap" property

"pixmap"	GdkPixmap	: Read / Write

A GdkPixmap to display.

The "stock" property

"stock"	gchararray	: Read / Write

Stock ID for a stock image to display.

Default value: NULL

The "storage-type" property

"storage-type"	GtkImageType	: Read

The representation being used for image data.

Default value: GTK_IMAGE_EMPTY

See Also

GdkPixbuf

<< GtkAccelLabel **GtkLabel >>**

GtkLabel

GtkLabel — A widget that displays a small to medium amount of text

Synopsis

```
#include <gtk/gtk.h>


            GtkLabel;
GtkWidget*  gtk_label_new                   (const char *str);
void        gtk_label_set_text              (GtkLabel *label,
                                             const char *str);
void        gtk_label_set_attributes        (GtkLabel *label,
                                             PangoAttrList *attrs);
void        gtk_label_set_markup            (GtkLabel *label,
                                             const gchar *str);
void        gtk_label_set_markup_with_mnemonic
                                            (GtkLabel *label,
                                             const gchar *str);
void        gtk_label_set_pattern           (GtkLabel *label,
                                             const gchar *pattern);
void        gtk_label_set_justify           (GtkLabel *label,
                                             GtkJustification jtype);
void        gtk_label_set_ellipsize         (GtkLabel *label,
                                             PangoEllipsizeMode mode);
void        gtk_label_set_width_chars       (GtkLabel *label,
                                             gint n_chars);
void        gtk_label_get                   (GtkLabel *label,
                                             char **str);
guint       gtk_label_parse_uline           (GtkLabel *label,
                                             const gchar *string);
void        gtk_label_set_line_wrap         (GtkLabel *label,
                                             gboolean wrap);
#define     gtk_label_set
```

```
void        gtk_label_get_layout_offsets    (GtkLabel *label,
                                             gint *x,
                                             gint *y);
guint       gtk_label_get_mnemonic_keyval   (GtkLabel *label);
gboolean    gtk_label_get_selectable        (GtkLabel *label);
G_CONST_RETURN gchar* gtk_label_get_text    (GtkLabel *label);
GtkWidget*  gtk_label_new_with_mnemonic     (const char *str);
void        gtk_label_select_region         (GtkLabel *label,
                                             gint start_offset,
                                             gint end_offset);
void        gtk_label_set_mnemonic_widget   (GtkLabel *label,
                                             GtkWidget *widget);
void        gtk_label_set_selectable        (GtkLabel *label,
                                             gboolean setting);
void        gtk_label_set_text_with_mnemonic
                                            (GtkLabel *label,
                                             const gchar *str);
PangoAttrList* gtk_label_get_attributes     (GtkLabel *label);
GtkJustification gtk_label_get_justify      (GtkLabel *label);
PangoEllipsizeMode gtk_label_get_ellipsize  (GtkLabel *label);
gint        gtk_label_get_width_chars       (GtkLabel *label);
G_CONST_RETURN gchar* gtk_label_get_label   (GtkLabel *label);
PangoLayout* gtk_label_get_layout           (GtkLabel *label);
gboolean    gtk_label_get_line_wrap         (GtkLabel *label);
GtkWidget*  gtk_label_get_mnemonic_widget   (GtkLabel *label);
gboolean    gtk_label_get_selection_bounds  (GtkLabel *label,
                                             gint *start,
                                             gint *end);
gboolean    gtk_label_get_use_markup        (GtkLabel *label);
gboolean    gtk_label_get_use_underline     (GtkLabel *label);
void        gtk_label_set_label             (GtkLabel *label,
                                             const gchar *str);
void        gtk_label_set_use_markup        (GtkLabel *label,
                                             gboolean setting);
void        gtk_label_set_use_underline     (GtkLabel *label,
                                             gboolean setting);
```

Object Hierarchy

```
GObject
 +----GtkObject
       +----GtkWidget
             +----GtkMisc
                   +----GtkLabel
                         +----GtkAccelLabel
                         +----GtkTipsQuery
```

Implemented Interfaces

GtkLabel implements AtkImplementorIface.

Properties

```
"attributes"           PangoAttrList         : Read / Write
"cursor-position"      gint                  : Read
"ellipsize"            PangoEllipsizeMode    : Read / Write
"justify"              GtkJustification      : Read / Write
"label"                gchararray            : Read / Write
"mnemonic-keyval"      guint                 : Read
"mnemonic-widget"      GtkWidget             : Read / Write
"pattern"              gchararray            : Write
"selectable"           gboolean              : Read / Write
"selection-bound"      gint                  : Read
"use-markup"           gboolean              : Read / Write
"use-underline"        gboolean              : Read / Write
"width-chars"          gint                  : Read / Write
"wrap"                 gboolean              : Read / Write
```

Signal Prototypes

```
"copy-clipboard"
            void        user_function      (GtkLabel *label,
                                            gpointer user_data);
```

```
"move-cursor"
            void        user_function      (GtkLabel *label,
                                            GtkMovementStep arg1,
                                            gint arg2,
                                            gboolean arg3,
                                            gpointer user_data);
"populate-popup"
            void        user_function      (GtkLabel *label,
                                            GtkMenu *arg1,
                                            gpointer user_data);
```

Description

The GtkLabel widget displays a small amount of text. As the name implies, most labels are used to label another widget such as a GtkButton, a GtkMenuItem, or a GtkOptionMenu.

Mnemonics

Labels may contain *mnemonics*. Mnemonics are underlined characters in the label, used for keyboard navigation. Mnemonics are created by providing a string with an underscore before the mnemonic character, such as `"_File"`, to the functions `gtk_label_new_with_mnemonic()` or `gtk_label_set_text_with_mnemonic()`.

Mnemonics automatically activate any activatable widget the label is inside, such as a GtkButton; if the label is not inside the mnemonic's target widget, you have to tell the label about the target using `gtk_label_set_mnemonic_widget()`. Here's a simple example where the label is inside a button:

```
/* Pressing Alt+H will activate this button */
button = gtk_button_new ();
label = gtk_label_new_with_mnemonic ("_Hello");
gtk_container_add (GTK_CONTAINER (button), label);
```

There's a convenience function to create buttons with a mnemonic label already inside:

```
/* Pressing Alt+H will activate this button */
button = gtk_button_new_with_mnemonic ("_Hello");
```

To create a mnemonic for a widget alongside the label, such as a GtkEntry, you have to point the label at the

entry with gtk_label_set_mnemonic_widget():

```
/* Pressing Alt+H will focus the entry */
entry = gtk_entry_new ();
label = gtk_label_new_with_mnemonic ("_Hello");
gtk_label_set_mnemonic_widget (GTK_LABEL (label), entry);
```

Markup (styled text)

To make it easy to format text in a label (changing colors, fonts, etc.), label text can be provided in a simple markup format. Here's how to create a label with a small font:

```
label = gtk_label_new (NULL);
gtk_label_set_markup (GTK_LABEL (label), "<small>Small text</small>");
```

(See complete documentation of available tags in the Pango manual.)

The markup passed to gtk_label_set_markup() must be valid; for example, literal </>/& characters must be escaped as <, >, and &. If you pass text obtained from the user, file, or a network to gtk_label_set_markup(), you'll want to escape it with g_markup_escape_text() or g_markup_printf_escaped().

Markup strings are just a convenient way to set the PangoAttrList on a label; gtk_label_set_attributes() may be a simpler way to set attributes in some cases. Be careful though; PangoAttrList tends to cause internationalization problems, unless you're applying attributes to the entire string (i.e. unless you set the range of each attribute to [0, G_MAXINT)). The reason is that specifying the start_index and end_index for a PangoAttribute requires knowledge of the exact string being displayed, so translations will cause problems.

Selectable labels

Labels can be made selectable with gtk_label_set_selectable(). Selectable labels allow the user to copy the label contents to the clipboard. Only labels that contain useful-to-copy information — such as error messages — should be made selectable.

Text layout

A label can contain any number of paragraphs, but will have performance problems if it contains more than a small number. Paragraphs are separated by newlines or other paragraph separators understood by Pango.

Labels can automatically wrap text if you call `gtk_label_set_line_wrap()`.

`gtk_label_set_justify()` sets how the lines in a label align with one another. If you want to set how the label as a whole aligns in its available space, see `gtk_misc_set_alignment()`.

Details

GtkLabel

```
typedef struct _GtkLabel GtkLabel;
```

This should not be accessed directly. Use the accessor functions as described below.

---

gtk_label_new ()

```
GtkWidget*  gtk_label_new                   (const char *str);
```

Creates a new label with the given text inside it. You can pass `NULL` to get an empty label widget.

str : The text of the label

Returns : the new GtkLabel

---

gtk_label_set_text ()

```
void        gtk_label_set_text              (GtkLabel *label,
                                             const char *str);
```

Sets the text within the GtkLabel widget. It overwrites any text that was there before.

This will also clear any previously set mnemonic accelerators.

label : a GtkLabel

str : The text you want to set.

---

gtk_label_set_attributes ()

```
void        gtk_label_set_attributes        (GtkLabel *label,
                                             PangoAttrList *attrs);
```

Sets a PangoAttrList; the attributes in the list are applied to the label text. The attributes set with this function will be ignored if the "use_underline" property or the "use_markup" property is TRUE.

label : a GtkLabel

attrs : a PangoAttrList

---

gtk_label_set_markup ()

```
void        gtk_label_set_markup            (GtkLabel *label,
                                             const gchar *str);
```

Parses *str* which is marked up with the Pango text markup language, setting the label's text and attribute list based on the parse results. If the *str* is external data, you may need to escape it with g_markup_escape_text() or g_markup_printf_escaped():

```
char *markup;

markup = g_markup_printf_escaped ("<span style=\"italic\">%s</span>", str);
gtk_label_set_markup (GTK_LABEL (label), markup);
g_free (markup);
```

label : a GtkLabel

str : a markup string (see Pango markup format)

gtk_label_set_markup_with_mnemonic ()

```
void        gtk_label_set_markup_with_mnemonic
                                            (GtkLabel *label,
                                             const gchar *str);
```

Parses *str* which is marked up with the Pango text markup language, setting the label's text and attribute list based on the parse results. If characters in *str* are preceded by an underscore, they are underlined indicating that they represent a keyboard accelerator called a mnemonic.

The mnemonic key can be used to activate another widget, chosen automatically, or explicitly using `gtk_label_set_mnemonic_widget()`.

label : a GtkLabel

str : a markup string (see Pango markup format)

gtk_label_set_pattern ()

```
void        gtk_label_set_pattern           (GtkLabel *label,
                                             const gchar *pattern);
```

The pattern of underlines you want under the existing text within the GtkLabel widget. For example if the current text of the label says "FooBarBaz" passing a pattern of "___ ___" will underline "Foo" and "Baz" but not "Bar".

label : The GtkLabel you want to set the pattern to.

pattern : The pattern as described above.

gtk_label_set_justify ()

```
void        gtk_label_set_justify           (GtkLabel *label,
                                             GtkJustification jtype);
```

Sets the alignment of the lines in the text of the label relative to each other. GTK_JUSTIFY_LEFT is the default value when the widget is first created with gtk_label_new(). If you instead want to set the alignment of the label as a whole, use gtk_misc_set_alignment() instead. gtk_label_set_justify() has no effect on labels containing only a single line.

label : a GtkLabel

jtype : a GtkJustification

---

gtk_label_set_ellipsize ()

```
void        gtk_label_set_ellipsize         (GtkLabel *label,
                                             PangoEllipsizeMode mode);
```

Sets the mode used to ellipsize (add an ellipsis: "...") to the text if there is not enough space to render the entire string.

label : a GtkLabel

mode : a PangoEllipsizeMode

Since 2.6

---

gtk_label_set_width_chars ()

```
void        gtk_label_set_width_chars       (GtkLabel *label,
                                             gint n_chars);
```

Sets the desired width in characters of *label* to *n_chars*.

label : a GtkLabel

n_chars : the new desired width, in characters.

Since 2.6

gtk_label_get ()

```
void        gtk_label_get                       (GtkLabel *label,
                                                 char **str);
```

Warning

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

Gets the current string of text within the GtkLabel and writes it to the given *`str`* argument. It does not make a copy of this string so you must not write to it.

`label` : The GtkLabel widget you want to get the text from.

`str` : The reference to the pointer you want to point to the text.

---

gtk_label_parse_uline ()

```
guint       gtk_label_parse_uline               (GtkLabel *label,
                                                 const gchar *string);
```

Warning

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

Parses the given string for underscores and converts the next character to an underlined character. The last character that was underlined will have its lower-cased accelerator keyval returned (i.e. "_File" would return the keyval for "f". This is probably only used within the Gtk+ library itself for menu items and such.

`label` : The GtkLabel you want to affect.

`string` : The string you want to parse for underlines.

Returns : The lowercase keyval of the last character underlined.

---

gtk_label_set_line_wrap ()

```
void        gtk_label_set_line_wrap         (GtkLabel *label,
                                             gboolean wrap);
```

Toggles line wrapping within the GtkLabel widget. TRUE makes it break lines if text exceeds the widget's size. FALSE lets the text get cut off by the edge of the widget if it exceeds the widget size.

label : a GtkLabel
wrap : the setting

---

gtk_label_set

```
#define  gtk_label_set           gtk_label_set_text
```

Warning

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

Aliases `gtk_label_set_text()`. Probably used for backward compatibility with Gtk+ 1.0.x.

---

gtk_label_get_layout_offsets ()

```
void        gtk_label_get_layout_offsets    (GtkLabel *label,
                                             gint *x,
                                             gint *y);
```

Obtains the coordinates where the label will draw the PangoLayout representing the text in the label; useful to convert mouse events into coordinates inside the PangoLayout, e.g. to take some action if some part of the label is clicked. Of course you will need to create a GtkEventBox to receive the events, and pack the label inside it, since labels are a GTK_NO_WINDOW widget. Remember when using the PangoLayout functions you need to convert to and from pixels using `PANGO_PIXELS()` or PANGO_SCALE.

label : a GtkLabel
x : location to store X offset of layout, or NULL

y : location to store Y offset of layout, or NULL

---

gtk_label_get_mnemonic_keyval ()

```
guint       gtk_label_get_mnemonic_keyval   (GtkLabel *label);
```

If the label has been set so that it has an mnemonic key this function returns the keyval used for the mnemonic accelerator. If there is no mnemonic set up it returns GDK_VoidSymbol.

label : a GtkLabel

Returns : GDK keyval usable for accelerators, or GDK_VoidSymbol

---

gtk_label_get_selectable ()

```
gboolean    gtk_label_get_selectable        (GtkLabel *label);
```

Gets the value set by gtk_label_set_selectable().

label : a GtkLabel

Returns : TRUE if the user can copy text from the label

---

gtk_label_get_text ()

```
G_CONST_RETURN gchar* gtk_label_get_text    (GtkLabel *label);
```

Fetches the text from a label widget, as displayed on the screen. This does not include any embedded underlines indicating mnemonics or Pango markup. (See gtk_label_get_label())

label : a GtkLabel

Returns : the text in the label widget. This is the internal string used by the label, and must not be modified.

---

gtk_label_new_with_mnemonic ()

```
GtkWidget*  gtk_label_new_with_mnemonic     (const char *str);
```

Creates a new GtkLabel, containing the text in *str*.

If characters in *str* are preceded by an underscore, they are underlined. If you need a literal underscore character in a label, use '__' (two underscores). The first underlined character represents a keyboard accelerator called a mnemonic. The mnemonic key can be used to activate another widget, chosen automatically, or explicitly using gtk_label_set_mnemonic_widget().

If gtk_label_set_mnemonic_widget() is not called, then the first activatable ancestor of the GtkLabel will be chosen as the mnemonic widget. For instance, if the label is inside a button or menu item, the button or menu item will automatically become the mnemonic widget and be activated by the mnemonic.

str : The text of the label, with an underscore in front of the mnemonic character

Returns : the new GtkLabel

---

gtk_label_select_region ()

```
void        gtk_label_select_region         (GtkLabel *label,
                                             gint start_offset,
                                             gint end_offset);
```

Selects a range of characters in the label, if the label is selectable. See gtk_label_set_selectable(). If the label is not selectable, this function has no effect. If *start_offset* or *end_offset* are -1, then the end of the label will be substituted.

label : a GtkLabel

start_offset : start offset (in characters not bytes)

end_offset : end offset (in characters not bytes)

---

gtk_label_set_mnemonic_widget ()

```
void        gtk_label_set_mnemonic_widget   (GtkLabel *label,
                                             GtkWidget *widget);
```

If the label has been set so that it has an mnemonic key (using i.e. `gtk_label_set_markup_with_mnemonic()`, `gtk_label_set_text_with_mnemonic()`, `gtk_label_new_with_mnemonic()` or the "use_underline" property) the label can be associated with a widget that is the target of the mnemonic. When the label is inside a widget (like a GtkButton or a GtkNotebook tab) it is automatically associated with the correct widget, but sometimes (i.e. when the target is a GtkEntry next to the label) you need to set it explicitly using this function.

The target widget will be accelerated by emitting "mnemonic_activate" on it. The default handler for this signal will activate the widget if there are no mnemonic collisions and toggle focus between the colliding widgets otherwise.

label : a GtkLabel

widget : the target GtkWidget

---

gtk_label_set_selectable ()

```
void        gtk_label_set_selectable        (GtkLabel *label,
                                             gboolean setting);
```

Selectable labels allow the user to select text from the label, for copy-and-paste.

label : a GtkLabel

setting : TRUE to allow selecting text in the label

---

gtk_label_set_text_with_mnemonic ()

```
void        gtk_label_set_text_with_mnemonic
                                            (GtkLabel *label,
                                             const gchar *str);
```

Sets the label's text from the string *str*. If characters in *str* are preceded by an underscore, they are underlined indicating that they represent a keyboard accelerator called a mnemonic. The mnemonic key can be used to

activate another widget, chosen automatically, or explicitly using gtk_label_set_mnemonic_widget().

label : a GtkLabel
str : a string

---

gtk_label_get_attributes ()

```
PangoAttrList* gtk_label_get_attributes     (GtkLabel *label);
```

Gets the attribute list that was set on the label using gtk_label_set_attributes(), if any. This function does not reflect attributes that come from the labels markup (see gtk_label_set_markup()). If you want to get the effective attributes for the label, use pango_layout_get_attribute (gtk_label_get_layout (label)).

label : a GtkLabel
Returns : the attribute list, or NULL if none was set.

---

gtk_label_get_justify ()

```
GtkJustification gtk_label_get_justify      (GtkLabel *label);
```

Returns the justification of the label. See gtk_label_set_justify().

label : a GtkLabel
Returns : GtkJustification

---

gtk_label_get_ellipsize ()

```
PangoEllipsizeMode gtk_label_get_ellipsize  (GtkLabel *label);
```

Returns the ellipsizing position of the label. See gtk_label_set_ellipsize().

label : a GtkLabel

Returns : PangoEllipsizeMode

Since 2.6

---

gtk_label_get_width_chars ()

```
gint        gtk_label_get_width_chars       (GtkLabel *label);
```

Retrieves the desired width of *label*, in characters. See gtk_label_set_width_chars().

label : a GtkLabel

Returns : the width of a label in characters.

Since 2.6

---

gtk_label_get_label ()

```
G_CONST_RETURN gchar* gtk_label_get_label   (GtkLabel *label);
```

Fetches the text from a label widget including any embedded underlines indicating mnemonics and Pango markup. (See gtk_label_get_text()).

label : a GtkLabel

Returns : the text of the label widget. This string is owned by the widget and must not be modified or freed.

---

gtk_label_get_layout ()

```
PangoLayout* gtk_label_get_layout           (GtkLabel *label);
```

Gets the PangoLayout used to display the label. The layout is useful to e.g. convert text positions to pixel positions, in combination with `gtk_label_get_layout_offsets()`. The returned layout is owned by the label so need not be freed by the caller.

label : a GtkLabel

Returns : the PangoLayout for this label

---

gtk_label_get_line_wrap ()

```
gboolean    gtk_label_get_line_wrap         (GtkLabel *label);
```

Returns whether lines in the label are automatically wrapped. See `gtk_label_set_line_wrap()`.

label : a GtkLabel

Returns : TRUE if the lines of the label are automatically wrapped.

---

gtk_label_get_mnemonic_widget ()

```
GtkWidget*  gtk_label_get_mnemonic_widget   (GtkLabel *label);
```

Retrieves the target of the mnemonic (keyboard shortcut) of this label. See `gtk_label_set_mnemonic_widget()`.

label : a GtkLabel

Returns : the target of the label's mnemonic, or NULL if none has been set and the default algorithm will be used.

---

gtk_label_get_selection_bounds ()

```
gboolean    gtk_label_get_selection_bounds  (GtkLabel *label,
                                             gint *start,
                                             gint *end);
```

Gets the selected range of characters in the label, returning TRUE if there's a selection.

label : a GtkLabel
start : return location for start of selection, as a character offset
end : return location for end of selection, as a character offset
Returns : TRUE if selection is non-empty

---

gtk_label_get_use_markup ()

```
gboolean    gtk_label_get_use_markup        (GtkLabel *label);
```

Returns whether the label's text is interpreted as marked up with the Pango text markup language. See gtk_label_set_use_markup().

label : a GtkLabel
Returns : TRUE if the label's text will be parsed for markup.

---

gtk_label_get_use_underline ()

```
gboolean    gtk_label_get_use_underline     (GtkLabel *label);
```

Returns whether an embedded underline in the label indicates a mnemonic. See gtk_label_set_use_underline().

label : a GtkLabel
Returns : TRUE whether an embedded underline in the label indicates the mnemonic accelerator keys.

---

gtk_label_set_label ()

```
void        gtk_label_set_label             (GtkLabel *label,
                                             const gchar *str);
```

Sets the text of the label. The label is interpreted as including embedded underlines and/or Pango markup depending on the values of label->use_underline and label->use_markup.

label : a GtkLabel
str : the new text to set for the label

---

gtk_label_set_use_markup ()

```
void        gtk_label_set_use_markup        (GtkLabel *label,
                                             gboolean setting);
```

Sets whether the text of the label contains markup in Pango's text markup language. See `gtk_label_set_markup()`.

label : a GtkLabel
setting : TRUE if the label's text should be parsed for markup.

---

gtk_label_set_use_underline ()

```
void        gtk_label_set_use_underline     (GtkLabel *label,
                                             gboolean setting);
```

If true, an underline in the text indicates the next character should be used for the mnemonic accelerator key.

label : a GtkLabel
setting : TRUE if underlines in the text indicate mnemonics

Properties

The "attributes" property

```
  "attributes"           PangoAttrList         : Read / Write
```

A list of style attributes to apply to the text of the label.

The "cursor-position" property

```
"cursor-position"      gint                  : Read
```

The current position of the insertion cursor in chars.

Allowed values: >= 0

Default value: 0

The "ellipsize" property

```
"ellipsize"            PangoEllipsizeMode    : Read / Write
```

The preferred place to ellipsize the string, if the label does not have enough room to display the entire string, specified as a PangoEllisizeMode.

Note that setting this property to a value other than PANGO_ELLIPSIZE_NONE has the side-effect that the label requests only enough space to display the ellipsis "...". In particular, this means that ellipsizing labels don't work well in notebook tabs, unless the tab's ::tab-expand property is set to TRUE. Other means to set a label's width are gtk_widget_set_size_request() and gtk_label_set_width_chars().

Default value: PANGO_ELLIPSIZE_NONE

Since 2.6

The "justify" property

```
"justify"              GtkJustification      : Read / Write
```

The alignment of the lines in the text of the label relative to each other. This does NOT affect the alignment of the label within its allocation. See GtkMisc::xalign for that.

Default value: GTK_JUSTIFY_LEFT

The "label" property

```
"label"                gchararray            : Read / Write
```

The text of the label.

Default value: NULL

The "mnemonic-keyval" property

```
"mnemonic-keyval"      guint                 : Read
```

The mnemonic accelerator key for this label.

Default value: 16777215

The "mnemonic-widget" property

```
"mnemonic-widget"      GtkWidget             : Read / Write
```

The widget to be activated when the label's mnemonic key is pressed.

The "pattern" property

```
"pattern"              gchararray            : Write
```

A string with _ characters in positions correspond to characters in the text to underline.

Default value: NULL

The "selectable" property

```
"selectable"           gboolean              : Read / Write
```

Whether the label text can be selected with the mouse.

Default value: FALSE

The "selection-bound" property

```
"selection-bound"      gint                  : Read
```

The position of the opposite end of the selection from the cursor in chars.

Allowed values: >= 0

Default value: 0

The "use-markup" property

```
"use-markup"           gboolean              : Read / Write
```

The text of the label includes XML markup. See pango_parse_markup().

Default value: FALSE

The "use-underline" property

```
"use-underline"       gboolean              : Read / Write
```

If set, an underline in the text indicates the next character should be used for the mnemonic accelerator key.

Default value: FALSE

The "width-chars" property

```
"width-chars"         gint                  : Read / Write
```

The desired width of the label, in characters. If this property is set to `-1`, the width will be calculated automatically, otherwise the label will request either 3 characters or the property value, whichever is greater.

Allowed values: >= -1

Default value: -1

Since 2.6

The "wrap" property

```
"wrap"                gboolean              : Read / Write
```

If set, wrap lines if the text becomes too wide.

Default value: FALSE

Signals

The "copy-clipboard" signal

```
void        user_function                   (GtkLabel *label,
                                             gpointer user_data);
```

label : the object which received the signal.
user_data : user data set when the signal handler was connected.

The "move-cursor" signal

```
void        user_function                   (GtkLabel *label,
                                             GtkMovementStep arg1,
                                             gint arg2,
                                             gboolean arg3,
                                             gpointer user_data);
```

label : the object which received the signal.
arg1 :
arg2 :
arg3 :
user_data : user data set when the signal handler was connected.

The "populate-popup" signal

```
void        user_function                   (GtkLabel *label,
                                             GtkMenu *arg1,
                                             gpointer user_data);
```

label : the object which received the signal.
arg1 :
user_data : user data set when the signal handler was connected.

<< GtkImage **GtkProgressBar >>**

GtkProgressBar

GtkProgressBar — A widget which indicates progress visually

Synopsis

```
#include <gtk/gtk.h>

            GtkProgressBar;
GtkWidget*  gtk_progress_bar_new            (void);
void        gtk_progress_bar_pulse          (GtkProgressBar *pbar);
void        gtk_progress_bar_set_text       (GtkProgressBar *pbar,
                                             const gchar *text);
void        gtk_progress_bar_set_fraction   (GtkProgressBar *pbar,
                                             gdouble fraction);
void        gtk_progress_bar_set_pulse_step (GtkProgressBar *pbar,
                                             gdouble fraction);
void        gtk_progress_bar_set_orientation
                                            (GtkProgressBar *pbar,
                                             GtkProgressBarOrientation orientation);
enum        GtkProgressBarOrientation;
G_CONST_RETURN gchar* gtk_progress_bar_get_text
                                            (GtkProgressBar *pbar);
gdouble     gtk_progress_bar_get_fraction   (GtkProgressBar *pbar);
gdouble     gtk_progress_bar_get_pulse_step (GtkProgressBar *pbar);
GtkProgressBarOrientation gtk_progress_bar_get_orientation
                                            (GtkProgressBar *pbar);
GtkWidget*  gtk_progress_bar_new_with_adjustment
                                            (GtkAdjustment *adjustment);
void        gtk_progress_bar_set_bar_style  (GtkProgressBar *pbar,
                                             GtkProgressBarStyle style);
enum        GtkProgressBarStyle;
void        gtk_progress_bar_set_discrete_blocks
                                            (GtkProgressBar *pbar,
                                             guint blocks);
void        gtk_progress_bar_set_activity_step
                                            (GtkProgressBar *pbar,
                                             guint step);
void        gtk_progress_bar_set_activity_blocks
                                            (GtkProgressBar *pbar,
```

```
                                                guint blocks);
void        gtk_progress_bar_update            (GtkProgressBar *pbar,
                                                gdouble percentage);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkWidget
               +----GtkProgress
                     +----GtkProgressBar
```

Implemented Interfaces

GtkProgressBar implements AtkImplementorIface.

Properties

```
  "activity-blocks"      guint                 : Read / Write
  "activity-step"        guint                 : Read / Write
  "adjustment"           GtkAdjustment         : Read / Write
  "bar-style"            GtkProgressBarStyle   : Read / Write
  "discrete-blocks"      guint                 : Read / Write
  "fraction"             gdouble               : Read / Write
  "orientation"          GtkProgressBarOrientation  : Read / Write
  "pulse-step"           gdouble               : Read / Write
  "text"                 gchararray            : Read / Write
```

Description

The GtkProgressBar is typically used to display the progress of a long running operation. It provides a visual clue that processing is underway. The GtkProgressBar can be used in two different modes: percentage mode and activity mode.

When an application can determine how much work needs to take place (e.g. read a fixed number of bytes from a file) and can monitor its progress, it can use the GtkProgressBar in percentage mode and the user sees a growing bar indicating the percentage of the work that has been completed. In this mode, the application is required to call `gtk_progress_bar_set_fraction()` periodically to update the progress bar.

When an application has no accurate way of knowing the amount of work to do, it can use the GtkProgressBar in activity mode,

which shows activity by a block moving back and forth within the progress area. In this mode, the application is required to call `gtk_progress_bar_pulse()` perodically to update the progress bar.

There is quite a bit of flexibility provided to control the appearance of the GtkProgressBar. Functions are provided to control the orientation of the bar, optional text can be displayed along with the bar, and the step size used in activity mode can be set.

Note

The GtkProgressBar/GtkProgress API in GTK 1.2 was bloated, needlessly complex and hard to use properly. Therefore GtkProgress has been deprecated completely and the GtkProgressBar API has been reduced to the following 10 functions: `gtk_progress_bar_new()`, `gtk_progress_bar_pulse()`, `gtk_progress_bar_set_text()`, `gtk_progress_bar_set_fraction()`, `gtk_progress_bar_set_pulse_step()`, `gtk_progress_bar_set_orientation()`, `gtk_progress_bar_get_text()`, `gtk_progress_bar_get_fraction()`, `gtk_progress_bar_get_pulse_step()`, `gtk_progress_bar_get_orientation()`. These have been grouped at the beginning of this section, followed by a large chunk of deprecated 1.2 compatibility functions.

Details

GtkProgressBar

```
typedef struct _GtkProgressBar GtkProgressBar;
```

The GtkProgressBar-struct struct contains private data only, and should be accessed using the functions below.

---

gtk_progress_bar_new ()

```
GtkWidget*  gtk_progress_bar_new           (void);
```

Creates a new GtkProgressBar.

Returns : a GtkProgressBar.

---

gtk_progress_bar_pulse ()

```
void        gtk_progress_bar_pulse         (GtkProgressBar *pbar);
```

Indicates that some progress is made, but you don't know how much. Causes the progress bar to enter "activity mode," where a block bounces back and forth. Each call to `gtk_progress_bar_pulse()` causes the block to move by a little bit (the

amount of movement per pulse is determined by gtk_progress_bar_set_pulse_step()).

pbar : a GtkProgressBar

---

gtk_progress_bar_set_text ()

```
void        gtk_progress_bar_set_text       (GtkProgressBar *pbar,
                                             const gchar *text);
```

Causes the given *text* to appear superimposed on the progress bar.

pbar : a GtkProgressBar
text : a UTF-8 string

---

gtk_progress_bar_set_fraction ()

```
void        gtk_progress_bar_set_fraction   (GtkProgressBar *pbar,
                                             gdouble fraction);
```

Causes the progress bar to "fill in" the given fraction of the bar. The fraction should be between 0.0 and 1.0, inclusive.

pbar : a GtkProgressBar
fraction : fraction of the task that's been completed

---

gtk_progress_bar_set_pulse_step ()

```
void        gtk_progress_bar_set_pulse_step (GtkProgressBar *pbar,
                                             gdouble fraction);
```

Sets the fraction of total progress bar length to move the bouncing block for each call to gtk_progress_bar_pulse().

pbar : a GtkProgressBar
fraction : fraction between 0.0 and 1.0

---

gtk_progress_bar_set_orientation ()

```
void        gtk_progress_bar_set_orientation
                                            (GtkProgressBar *pbar,
                                             GtkProgressBarOrientation orientation);
```

Causes the progress bar to switch to a different orientation (left-to-right, right-to-left, top-to-bottom, or bottom-to-top).

pbar : a GtkProgressBar

orientation : orientation of the progress bar

---

enum GtkProgressBarOrientation

```
typedef enum
{
  GTK_PROGRESS_LEFT_TO_RIGHT,
  GTK_PROGRESS_RIGHT_TO_LEFT,
  GTK_PROGRESS_BOTTOM_TO_TOP,
  GTK_PROGRESS_TOP_TO_BOTTOM
} GtkProgressBarOrientation;
```

An enumeration representing possible orientations and growth directions for the visible progress bar.

`GTK_PROGRESS_LEFT_TO_RIGHT` A horizontal progress bar growing from left to right.

`GTK_PROGRESS_RIGHT_TO_LEFT` A horizontal progress bar growing from right to left.

`GTK_PROGRESS_BOTTOM_TO_TOP` A vertical progress bar growing from bottom to top.

`GTK_PROGRESS_TOP_TO_BOTTOM` A vertical progress bar growing from top to bottom.

---

gtk_progress_bar_get_text ()

```
G_CONST_RETURN gchar* gtk_progress_bar_get_text
                                            (GtkProgressBar *pbar);
```

Retrieves the text displayed superimposed on the progress bar, if any, otherwise `NULL`. The return value is a reference to the text, not a copy of it, so will become invalid if you change the text in the progress bar.

pbar : a GtkProgressBar

Returns : text, or `NULL`; this string is owned by the widget and should not be modified or freed.

---

gtk_progress_bar_get_fraction ()

```
gdouble     gtk_progress_bar_get_fraction   (GtkProgressBar *pbar);
```

Returns the current fraction of the task that's been completed.

pbar : a GtkProgressBar

Returns : a fraction from 0.0 to 1.0

---

gtk_progress_bar_get_pulse_step ()

```
gdouble     gtk_progress_bar_get_pulse_step (GtkProgressBar *pbar);
```

Retrieves the pulse step set with `gtk_progress_bar_set_pulse_step()`

pbar : a GtkProgressBar

Returns : a fraction from 0.0 to 1.0

---

gtk_progress_bar_get_orientation ()

```
GtkProgressBarOrientation gtk_progress_bar_get_orientation
                                            (GtkProgressBar *pbar);
```

Retrieves the current progress bar orientation.

pbar : a GtkProgressBar

Returns : orientation of the progress bar

---

gtk_progress_bar_new_with_adjustment ()

```
GtkWidget*  gtk_progress_bar_new_with_adjustment
                                            (GtkAdjustment *adjustment);
```

Warning

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

Creates a new GtkProgressBar with an associated GtkAdjustment.

adjustment : a GtkAdjustment.

Returns : a GtkProgressBar.

---

gtk_progress_bar_set_bar_style ()

```
void        gtk_progress_bar_set_bar_style  (GtkProgressBar *pbar,
                                             GtkProgressBarStyle style);
```

Warning

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

Sets the style of the GtkProgressBar. The default style is `GTK_PROGRESS_CONTINUOUS`.

pbar : a GtkProgressBar.

style : a GtkProgressBarStyle value indicating the desired style.

---

enum GtkProgressBarStyle

```
typedef enum
{
  GTK_PROGRESS_CONTINUOUS,
  GTK_PROGRESS_DISCRETE
} GtkProgressBarStyle;
```

An enumeration representing the styles for drawing the progress bar.

`GTK_PROGRESS_CONTINUOUS` The progress bar grows in a smooth, continuous manner.

`GTK_PROGRESS_DISCRETE` The progress bar grows in discrete, visible blocks.

---

gtk_progress_bar_set_discrete_blocks ()

```
void        gtk_progress_bar_set_discrete_blocks
                                            (GtkProgressBar *pbar,
                                             guint blocks);
```

Warning

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

Sets the number of blocks that the progress bar is divided into when the style is `GTK_PROGRESS_DISCRETE`.

`pbar` : a GtkProgressBar.

`blocks` : number of individual blocks making up the bar.

---

gtk_progress_bar_set_activity_step ()

```
void        gtk_progress_bar_set_activity_step
                                            (GtkProgressBar *pbar,
                                             guint step);
```

Warning

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

Sets the step value used when the progress bar is in activity mode. The step is the amount by which the progress is incremented each iteration.

`pbar` : a GtkProgressBar.

`step` : the amount which the progress is incremented in activity mode.

---

gtk_progress_bar_set_activity_blocks ()

```
void        gtk_progress_bar_set_activity_blocks
                                            (GtkProgressBar *pbar,
                                             guint blocks);
```

Warning

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

Sets the number of blocks used when the progress bar is in activity mode. Larger numbers make the visible block smaller.

`pbar` : a GtkProgressBar.

`blocks` : number of blocks which can fit within the progress bar area.

---

gtk_progress_bar_update ()

```
void        gtk_progress_bar_update         (GtkProgressBar *pbar,
                                             gdouble percentage);
```

Warning

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

This function is deprecated. Please use `gtk_progress_set_value()` or `gtk_progress_set_percentage()` instead.

pbar : a GtkProgressBar.

percentage : the new percent complete value.

Properties

The "activity-blocks" property

```
  "activity-blocks"      guint                 : Read / Write
```

The number of blocks which can fit in the progress bar area in activity mode (Deprecated).

Allowed values: >= 2

Default value: 5

The "activity-step" property

```
  "activity-step"        guint                 : Read / Write
```

The increment used for each iteration in activity mode (Deprecated).

Allowed values: >= 1

Default value: 3

The "adjustment" property

```
"adjustment"           GtkAdjustment         : Read / Write
```

The GtkAdjustment connected to the progress bar (Deprecated).

The "bar-style" property

```
"bar-style"            GtkProgressBarStyle   : Read / Write
```

Specifies the visual style of the bar in percentage mode (Deprecated).

Default value: GTK_PROGRESS_CONTINUOUS

The "discrete-blocks" property

```
"discrete-blocks"      guint                 : Read / Write
```

The number of discrete blocks in a progress bar (when shown in the discrete style).

Allowed values: >= 2

Default value: 10

The "fraction" property

```
"fraction"             gdouble               : Read / Write
```

The fraction of total work that has been completed.

Allowed values: [0,1]

Default value: 0

The "orientation" property

```
  "orientation"          GtkProgressBarOrientation  : Read / Write
```

Orientation and growth direction of the progress bar.

Default value: GTK_PROGRESS_LEFT_TO_RIGHT

The "pulse-step" property

```
  "pulse-step"           gdouble                    : Read / Write
```

The fraction of total progress to move the bouncing block when pulsed.

Allowed values: [0,1]

Default value: 0.1

The "text" property

```
  "text"                 gchararray                 : Read / Write
```

Text to be displayed in the progress bar.

Default value: "%P %%"

<< GtkLabel **GtkStatusbar >>**

GtkStatusbar

GtkStatusbar — Report messages of minor importance to the user

Synopsis

```
#include <gtk/gtk.h>

            GtkStatusbar;
GtkWidget*  gtk_statusbar_new               (void);
guint       gtk_statusbar_get_context_id    (GtkStatusbar *statusbar,
                                             const gchar *context_description);
guint       gtk_statusbar_push              (GtkStatusbar *statusbar,
                                             guint context_id,
                                             const gchar *text);
void        gtk_statusbar_pop               (GtkStatusbar *statusbar,
                                             guint context_id);
void        gtk_statusbar_remove            (GtkStatusbar *statusbar,
                                             guint context_id,
                                             guint message_id);
void        gtk_statusbar_set_has_resize_grip
                                            (GtkStatusbar *statusbar,
                                             gboolean setting);
gboolean    gtk_statusbar_get_has_resize_grip
                                            (GtkStatusbar *statusbar);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkWidget
               +----GtkContainer
                     +----GtkBox
```

```
                    +----GtkHBox
                          +----GtkStatusbar
```

Implemented Interfaces

GtkStatusbar implements AtkImplementorIface.

Properties

```
  "has-resize-grip"      gboolean             : Read / Write
```

Style Properties

```
  "shadow-type"          GtkShadowType        : Read
```

Signal Prototypes

```
"text-popped"
            void        user_function      (GtkStatusbar *statusbar,
                                            guint context_id,
                                            gchar *text,
                                            gpointer user_data);
"text-pushed"
            void        user_function      (GtkStatusbar *statusbar,
                                            guint context_id,
                                            gchar *text,
                                            gpointer user_data);
```

Description

A GtkStatusbar is usually placed along the bottom of an application's main GtkWindow. It may provide a regular commentary of the application's status (as is usually the case in a web browser, for example), or may be used to simply output a message when the status changes, (when an upload is complete in an FTP client, for example). It may also have a resize grip (a triangular area in the lower right corner) which can be clicked on to resize the window containing the statusbar.

Status bars in Gtk+ maintain a stack of messages. The message at the top of the each bar's stack is the one that will currently be displayed.

Any messages added to a statusbar's stack must specify a *context_id* that is used to uniquely identify the source of a message. This context_id can be generated by `gtk_statusbar_get_context_id()`, given a message and the statusbar that it will be added to. Note that messages are stored in a stack, and when choosing which message to display, the stack structure is adhered to, regardless of the context identifier of a message.

Status bars are created using `gtk_statusbar_new()`.

Messages are added to the bar's stack with `gtk_statusbar_push()`.

The message at the top of the stack can be removed using `gtk_statusbar_pop()`. A message can be removed from anywhere in the stack if its message_id was recorded at the time it was added. This is done using `gtk_statusbar_remove()`.

Details

GtkStatusbar

```
typedef struct _GtkStatusbar GtkStatusbar;
```

Contains private data that should be modified with the functions described below.

---

gtk_statusbar_new ()

```
GtkWidget*  gtk_statusbar_new               (void);
```

Creates a new GtkStatusbar ready for messages.

Returns : the new GtkStatusbar.

---

gtk_statusbar_get_context_id ()

```
guint       gtk_statusbar_get_context_id    (GtkStatusbar *statusbar,
                                             const gchar *context_description);
```

Returns a new context identifier, given a description of the actual context.

statusbar : a GtkStatusbar.
context_description : textual description of what context the new message is being used in.
Returns : an integer id.

---

gtk_statusbar_push ()

```
guint       gtk_statusbar_push              (GtkStatusbar *statusbar,
                                             guint context_id,
                                             const gchar *text);
```

Pushes a new message onto a statusbar's stack.

statusbar : a GtkStatusbar.
context_id : the message's context id, as returned by gtk_statusbar_get_context_id().
text : the message to add to the statusbar.
Returns : the message's new message id for use with gtk_statusbar_remove().

---

gtk_statusbar_pop ()

```
void        gtk_statusbar_pop               (GtkStatusbar *statusbar,
                                             guint context_id);
```

Removes the message at the top of a GtkStatusBar's stack.

statusbar : a GtkStatusBar.
context_id : a context identifier.

---

gtk_statusbar_remove ()

```
void        gtk_statusbar_remove            (GtkStatusbar *statusbar,
                                             guint context_id,
                                             guint message_id);
```

Forces the removal of a message from a statusbar's stack. The exact *context_id* and *message_id* must be specified.

statusbar : a GtkStatusBar.
context_id : a context identifier.
message_id : a message identifier, as returned by gtk_statusbar_push().

---

gtk_statusbar_set_has_resize_grip ()

```
void        gtk_statusbar_set_has_resize_grip
                                            (GtkStatusbar *statusbar,
                                             gboolean setting);
```

Sets whether the statusbar has a resize grip. TRUE by default.

statusbar : a GtkStatusBar.
setting : TRUE to have a resize grip.

---

gtk_statusbar_get_has_resize_grip ()

```
gboolean    gtk_statusbar_get_has_resize_grip
                                            (GtkStatusbar *statusbar);
```

Returns whether the statusbar has a resize grip.

statusbar : a GtkStatusBar.
Returns : TRUE if the statusbar has a resize grip.

Properties

The "has-resize-grip" property

```
  "has-resize-grip"      gboolean              : Read / Write
```

Whether the statusbar has a grip for resizing the toplevel window.

Default value: TRUE

Since 2.4

Style Properties

The "shadow-type" style property

```
  "shadow-type"          GtkShadowType         : Read
```

Style of bevel around the statusbar text.

Default value: GTK_SHADOW_IN

Signals

The "text-popped" signal

```
void        user_function                  (GtkStatusbar *statusbar,
                                            guint context_id,
                                            gchar *text,
                                            gpointer user_data);
```

Is emitted whenever a new message is popped off a statusbar's stack.

statusbar : the object which received the signal.
context_id : the context id of the relevant message/statusbar.
text : the message that was just popped.
user_data : user data set when the signal handler was connected.

The "text-pushed" signal

```
void        user_function                  (GtkStatusbar *statusbar,
                                            guint context_id,
                                            gchar *text,
                                            gpointer user_data);
```

Is emitted whenever a new message gets pushed onto a statusbar's stack.

statusbar : the object which received the signal.
context_id : the context id of the relevant message/statusbar.
text : the message that was pushed.
user_data : user data set when the signal handler was connected.

See Also

GtkDialog another way of reporting information to the user.

<< GtkProgressBar **Buttons and Toggles >>**

Buttons and Toggles

GtkButton - A widget that creates a signal when clicked on
GtkCheckButton - Create widgets with a discrete toggle button
GtkRadioButton - A choice from multiple check buttons
GtkToggleButton - Create buttons which retain their state

<< GtkStatusbar **GtkButton >>**

GtkButton

GtkButton — A widget that creates a signal when clicked on

Synopsis

```
#include <gtk/gtk.h>

            GtkButton;
GtkWidget*  gtk_button_new                  (void);
GtkWidget*  gtk_button_new_with_label       (const gchar *label);
GtkWidget*  gtk_button_new_with_mnemonic    (const gchar *label);
GtkWidget*  gtk_button_new_from_stock       (const gchar *stock_id);
void        gtk_button_pressed              (GtkButton *button);
void        gtk_button_released             (GtkButton *button);
void        gtk_button_clicked              (GtkButton *button);
void        gtk_button_enter                (GtkButton *button);
void        gtk_button_leave                (GtkButton *button);
void        gtk_button_set_relief           (GtkButton *button,
                                             GtkReliefStyle newstyle);
GtkReliefStyle gtk_button_get_relief        (GtkButton *button);
G_CONST_RETURN gchar* gtk_button_get_label  (GtkButton *button);
void        gtk_button_set_label            (GtkButton *button,
                                             const gchar *label);
gboolean    gtk_button_get_use_stock        (GtkButton *button);
void        gtk_button_set_use_stock        (GtkButton *button,
                                             gboolean use_stock);
gboolean    gtk_button_get_use_underline    (GtkButton *button);
void        gtk_button_set_use_underline    (GtkButton *button,
                                             gboolean use_underline);
void        gtk_button_set_focus_on_click   (GtkButton *button,
                                             gboolean focus_on_click);
gboolean    gtk_button_get_focus_on_click   (GtkButton *button);
void        gtk_button_set_alignment        (GtkButton *button,
```

```
                                                    gfloat xalign,
                                                    gfloat yalign);
void        gtk_button_get_alignment        (GtkButton *button,
                                                    gfloat *xalign,
                                                    gfloat *yalign);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkWidget
               +----GtkContainer
                     +----GtkBin
                           +----GtkButton
                                 +----GtkToggleButton
                                 +----GtkColorButton
                                 +----GtkFontButton
                                 +----GtkOptionMenu
```

Implemented Interfaces

GtkButton implements AtkImplementorIface.

Properties

```
  "focus-on-click"       gboolean              : Read / Write
  "label"                gchararray            : Read / Write / Construct
  "relief"               GtkReliefStyle        : Read / Write
  "use-stock"            gboolean              : Read / Write / Construct
  "use-underline"        gboolean              : Read / Write / Construct
  "xalign"               gfloat                : Read / Write
  "yalign"               gfloat                : Read / Write
```

Style Properties

```
  "child-displacement-x" gint                  : Read
  "child-displacement-y" gint                  : Read
  "default-border"       GtkBorder             : Read
  "default-outside-border" GtkBorder             : Read
  "displace-focus"       gboolean              : Read
```

Signal Prototypes

```
"activate"  void        user_function      (GtkButton *widget,
                                            gpointer user_data);
"clicked"   void        user_function      (GtkButton *button,
                                            gpointer user_data);
"enter"     void        user_function      (GtkButton *button,
                                            gpointer user_data);
"leave"     void        user_function      (GtkButton *button,
                                            gpointer user_data);
"pressed"   void        user_function      (GtkButton *button,
                                            gpointer user_data);
"released"  void        user_function      (GtkButton *button,
                                            gpointer user_data);
```

Description

The GtkButton widget is generally used to attach a function to that is called when the button is pressed. The various signals and how to use them are outlined below.

The GtkButton widget can hold any valid child widget. That is it can hold most any other standard GtkWidget. The most commonly used child is the GtkLabel.

Details

GtkButton

```
typedef struct _GtkButton GtkButton;
```

This should not be accessed directly. Use the accessor functions below.

---

gtk_button_new ()

```
GtkWidget*  gtk_button_new                  (void);
```

Creates a new GtkButton widget. To add a child widget to the button, use `gtk_container_add()`.

Returns : The newly created GtkButton widget.

---

gtk_button_new_with_label ()

```
GtkWidget*  gtk_button_new_with_label       (const gchar *label);
```

Creates a GtkButton widget with a GtkLabel child containing the given text.

label : The text you want the GtkLabel to hold.
Returns : The newly created GtkButton widget.

---

gtk_button_new_with_mnemonic ()

```
GtkWidget*  gtk_button_new_with_mnemonic    (const gchar *label);
```

Creates a new GtkButton containing a label. If characters in *label* are preceded by an underscore, they are underlined. If you need a literal underscore character in a label, use '__' (two underscores). The first underlined character represents a keyboard accelerator called a mnemonic. Pressing Alt and that key activates the button.

label : The text of the button, with an underscore in front of the mnemonic character

Returns : a new GtkButton

---

gtk_button_new_from_stock ()

```
GtkWidget*  gtk_button_new_from_stock       (const gchar *stock_id);
```

Creates a new GtkButton containing the image and text from a stock item. Some stock ids have preprocessor macros like GTK_STOCK_OK and GTK_STOCK_APPLY.

If *stock_id* is unknown, then it will be treated as a mnemonic label (as for gtk_button_new_with_mnemonic()).

stock_id : the name of the stock item

Returns : a new GtkButton

---

gtk_button_pressed ()

```
void        gtk_button_pressed              (GtkButton *button);
```

Emits a GtkButton::pressed signal to the given GtkButton.

button : The GtkButton you want to send the signal to.

---

gtk_button_released ()

```
void        gtk_button_released             (GtkButton *button);
```

Emits a GtkButton::released signal to the given GtkButton.

button : The GtkButton you want to send the signal to.

---

gtk_button_clicked ()

```
void        gtk_button_clicked              (GtkButton *button);
```

Emits a GtkButton::clicked signal to the given GtkButton.

button : The GtkButton you want to send the signal to.

---

gtk_button_enter ()

```
void        gtk_button_enter                (GtkButton *button);
```

Emits a GtkButton::enter signal to the given GtkButton.

button : The GtkButton you want to send the signal to.

---

gtk_button_leave ()

```
void        gtk_button_leave                (GtkButton *button);
```

Emits a GtkButton::leave signal to the given GtkButton.

button : The GtkButton you want to send the signal to.

---

gtk_button_set_relief ()

```
void        gtk_button_set_relief           (GtkButton *button,
                                             GtkReliefStyle newstyle);
```

Sets the relief style of the edges of the given GtkButton widget. Three styles exist, GTK_RELIEF_NORMAL,

GTK_RELIEF_HALF, GTK_RELIEF_NONE. The default style is, as one can guess, GTK_RELIEF_NORMAL.

button : The GtkButton you want to set relief styles of.

newstyle : The GtkReliefStyle as described above.

---

gtk_button_get_relief ()

```
GtkReliefStyle gtk_button_get_relief        (GtkButton *button);
```

Returns the current relief style of the given GtkButton.

button : The GtkButton you want the GtkReliefStyle from.

Returns : The current GtkReliefStyle

---

gtk_button_get_label ()

```
G_CONST_RETURN gchar* gtk_button_get_label  (GtkButton *button);
```

Fetches the text from the label of the button, as set by `gtk_button_set_label()`. If the label text has not been set the return value will be `NULL`. This will be the case if you create an empty button with `gtk_button_new()` to use as a container.

button : a GtkButton

Returns : The text of the label widget. This string is owned by the widget and must not be modified or freed.

---

gtk_button_set_label ()

```
void        gtk_button_set_label            (GtkButton *button,
                                             const gchar *label);
```

Sets the text of the label of the button to *str*. This text is also used to select the stock item if gtk_button_set_use_stock() is used.

This will also clear any previously set labels.

button : a GtkButton
label : a string

---

gtk_button_get_use_stock ()

```
gboolean    gtk_button_get_use_stock        (GtkButton *button);
```

Returns whether the button label is a stock item.

button : a GtkButton
Returns : TRUE if the button label is used to select a stock item instead of being used directly as the label text.

---

gtk_button_set_use_stock ()

```
void        gtk_button_set_use_stock        (GtkButton *button,
                                             gboolean use_stock);
```

If true, the label set on the button is used as a stock id to select the stock item for the button.

button : a GtkButton
use_stock : TRUE if the button should use a stock item

---

gtk_button_get_use_underline ()

```
gboolean    gtk_button_get_use_underline    (GtkButton *button);
```

Returns whether an embedded underline in the button label indicates a mnemonic. See `gtk_button_set_use_underline()`.

button : a GtkButton

Returns : TRUE if an embedded underline in the button label indicates the mnemonic accelerator keys.

---

gtk_button_set_use_underline ()

```
void        gtk_button_set_use_underline    (GtkButton *button,
                                             gboolean use_underline);
```

If true, an underline in the text of the button label indicates the next character should be used for the mnemonic accelerator key.

button : a GtkButton

use_underline : TRUE if underlines in the text indicate mnemonics

---

gtk_button_set_focus_on_click ()

```
void        gtk_button_set_focus_on_click   (GtkButton *button,
                                             gboolean focus_on_click);
```

Sets whether the button will grab focus when it is clicked with the mouse. Making mouse clicks not grab focus is useful in places like toolbars where you don't want the keyboard focus removed from the main area of the application.

button : a GtkButton

focus_on_click : whether the button grabs focus when clicked with the mouse

Since 2.4

---

gtk_button_get_focus_on_click ()

```
gboolean    gtk_button_get_focus_on_click   (GtkButton *button);
```

Returns whether the button grabs focus when it is clicked with the mouse. See `gtk_button_set_focus_on_click()`.

button : a GtkButton

Returns : TRUE if the button grabs focus when it is clicked with the mouse.

Since 2.4

---

gtk_button_set_alignment ()

```
void        gtk_button_set_alignment        (GtkButton *button,
                                             gfloat xalign,
                                             gfloat yalign);
```

Sets the alignment of the child. This property has no effect unless the child is a GtkMisc or a GtkAligment.

button : a GtkButton

xalign : the horizontal position of the child, 0.0 is left aligned, 1.0 is right aligned

yalign : the vertical position of the child, 0.0 is top aligned, 1.0 is bottom aligned

Since 2.4

---

gtk_button_get_alignment ()

```
void        gtk_button_get_alignment        (GtkButton *button,
                                             gfloat *xalign,
                                             gfloat *yalign);
```

Gets the alignment of the child in the button.

button : a GtkButton
xalign : return location for horizontal alignment
yalign : return location for vertical alignment

Since 2.4

Properties

The "focus-on-click" property

```
"focus-on-click"       gboolean              : Read / Write
```

Whether the button grabs focus when it is clicked with the mouse.

Default value: TRUE

The "label" property

```
"label"                gchararray            : Read / Write / Construct
```

Text of the label widget inside the button, if the button contains a label widget.

Default value: NULL

The "relief" property

```
"relief"               GtkReliefStyle        : Read / Write
```

The border relief style.

Default value: GTK_RELIEF_NORMAL

The "use-stock" property

```
"use-stock"            gboolean              : Read / Write / Construct
```

If set, the label is used to pick a stock item instead of being displayed.

Default value: FALSE

The "use-underline" property

```
"use-underline"        gboolean              : Read / Write / Construct
```

If set, an underline in the text indicates the next character should be used for the mnemonic accelerator key.

Default value: FALSE

The "xalign" property

```
"xalign"               gfloat                : Read / Write
```

If the child of the button is a GtkMisc or GtkAlignment, this property can be used to control it's horizontal alignment. 0.0 is left aligned, 1.0 is right aligned.

Allowed values: [0,1]

Default value: 0.5

Since 2.4

The "yalign" property

```
"yalign"               gfloat              : Read / Write
```

If the child of the button is a GtkMisc or GtkAlignment, this property can be used to control it's vertical alignment. 0.0 is top aligned, 1.0 is bottom aligned.

Allowed values: [0,1]

Default value: 0.5

Since 2.4

Style Properties

The "child-displacement-x" style property

```
"child-displacement-x" gint                : Read
```

How far in the x direction to move the child when the button is depressed.

Default value: 0

The "child-displacement-y" style property

```
"child-displacement-y" gint                : Read
```

How far in the y direction to move the child when the button is depressed.

Default value: 0

The "default-border" style property

```
"default-border"       GtkBorder             : Read
```

Extra space to add for CAN_DEFAULT buttons.

The "default-outside-border" style property

```
"default-outside-border" GtkBorder             : Read
```

Extra space to add for CAN_DEFAULT buttons that is always drawn outside the border.

The "displace-focus" style property

```
"displace-focus"       gboolean              : Read
```

Whether the child_displacement_x/child_displacement_y properties should also affect the focus rectangle.

Default value: FALSE

Since 2.6

Signals

The "activate" signal

```
void        user_function                     (GtkButton *widget,
                                               gpointer user_data);
```

The "activate" signal on GtkButton is an action signal and emitting it causes the button to animate press then release. Applications should never connect to this signal, but use the "clicked" signal.

widget : the object which received the signal.
user_data : user data set when the signal handler was connected.

The "clicked" signal

```
void        user_function                   (GtkButton *button,
                                             gpointer user_data);
```

Emitted when a button clicked on by the mouse and the cursor stays on the button. If the cursor is not on the button when the mouse button is released, the signal is not emitted.

button : the object which received the signal.
user_data : user data set when the signal handler was connected.

The "enter" signal

```
void        user_function                   (GtkButton *button,
                                             gpointer user_data);
```

Emitted when the mouse cursor enters the region of the button.

button : the object which received the signal.
user_data : user data set when the signal handler was connected.

The "leave" signal

```
void        user_function                   (GtkButton *button,
                                             gpointer user_data);
```

Emitted when the mouse cursor leaves the region of the button.

button : the object which received the signal.

user_data : user data set when the signal handler was connected.

The "pressed" signal

```
void        user_function                   (GtkButton *button,
                                             gpointer user_data);
```

Emitted when the button is initially pressed.

button : the object which received the signal.
user_data : user data set when the signal handler was connected.

The "released" signal

```
void        user_function                   (GtkButton *button,
                                             gpointer user_data);
```

Emitted when a button which is pressed is released, no matter where the mouse cursor is.

button : the object which received the signal.
user_data : user data set when the signal handler was connected.

<< Buttons and Toggles **GtkCheckButton >>**

GtkCheckButton

GtkCheckButton — Create widgets with a discrete toggle button

Synopsis

```
#include <gtk/gtk.h>


            GtkCheckButton;
GtkWidget*  gtk_check_button_new            (void);
GtkWidget*  gtk_check_button_new_with_label (const gchar *label);
GtkWidget*  gtk_check_button_new_with_mnemonic
                                            (const gchar *label);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkWidget
               +----GtkContainer
                     +----GtkBin
                           +----GtkButton
                                 +----GtkToggleButton
                                       +----GtkCheckButton
                                             +----GtkRadioButton
```

Implemented Interfaces

GtkCheckButton implements AtkImplementorIface.

Style Properties

```
  "indicator-size"       gint                  : Read
  "indicator-spacing"    gint                  : Read
```

Description

A GtkCheckButton places a discrete GtkToggleButton next to a widget, (usually a GtkLabel). See the section on GtkToggleButton widgets for more information about toggle/check buttons.

The important signal ('toggled') is also inherited from GtkToggleButton.

Details

GtkCheckButton

```
typedef struct _GtkCheckButton GtkCheckButton;
```

toggle_button is a GtkToggleButton representing the actual toggle button that composes the check button.

---

gtk_check_button_new ()

```
GtkWidget*  gtk_check_button_new           (void);
```

Creates a new GtkCheckButton.

Returns : a GtkWidget.

---

gtk_check_button_new_with_label ()

```
GtkWidget*  gtk_check_button_new_with_label (const gchar *label);
```

Creates a new GtkCheckButton with a GtkLabel to the right of it.

label : the text for the check button.
Returns : a GtkWidget.

---

gtk_check_button_new_with_mnemonic ()

```
GtkWidget*  gtk_check_button_new_with_mnemonic
                                        (const gchar *label);
```

Creates a new GtkCheckButton containing a label. The label will be created using `gtk_label_new_with_mnemonic()`, so underscores in *label* indicate the mnemonic for the check button.

label : The text of the button, with an underscore in front of the mnemonic character
Returns : a new GtkCheckButton

Style Properties

The "indicator-size" style property

```
  "indicator-size"       gint                  : Read
```

Size of check or radio indicator.

Allowed values: >= 0

Default value: 13

The "indicator-spacing" style property

```
  "indicator-spacing"    gint                  : Read
```

Spacing around check or radio indicator.

Allowed values: >= 0

Default value: 2

See Also

GtkCheckMenuItem	add check buttons to your menus.
GtkButton	a more general button.
GtkToggleButton	GtkCheckButton's parent.
GtkRadioButton	group check buttons together.

<< GtkButton **GtkRadioButton >>**

GtkRadioButton

GtkRadioButton — A choice from multiple check buttons

Synopsis

```
#include <gtk/gtk.h>

            GtkRadioButton;
GtkWidget*  gtk_radio_button_new            (GSList *group);
GtkWidget*  gtk_radio_button_new_from_widget
                                            (GtkRadioButton *group);
GtkWidget*  gtk_radio_button_new_with_label (GSList *group,
                                             const gchar *label);
GtkWidget*  gtk_radio_button_new_with_label_from_widget
                                            (GtkRadioButton *group,
                                             const gchar *label);
GtkWidget*  gtk_radio_button_new_with_mnemonic
                                            (GSList *group,
                                             const gchar *label);
GtkWidget*  gtk_radio_button_new_with_mnemonic_from_widget
                                            (GtkRadioButton *group,
                                             const gchar *label);
#define     gtk_radio_button_group
void        gtk_radio_button_set_group      (GtkRadioButton *radio_button,
                                             GSList *group);
GSList*     gtk_radio_button_get_group      (GtkRadioButton *radio_button);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkWidget
               +----GtkContainer
                     +----GtkBin
```

```
                        +----GtkButton
                              +----GtkToggleButton
                                    +----GtkCheckButton
                                          +----GtkRadioButton
```

Implemented Interfaces

GtkRadioButton implements AtkImplementorIface.

Properties

```
  "group"               GtkRadioButton        : Write
```

Signal Prototypes

```
"group-changed"
            void        user_function      (GtkRadioButton *radiobutton,
                                            gpointer user_data);
```

Description

A single radio button performs the same basic function as a GtkCheckButton, as its position in the object hierarchy reflects. It is only when multiple radio buttons are grouped together that they become a different user interface component in their own right.

Every radio button is a member of some group of radio buttons. When one is selected, all other radio buttons in the same group are deselected. A GtkRadioButton is one way of giving the user a choice from many options.

Radio button widgets are created with `gtk_radio_button_new()`, passing NULL as the argument if this is the first radio button in a group. In subsequent calls, the group you wish to add this button to should be passed as an argument. Optionally, `gtk_radio_button_new_with_label()` can be used if you want a text label on the radio button.

Alternatively, when adding widgets to an existing group of radio buttons, use `gtk_radio_button_new_from_widget()` with a GtkRadioButton that already has a group assigned to it. The convenience function `gtk_radio_button_new_with_label_from_widget()` is also provided.

To retrieve the group a GtkRadioButton is assigned to, use `gtk_radio_button_get_group()`.

To remove a GtkRadioButton from one group and make it part of a new one, use `gtk_radio_button_set_group()`.

The group list does not need to be freed, as each GtkRadioButton will remove itself and its list item when it is destroyed.

Example 1. How to create a group of two radio buttons.

```
void create_radio_buttons (void) {

   GtkWidget *window, *radio1, *radio2, *box, *entry;
   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
   box = gtk_vbox_new (TRUE, 2);

   /* Create a radio button with a GtkEntry widget */
   radio1 = gtk_radio_button_new (NULL);
   entry = gtk_entry_new ();
   gtk_container_add (GTK_CONTAINER (radio1), entry);


   /* Create a radio button with a label */
   radio2 = gtk_radio_button_new_with_label_from_widget (GTK_RADIO_BUTTON (radio1),
                                                         "I'm the second radio
button.");

   /* Pack them into a box, then show all the widgets */
   gtk_box_pack_start (GTK_BOX (box), radio1, TRUE, TRUE, 2);
   gtk_box_pack_start (GTK_BOX (box), radio2, TRUE, TRUE, 2);
   gtk_container_add (GTK_CONTAINER (window), box);
   gtk_widget_show_all (window);
   return;
}
```

Details

GtkRadioButton

```
typedef struct _GtkRadioButton GtkRadioButton;
```

Contains only private data that should be read and manipulated using the functions below.

---

gtk_radio_button_new ()

```
GtkWidget*  gtk_radio_button_new           (GSList *group);
```

Creates a new GtkRadioButton. To be of any practical value, a widget should then be packed into the radio button.

group : an existing radio button group, or NULL if you are creating a new group.

Returns : a new radio button.

---

gtk_radio_button_new_from_widget ()

```
GtkWidget*  gtk_radio_button_new_from_widget
                                            (GtkRadioButton *group);
```

Creates a new GtkRadioButton, adding it to the same group as *group*. As with gtk_radio_button_new(), a widget should be packed into the radio button.

group : an existing GtkRadioButton.

Returns : a new radio button.

---

gtk_radio_button_new_with_label ()

```
GtkWidget*  gtk_radio_button_new_with_label (GSList *group,
                                             const gchar *label);
```

Creates a new GtkRadioButton with a text label.

group : an existing radio button group, or NULL if you are creating a new group.

label : the text label to display next to the radio button.

Returns : a new radio button.

---

gtk_radio_button_new_with_label_from_widget ()

```
GtkWidget*  gtk_radio_button_new_with_label_from_widget
                                            (GtkRadioButton *group,
                                             const gchar *label);
```

Creates a new GtkRadioButton with a text label, adding it to the same group as *group*.

group : an existing GtkRadioButton.

label : a text string to display next to the radio button.

Returns : a new radio button.

---

gtk_radio_button_new_with_mnemonic ()

```
GtkWidget*  gtk_radio_button_new_with_mnemonic
                                              (GSList *group,
                                               const gchar *label);
```

Creates a new GtkRadioButton containing a label, adding it to the same group as *group*. The label will be created using gtk_label_new_with_mnemonic(), so underscores in *label* indicate the mnemonic for the button.

group : the radio button group
label : the text of the button, with an underscore in front of the mnemonic character
Returns : a new GtkRadioButton

---

gtk_radio_button_new_with_mnemonic_from_widget ()

```
GtkWidget*  gtk_radio_button_new_with_mnemonic_from_widget
                                              (GtkRadioButton *group,
                                               const gchar *label);
```

Creates a new GtkRadioButton containing a label. The label will be created using gtk_label_new_with_mnemonic(), so underscores in *label* indicate the mnemonic for the button.

group : widget to get radio group from
label : the text of the button, with an underscore in front of the mnemonic character
Returns : a new GtkRadioButton

---

gtk_radio_button_group

```
#define gtk_radio_button_group gtk_radio_button_get_group
```

Warning

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

Deprecated compatibility macro. Use gtk_radio_button_get_group() instead.

---

gtk_radio_button_set_group ()

```
void        gtk_radio_button_set_group      (GtkRadioButton *radio_button,
                                             GSList *group);
```

Sets a GtkRadioButton's group. It should be noted that this does not change the layout of your interface in any way, so if you are changing the group, it is likely you will need to re-arrange the user interface to reflect these changes.

radio_button : a GtkRadioButton.

group : an existing radio button group, such as one returned from gtk_radio_button_get_group().

---

gtk_radio_button_get_group ()

```
GSList*     gtk_radio_button_get_group      (GtkRadioButton *radio_button);
```

Retrieves the group assigned to a radio button.

radio_button : a GtkRadioButton.

Returns : a linked list containing all the radio buttons in the same group as *radio_button*.

Properties

The "group" property

```
  "group"               GtkRadioButton        : Write
```

Sets a new group for a radio button.

Signals

The "group-changed" signal

```
void        user_function                   (GtkRadioButton *radiobutton,
                                             gpointer user_data);
```

radiobutton : the object which received the signal.
user_data : user data set when the signal handler was connected.

See Also

GtkOptionMenu Another way of offering the user a single choice from many.

<< GtkCheckButton **GtkToggleButton >>**

GtkToggleButton

GtkToggleButton — Create buttons which retain their state

Synopsis

```
#include <gtk/gtk.h>


            GtkToggleButton;
GtkWidget*  gtk_toggle_button_new           (void);
GtkWidget*  gtk_toggle_button_new_with_label
                                            (const gchar *label);
GtkWidget*  gtk_toggle_button_new_with_mnemonic
                                            (const gchar *label);
void        gtk_toggle_button_set_mode      (GtkToggleButton *toggle_button,
                                             gboolean draw_indicator);
gboolean    gtk_toggle_button_get_mode      (GtkToggleButton *toggle_button);
#define     gtk_toggle_button_set_state
void        gtk_toggle_button_toggled       (GtkToggleButton *toggle_button);
gboolean    gtk_toggle_button_get_active    (GtkToggleButton *toggle_button);
void        gtk_toggle_button_set_active    (GtkToggleButton *toggle_button,
                                             gboolean is_active);
gboolean    gtk_toggle_button_get_inconsistent
                                            (GtkToggleButton *toggle_button);
void        gtk_toggle_button_set_inconsistent
                                            (GtkToggleButton *toggle_button,
                                             gboolean setting);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkWidget
               +----GtkContainer
```

```
                    +----GtkBin
                          +----GtkButton
                                +----GtkToggleButton
                                      +----GtkCheckButton
```

Implemented Interfaces

GtkToggleButton implements AtkImplementorIface.

Properties

```
  "active"               gboolean             : Read / Write
  "draw-indicator"       gboolean             : Read / Write
  "inconsistent"         gboolean             : Read / Write
```

Signal Prototypes

```
"toggled"   void        user_function      (GtkToggleButton *togglebutton,
                                            gpointer user_data);
```

Description

A GtkToggleButton is a GtkButton which will remain 'pressed-in' when clicked. Clicking again will cause the toggle button to return to its normal state.

A toggle button is created by calling either `gtk_toggle_button_new()` or `gtk_toggle_button_new_with_label()`. If using the former, it is advisable to pack a widget, (such as a GtkLabel and/or a GtkPixmap), into the toggle button's container. (See GtkButton for more information).

The state of a GtkToggleButton can be set specifically using `gtk_toggle_button_set_active()`, and retrieved using `gtk_toggle_button_get_active()`.

To simply switch the state of a toggle button, use gtk_toggle_button_toggled.

Example 2. Creating two GtkToggleButton widgets.

```
void make_toggles (void) {
   GtkWidget *dialog, *toggle1, *toggle2;
```

```
   dialog = gtk_dialog_new ();
   toggle1 = gtk_toggle_button_new_with_label ("Hi, i'm a toggle button.");

   /* Makes this toggle button invisible */
   gtk_toggle_button_set_mode (GTK_TOGGLE_BUTTON (toggle1), TRUE);

   g_signal_connect (toggle1, "toggled",
                     G_CALLBACK (output_state), NULL);
   gtk_box_pack_start (GTK_BOX (GTK_DIALOG (dialog)->action_area),
                       toggle1, FALSE, FALSE, 2);

   toggle2 = gtk_toggle_button_new_with_label ("Hi, i'm another toggle button.");
   gtk_toggle_button_set_mode (GTK_TOGGLE_BUTTON (toggle2), FALSE);
   g_signal_connect (toggle2, "toggled",
                     G_CALLBACK (output_state), NULL);
   gtk_box_pack_start (GTK_BOX (GTK_DIALOG (dialog)->action_area),
                       toggle2, FALSE, FALSE, 2);

   gtk_widget_show_all (dialog);
}
```

Details

GtkToggleButton

```
typedef struct _GtkToggleButton GtkToggleButton;
```

The GtkToggleButton struct contains private data only, and should be manipulated using the functions below.

---

gtk_toggle_button_new ()

```
GtkWidget*  gtk_toggle_button_new           (void);
```

Creates a new toggle button. A widget should be packed into the button, as in `gtk_button_new()`.

Returns : a new toggle button.

---

gtk_toggle_button_new_with_label ()

```
GtkWidget*  gtk_toggle_button_new_with_label
                                            (const gchar *label);
```

Creates a new toggle button with a text label.

label : a string containing the message to be placed in the toggle button.
Returns : a new toggle button.

---

gtk_toggle_button_new_with_mnemonic ()

```
GtkWidget*  gtk_toggle_button_new_with_mnemonic
                                            (const gchar *label);
```

Creates a new GtkToggleButton containing a label. The label will be created using gtk_label_new_with_mnemonic(), so underscores in *label* indicate the mnemonic for the button.

label : the text of the button, with an underscore in front of the mnemonic character
Returns : a new GtkToggleButton

---

gtk_toggle_button_set_mode ()

```
void        gtk_toggle_button_set_mode      (GtkToggleButton *toggle_button,
                                             gboolean draw_indicator);
```

Sets whether the button is displayed as a separate indicator and label. You can call this function on a checkbutton or a radiobutton with *draw_indicator* = FALSE to make the button look like a normal button

This function only effects instances of classes like GtkCheckButton and GtkRadioButton that derive from GtkToggleButton, not instances of GtkToggleButton itself.

toggle_button : a GtkToggleButton
draw_indicator : if TRUE, draw the button as a separate indicator and label; if FALSE, draw the button like a normal button

---

gtk_toggle_button_get_mode ()

```
gboolean    gtk_toggle_button_get_mode      (GtkToggleButton *toggle_button);
```

Retrieves whether the button is displayed as a separate indicator and label. See `gtk_toggle_button_set_mode()`.

`toggle_button` : a GtkToggleButton
Returns : TRUE if the togglebutton is drawn as a separate indicator and label.

---

gtk_toggle_button_set_state

```
#define gtk_toggle_button_set_state             gtk_toggle_button_set_active
```

Warning

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

This is a deprecated macro, and is only maintained for compatibility reasons.

---

gtk_toggle_button_toggled ()

```
void        gtk_toggle_button_toggled       (GtkToggleButton *toggle_button);
```

Emits the toggled signal on the GtkToggleButton. There is no good reason for an application ever to call this function.

`toggle_button` : a GtkToggleButton.

---

gtk_toggle_button_get_active ()

```
gboolean    gtk_toggle_button_get_active    (GtkToggleButton *toggle_button);
```

Queries a GtkToggleButton and returns its current state. Returns TRUE if the toggle button is pressed in and FALSE if it is raised.

`toggle_button` : a GtkToggleButton.
Returns : a gboolean value.

---

gtk_toggle_button_set_active ()

```
void        gtk_toggle_button_set_active    (GtkToggleButton *toggle_button,
                                             gboolean is_active);
```

Sets the status of the toggle button. Set to TRUE if you want the GtkToggleButton to be 'pressed in', and FALSE to raise it. This action causes the toggled signal to be emitted.

toggle_button : a GtkToggleButton.
is_active : TRUE or FALSE.

---

gtk_toggle_button_get_inconsistent ()

```
gboolean    gtk_toggle_button_get_inconsistent
                                            (GtkToggleButton *toggle_button);
```

Gets the value set by gtk_toggle_button_set_inconsistent().

toggle_button : a GtkToggleButton
Returns : TRUE if the button is displayed as inconsistent, FALSE otherwise

---

gtk_toggle_button_set_inconsistent ()

```
void        gtk_toggle_button_set_inconsistent
                                            (GtkToggleButton *toggle_button,
                                             gboolean setting);
```

If the user has selected a range of elements (such as some text or spreadsheet cells) that are affected by a toggle button, and the current values in that range are inconsistent, you may want to display the toggle in an "in between" state. This function turns on "in between" display. Normally you would turn off the inconsistent state again if the user toggles the toggle button. This has to be done manually, gtk_toggle_button_set_inconsistent() only affects visual appearance, it doesn't affect the semantics of the button.

toggle_button : a GtkToggleButton
setting : TRUE if state is inconsistent

Properties

The "active" property

```
  "active"              gboolean              : Read / Write
```

If the toggle button should be pressed in or not.

Default value: FALSE

The "draw-indicator" property

```
  "draw-indicator"      gboolean              : Read / Write
```

If the toggle part of the button is displayed.

Default value: FALSE

The "inconsistent" property

```
  "inconsistent"        gboolean              : Read / Write
```

If the toggle button is in an "in between" state.

Default value: FALSE

Signals

The "toggled" signal

```
void        user_function                 (GtkToggleButton *togglebutton,
                                           gpointer user_data);
```

Should be connected if you wish to perform an action whenever the GtkToggleButton's state is changed.

togglebutton : the object which received the signal.
user_data : user data set when the signal handler was connected.

See Also

GtkButton	a more general button.
GtkCheckButton	another way of presenting a toggle option.
GtkCheckMenuItem	a GtkToggleButton as a menu item.

<< GtkRadioButton **Numeric/Text Data Entry >>**

Numeric/Text Data Entry

GtkEntry - A single line text entry field
GtkEntryCompletion - Completion functionality for GtkEntry
GtkHScale - A horizontal slider widget for selecting a value from a range
GtkVScale - A vertical slider widget for selecting a value from a range
GtkSpinButton - Retrieve an integer or floating-point number from the user
GtkEditable - Interface for text-editing widgets

<< GtkToggleButton **GtkEntry >>**

GtkEntry

GtkEntry — A single line text entry field

Synopsis

```
#include <gtk/gtk.h>


            GtkEntry;
GtkWidget*  gtk_entry_new                   (void);
GtkWidget*  gtk_entry_new_with_max_length   (gint max);
void        gtk_entry_set_text              (GtkEntry *entry,
                                             const gchar *text);
void        gtk_entry_append_text           (GtkEntry *entry,
                                             const gchar *text);
void        gtk_entry_prepend_text          (GtkEntry *entry,
                                             const gchar *text);
void        gtk_entry_set_position          (GtkEntry *entry,
                                             gint position);
G_CONST_RETURN gchar* gtk_entry_get_text    (GtkEntry *entry);
void        gtk_entry_select_region         (GtkEntry *entry,
                                             gint start,
                                             gint end);
void        gtk_entry_set_visibility        (GtkEntry *entry,
                                             gboolean visible);
void        gtk_entry_set_invisible_char    (GtkEntry *entry,
                                             gunichar ch);
void        gtk_entry_set_editable          (GtkEntry *entry,
                                             gboolean editable);
void        gtk_entry_set_max_length        (GtkEntry *entry,
                                             gint max);
gboolean    gtk_entry_get_activates_default (GtkEntry *entry);
gboolean    gtk_entry_get_has_frame         (GtkEntry *entry);
gint        gtk_entry_get_width_chars       (GtkEntry *entry);
void        gtk_entry_set_activates_default (GtkEntry *entry,
                                             gboolean setting);
```

```
void        gtk_entry_set_has_frame         (GtkEntry *entry,
                                             gboolean setting);
void        gtk_entry_set_width_chars       (GtkEntry *entry,
                                             gint n_chars);
gunichar    gtk_entry_get_invisible_char    (GtkEntry *entry);
void        gtk_entry_set_alignment         (GtkEntry *entry,
                                             gfloat xalign);
gfloat      gtk_entry_get_alignment         (GtkEntry *entry);
PangoLayout* gtk_entry_get_layout           (GtkEntry *entry);
void        gtk_entry_get_layout_offsets    (GtkEntry *entry,
                                             gint *x,
                                             gint *y);
gint        gtk_entry_layout_index_to_text_index
                                            (GtkEntry *entry,
                                             gint layout_index);
gint        gtk_entry_text_index_to_layout_index
                                            (GtkEntry *entry,
                                             gint text_index);
gint        gtk_entry_get_max_length        (GtkEntry *entry);
gboolean    gtk_entry_get_visibility        (GtkEntry *entry);
void        gtk_entry_set_completion        (GtkEntry *entry,
                                             GtkEntryCompletion *completion);
GtkEntryCompletion* gtk_entry_get_completion
                                            (GtkEntry *entry);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkWidget
               +----GtkEntry
                     +----GtkSpinButton
```

Implemented Interfaces

GtkEntry implements AtkImplementorIface, GtkCellEditable and GtkEditable.

Properties

```
  "activates-default"    gboolean             : Read / Write
  "cursor-position"      gint                 : Read
  "editable"             gboolean             : Read / Write
  "has-frame"            gboolean             : Read / Write
  "invisible-char"       guint                : Read / Write
  "max-length"           gint                 : Read / Write
  "scroll-offset"        gint                 : Read
  "selection-bound"      gint                 : Read
  "text"                 gchararray           : Read / Write
  "visibility"           gboolean             : Read / Write
  "width-chars"          gint                 : Read / Write
  "xalign"               gfloat               : Read / Write
```

Signal Prototypes

```
"activate"  void        user_function      (GtkEntry *entry,
                                            gpointer user_data);
"backspace" void        user_function      (GtkEntry *entry,
                                            gpointer user_data);
"copy-clipboard"
            void        user_function      (GtkEntry *entry,
                                            gpointer user_data);
"cut-clipboard"
            void        user_function      (GtkEntry *entry,
                                            gpointer user_data);
"delete-from-cursor"
            void        user_function      (GtkEntry *entry,
                                            GtkDeleteType arg1,
                                            gint arg2,
                                            gpointer user_data);
"insert-at-cursor"
            void        user_function      (GtkEntry *entry,
                                            gchar *arg1,
                                            gpointer user_data);
"move-cursor"
            void        user_function      (GtkEntry *entry,
```

```
                                                GtkMovementStep arg1,
                                                gint arg2,
                                                gboolean arg3,
                                                gpointer user_data);
"paste-clipboard"
            void        user_function      (GtkEntry *entry,
                                                gpointer user_data);
"populate-popup"
            void        user_function      (GtkEntry *entry,
                                                GtkMenu *arg1,
                                                gpointer user_data);
"toggle-overwrite"
            void        user_function      (GtkEntry *entry,
                                                gpointer user_data);
```

Description

The GtkEntry widget is a single line text entry widget. A fairly large set of key bindings are supported by default. If the entered text is longer than the allocation of the widget, the widget will scroll so that the cursor position is visible.

Details

GtkEntry

```
typedef struct _GtkEntry GtkEntry;
```

The GtkEntry-struct struct contains only private data.

---

gtk_entry_new ()

```
GtkWidget*  gtk_entry_new                       (void);
```

Creates a new GtkEntry widget.

Returns : a new GtkEntry.

---

gtk_entry_new_with_max_length ()

```
GtkWidget*  gtk_entry_new_with_max_length   (gint max);
```

Warning

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

Creates a new GtkEntry widget with the given maximum length.

Note: the existence of this function is inconsistent with the rest of the GTK+ API. The normal setup would be to just require the user to make an extra call to `gtk_entry_set_max_length()` instead. It is not expected that this function will be removed, but it would be better practice not to use it.

max : the maximum length of the entry, or 0 for no maximum. (other than the maximum length of entries.) The value passed in will be clamped to the range 0-65536.

Returns : a new GtkEntry.

---

gtk_entry_set_text ()

```
void        gtk_entry_set_text              (GtkEntry *entry,
                                             const gchar *text);
```

Sets the text in the widget to the given value, replacing the current contents.

entry : a GtkEntry.

text : the new text.

---

gtk_entry_append_text ()

```
void        gtk_entry_append_text           (GtkEntry *entry,
                                             const gchar *text);
```

Warning

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

Appends the given text to the contents of the widget.

`entry`: a GtkEntry.
`text`: the text to append.

---

gtk_entry_prepend_text ()

```
void        gtk_entry_prepend_text          (GtkEntry *entry,
                                             const gchar *text);
```

Warning

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

Prepends the given text to the contents of th ewidget.

`entry`: a GtkEntry.
`text`: the text to prepend.

---

gtk_entry_set_position ()

```
void        gtk_entry_set_position          (GtkEntry *entry,
                                             gint position);
```

Warning

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

Sets the cursor position in an entry to the given value. This function is obsolete. You should use `gtk_editable_set_position()` instead.

`entry`: a GtkEntry.

position : the position of the cursor. The cursor is displayed before the character with the given (base 0) index in the widget. The value must be less than or equal to the number of characters in the widget. A value of -1 indicates that the position should be set after the last character in the entry. Note that this position is in characters, not in bytes.

---

gtk_entry_get_text ()

```
G_CONST_RETURN gchar* gtk_entry_get_text    (GtkEntry *entry);
```

Retrieves the contents of the entry widget. See also `gtk_editable_get_chars()`.

entry : a GtkEntry

Returns : a pointer to the contents of the widget as a string. This string points to internally allocated storage in the widget and must not be freed, modified or stored.

---

gtk_entry_select_region ()

```
void        gtk_entry_select_region         (GtkEntry *entry,
                                             gint start,
                                             gint end);
```

Warning

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

Selects a region of text. The characters that are selected are those characters at positions from *start_pos* up to, but not including *end_pos*. If *end_pos* is negative, then the the characters selected will be those characters from *start_pos* to the end of the text. This function is obsolete. You should use `gtk_editable_select_region()` instead.

entry : a GtkEntry.

start : the starting position.

end : the end position.

---

gtk_entry_set_visibility ()

```
void        gtk_entry_set_visibility        (GtkEntry *entry,
                                             gboolean visible);
```

Sets whether the contents of the entry are visible or not. When visibility is set to FALSE, characters are displayed as the invisible char, and will also appear that way when the text in the entry widget is copied elsewhere.

The default invisible char is the asterisk '*', but it can be changed with `gtk_entry_set_invisible_char()`.

entry : a GtkEntry.

visible : TRUE if the contents of the entry are displayed as plaintext.

---

gtk_entry_set_invisible_char ()

```
void        gtk_entry_set_invisible_char    (GtkEntry *entry,
                                             gunichar ch);
```

Sets the character to use in place of the actual text when `gtk_entry_set_visibility()` has been called to set text visibility to FALSE. i.e. this is the character used in "password mode" to show the user how many characters have been typed. The default invisible char is an asterisk ('*'). If you set the invisible char to 0, then the user will get no feedback at all; there will be no text on the screen as they type.

entry : a GtkEntry

ch : a Unicode character

---

gtk_entry_set_editable ()

```
void        gtk_entry_set_editable          (GtkEntry *entry,
                                             gboolean editable);
```

Warning

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

Determines if the user can edit the text in the editable widget or not. This function is obsolete. You should use `gtk_editable_set_editable()` instead.

entry : a GtkEntry.

editable : TRUE if the user is allowed to edit the text in the widget.

---

gtk_entry_set_max_length ()

```
void        gtk_entry_set_max_length        (GtkEntry *entry,
                                             gint max);
```

Sets the maximum allowed length of the contents of the widget. If the current contents are longer than the given length, then they will be truncated to fit.

entry : a GtkEntry.

max : the maximum length of the entry, or 0 for no maximum. (other than the maximum length of entries.) The value passed in will be clamped to the range 0-65536.

---

gtk_entry_get_activates_default ()

```
gboolean    gtk_entry_get_activates_default (GtkEntry *entry);
```

Retrieves the value set by gtk_entry_set_activates_default().

entry : a GtkEntry

Returns : TRUE if the entry will activate the default widget

---

gtk_entry_get_has_frame ()

```
gboolean    gtk_entry_get_has_frame         (GtkEntry *entry);
```

Gets the value set by gtk_entry_set_has_frame().

entry : a GtkEntry

Returns : whether the entry has a beveled frame

---

gtk_entry_get_width_chars ()

```
gint        gtk_entry_get_width_chars       (GtkEntry *entry);
```

Gets the value set by gtk_entry_set_width_chars().

entry : a GtkEntry

Returns : number of chars to request space for, or negative if unset

---

gtk_entry_set_activates_default ()

```
void        gtk_entry_set_activates_default (GtkEntry *entry,
                                             gboolean setting);
```

If *setting* is TRUE, pressing Enter in the *entry* will activate the default widget for the window containing the entry. This usually means that the dialog box containing the entry will be closed, since the default widget is usually one of the dialog buttons.

(For experts: if *setting* is TRUE, the entry calls gtk_window_activate_default() on the window containing the entry, in the default handler for the "activate" signal.)

entry : a GtkEntry

setting : TRUE to activate window's default widget on Enter keypress

---

gtk_entry_set_has_frame ()

```
void        gtk_entry_set_has_frame         (GtkEntry *entry,
                                             gboolean setting);
```

Sets whether the entry has a beveled frame around it.

entry : a GtkEntry

setting : new value

---

gtk_entry_set_width_chars ()

```
void        gtk_entry_set_width_chars       (GtkEntry *entry,
                                             gint n_chars);
```

Changes the size request of the entry to be about the right size for *n_chars* characters. Note that it changes the size *request*, the size can still be affected by how you pack the widget into containers. If *n_chars* is -1, the size reverts to the default entry size.

entry : a GtkEntry

n_chars : width in chars

---

gtk_entry_get_invisible_char ()

```
gunichar    gtk_entry_get_invisible_char    (GtkEntry *entry);
```

Retrieves the character displayed in place of the real characters for entries with visisbility set to false. See gtk_entry_set_invisible_char().

entry : a GtkEntry

Returns : the current invisible char, or 0, if the entry does not show invisible text at all.

---

gtk_entry_set_alignment ()

```
void        gtk_entry_set_alignment         (GtkEntry *entry,
                                             gfloat xalign);
```

Sets the alignment for the contents of the entry. This controls the horizontal positioning of the contents when the displayed text is shorter than the width of the entry.

entry : a GtkEntry

xalign : The horizontal alignment, from 0 (left) to 1 (right). Reversed for RTL layouts

Since 2.4

---

gtk_entry_get_alignment ()

```
gfloat      gtk_entry_get_alignment         (GtkEntry *entry);
```

Gets the value set by gtk_entry_set_alignment().

entry : a GtkEntry

Returns : the alignment

Since 2.4

---

gtk_entry_get_layout ()

```
PangoLayout* gtk_entry_get_layout           (GtkEntry *entry);
```

Gets the PangoLayout used to display the entry. The layout is useful to e.g. convert text positions to pixel positions, in combination with gtk_entry_get_layout_offsets(). The returned layout is owned by the entry so need not be freed by the caller.

Keep in mind that the layout text may contain a preedit string, so gtk_entry_layout_index_to_text_index() and gtk_entry_text_index_to_layout_index() are needed to convert byte indices in the layout to byte indices in the entry contents.

entry : a GtkEntry

Returns : the PangoLayout for this entry

---

gtk_entry_get_layout_offsets ()

```
void        gtk_entry_get_layout_offsets    (GtkEntry *entry,
                                             gint *x,
                                             gint *y);
```

Obtains the position of the PangoLayout used to render text in the entry, in widget coordinates. Useful if you want to line up the text in an entry with some other text, e.g. when using the entry to implement editable cells in a sheet widget.

Also useful to convert mouse events into coordinates inside the PangoLayout, e.g. to take some action if some part of the entry text is clicked.

Note that as the user scrolls around in the entry the offsets will change; you'll need to connect to the "notify::scroll-offset" signal to track this. Remember when using the PangoLayout functions you need to convert to and from pixels using `PANGO_PIXELS()` or PANGO_SCALE.

Keep in mind that the layout text may contain a preedit string, so `gtk_entry_layout_index_to_text_index()` and `gtk_entry_text_index_to_layout_index()` are needed to convert byte indices in the layout to byte indices in the entry contents.

entry : a GtkEntry
x : location to store X offset of layout, or `NULL`
y : location to store Y offset of layout, or `NULL`

---

gtk_entry_layout_index_to_text_index ()

```
gint        gtk_entry_layout_index_to_text_index
                                            (GtkEntry *entry,
                                             gint layout_index);
```

Converts from a position in the entry contents (returned by `gtk_entry_get_text()`) to a position in the entry's PangoLayout (returned by `gtk_entry_get_layout()`, with text retrieved via `pango_layout_get_text()`).

entry : a GtkEntry
layout_index : byte index into the entry layout text
Returns : byte index into the entry contents

---

gtk_entry_text_index_to_layout_index ()

```
gint        gtk_entry_text_index_to_layout_index
                                            (GtkEntry *entry,
```

```
                                                gint text_index);
```

Converts from a position in the entry's PangoLayout (returned by `gtk_entry_get_layout()`) to a position in the entry contents (returned by `gtk_entry_get_text()`).

entry : a GtkEntry
text_index : byte index into the entry contents
Returns : byte index into the entry layout text

---

gtk_entry_get_max_length ()

```
gint        gtk_entry_get_max_length        (GtkEntry *entry);
```

Retrieves the maximum allowed length of the text in *entry*. See `gtk_entry_set_max_length()`.

entry : a GtkEntry
Returns : the maximum allowed number of characters in GtkEntry, or 0 if there is no maximum.

---

gtk_entry_get_visibility ()

```
gboolean    gtk_entry_get_visibility        (GtkEntry *entry);
```

Retrieves whether the text in *entry* is visible. See `gtk_entry_set_visibility()`.

entry : a GtkEntry
Returns : TRUE if the text is currently visible

---

gtk_entry_set_completion ()

```
void        gtk_entry_set_completion        (GtkEntry *entry,
                                             GtkEntryCompletion *completion);
```

Sets *completion* to be the auxiliary completion object to use with *entry*. All further configuration of the

completion mechanism is done on *completion* using the GtkEntryCompletion API.

entry : A GtkEntry.
completion : The GtkEntryCompletion.

Since 2.4

---

gtk_entry_get_completion ()

```
GtkEntryCompletion* gtk_entry_get_completion
                                            (GtkEntry *entry);
```

Returns the auxiliary completion object currently in use by *entry*.

entry : A GtkEntry.
Returns : The auxiliary completion object currently in use by *entry*.

Since 2.4

Properties

The "activates-default" property

```
  "activates-default"    gboolean              : Read / Write
```

Whether to activate the default widget (such as the default button in a dialog) when Enter is pressed.

Default value: FALSE

The "cursor-position" property

```
  "cursor-position"      gint                  : Read
```

The current position of the insertion cursor in chars.

Allowed values: [0,65535]

Default value: 0

The "editable" property

```
"editable"            gboolean              : Read / Write
```

Whether the entry contents can be edited.

Default value: TRUE

The "has-frame" property

```
"has-frame"           gboolean              : Read / Write
```

FALSE removes outside bevel from entry.

Default value: TRUE

The "invisible-char" property

```
"invisible-char"      guint                 : Read / Write
```

The character to use when masking entry contents (in "password mode").

Default value: '*'

The "max-length" property

```
"max-length"           gint                : Read / Write
```

Maximum number of characters for this entry. Zero if no maximum.

Allowed values: [0,65535]

Default value: 0

The "scroll-offset" property

```
"scroll-offset"        gint                : Read
```

Number of pixels of the entry scrolled off the screen to the left.

Allowed values: >= 0

Default value: 0

The "selection-bound" property

```
"selection-bound"      gint                : Read
```

The position of the opposite end of the selection from the cursor in chars.

Allowed values: [0,65535]

Default value: 0

The "text" property

```
"text"                 gchararray          : Read / Write
```

The contents of the entry.

Default value: ""

The "visibility" property

```
"visibility"          gboolean              : Read / Write
```

FALSE displays the "invisible char" instead of the actual text (password mode).

Default value: TRUE

The "width-chars" property

```
"width-chars"         gint                  : Read / Write
```

Number of characters to leave space for in the entry.

Allowed values: >= -1

Default value: -1

The "xalign" property

```
"xalign"              gfloat                : Read / Write
```

The horizontal alignment, from 0 (left) to 1 (right). Reversed for RTL layouts.

Allowed values: [0,1]

Default value: 0

Since 2.4

Signals

The "activate" signal

```
void        user_function                   (GtkEntry *entry,
                                             gpointer user_data);
```

entry : the object which received the signal.
user_data : user data set when the signal handler was connected.

The "backspace" signal

```
void        user_function                   (GtkEntry *entry,
                                             gpointer user_data);
```

entry : the object which received the signal.
user_data : user data set when the signal handler was connected.

The "copy-clipboard" signal

```
void        user_function                   (GtkEntry *entry,
                                             gpointer user_data);
```

entry : the object which received the signal.
user_data : user data set when the signal handler was connected.

The "cut-clipboard" signal

```
void        user_function                   (GtkEntry *entry,
                                             gpointer user_data);
```

entry : the object which received the signal.
user_data : user data set when the signal handler was connected.

The "delete-from-cursor" signal

```
void        user_function                   (GtkEntry *entry,
                                             GtkDeleteType arg1,
                                             gint arg2,
                                             gpointer user_data);
```

entry : the object which received the signal.
arg1 :
arg2 :
user_data : user data set when the signal handler was connected.

The "insert-at-cursor" signal

```
void        user_function                   (GtkEntry *entry,
                                             gchar *arg1,
                                             gpointer user_data);
```

entry : the object which received the signal.
arg1 :
user_data : user data set when the signal handler was connected.

The "move-cursor" signal

```
void        user_function                   (GtkEntry *entry,
                                             GtkMovementStep arg1,
                                             gint arg2,
                                             gboolean arg3,
                                             gpointer user_data);
```

entry : the object which received the signal.
arg1 :
arg2 :

arg3 :
user_data : user data set when the signal handler was connected.

The "paste-clipboard" signal

```
void        user_function                   (GtkEntry *entry,
                                             gpointer user_data);
```

entry : the object which received the signal.
user_data : user data set when the signal handler was connected.

The "populate-popup" signal

```
void        user_function                   (GtkEntry *entry,
                                             GtkMenu *arg1,
                                             gpointer user_data);
```

entry : the object which received the signal.
arg1 :
user_data : user data set when the signal handler was connected.

The "toggle-overwrite" signal

```
void        user_function                   (GtkEntry *entry,
                                             gpointer user_data);
```

entry : the object which received the signal.
user_data : user data set when the signal handler was connected.

See Also

GtkText a widget for handling multi-line text entry.

<< Numeric/Text Data Entry **GtkEntryCompletion >>**

GtkEntryCompletion

GtkEntryCompletion — Completion functionality for GtkEntry

Synopsis

```
#include <gtk/gtk.h>

            GtkEntryCompletion;
gboolean    (*GtkEntryCompletionMatchFunc)  (GtkEntryCompletion *completion,
                                             const gchar *key,
                                             GtkTreeIter *iter,
                                             gpointer user_data);
GtkEntryCompletion* gtk_entry_completion_new
                                            (void);
GtkWidget*  gtk_entry_completion_get_entry  (GtkEntryCompletion *completion);
void        gtk_entry_completion_set_model  (GtkEntryCompletion *completion,
                                             GtkTreeModel *model);
GtkTreeModel* gtk_entry_completion_get_model
                                            (GtkEntryCompletion *completion);
void        gtk_entry_completion_set_match_func
                                            (GtkEntryCompletion *completion,
                                             GtkEntryCompletionMatchFunc func,
                                             gpointer func_data,
                                             GDestroyNotify func_notify);
void        gtk_entry_completion_set_minimum_key_length
                                            (GtkEntryCompletion *completion,
                                             gint length);
gint        gtk_entry_completion_get_minimum_key_length
                                            (GtkEntryCompletion *completion);
void        gtk_entry_completion_complete   (GtkEntryCompletion *completion);
void        gtk_entry_completion_insert_prefix
                                            (GtkEntryCompletion *completion);
void        gtk_entry_completion_insert_action_text
                                            (GtkEntryCompletion *completion,
                                             gint index_,
```

```
                                                const gchar *text);
void        gtk_entry_completion_insert_action_markup
                                               (GtkEntryCompletion *completion,
                                                gint index_,
                                                const gchar *markup);
void        gtk_entry_completion_delete_action
                                               (GtkEntryCompletion *completion,
                                                gint index_);
void        gtk_entry_completion_set_text_column
                                               (GtkEntryCompletion *completion,
                                                gint column);
gint        gtk_entry_completion_get_text_column
                                               (GtkEntryCompletion *completion);
void        gtk_entry_completion_set_inline_completion
                                               (GtkEntryCompletion *completion,
                                                gboolean inline_completion);
gboolean    gtk_entry_completion_get_inline_completion
                                               (GtkEntryCompletion *completion);
void        gtk_entry_completion_set_popup_completion
                                               (GtkEntryCompletion *completion,
                                                gboolean popup_completion);
gboolean    gtk_entry_completion_get_popup_completion
                                               (GtkEntryCompletion *completion);
```

Object Hierarchy

```
  GObject
   +----GtkEntryCompletion
```

Implemented Interfaces

GtkEntryCompletion implements GtkCellLayout.

Properties

```
  "inline-completion"    gboolean              : Read / Write
```

```
  "minimum-key-length"   gint                : Read / Write
  "model"                GtkTreeModel        : Read / Write
  "popup-completion"     gboolean            : Read / Write
  "text-column"          gint                : Read / Write
```

Signal Prototypes

```
"action-activated"
            void        user_function      (GtkEntryCompletion *widget,
                                            gint index,
                                            gpointer user_data);
"insert-prefix"
            gboolean    user_function      (GtkEntryCompletion *widget,
                                            gchar *prefix,
                                            gpointer user_data);
"match-selected"
            gboolean    user_function      (GtkEntryCompletion *widget,
                                            GtkTreeModel *model,
                                            GtkTreeIter *iter,
                                            gpointer user_data);
```

Description

GtkEntryCompletion is an auxiliary object to be used in conjunction with GtkEntry to provide the completion functionality. It implements the GtkCellLayout interface, to allow the user to add extra cells to the GtkTreeView with completion matches.

"Completion functionality" means that when the user modifies the text in the entry, GtkEntryCompletion checks which rows in the model match the current content of the entry, and displays a list of matches. By default, the matching is done by comparing the entry text case-insensitively against the text column of the model (see `gtk_entry_completion_set_text_column()`), but this can be overridden with a custom match function (see `gtk_entry_completion_set_match_func()`).

When the user selects a completion, the content of the entry is updated. By default, the content of the entry is replaced by the text column of the model, but this can be overridden by connecting to the ::match-selected signal and updating the entry in the signal handler. Note that you should return TRUE from the signal handler to suppress the default behaviour.

To add completion functionality to an entry, use `gtk_entry_set_completion()`.

In addition to regular completion matches, which will be inserted into the entry when they are selected,

GtkEntryCompletion also allows to display "actions" in the popup window. Their appearance is similar to menuitems, to differentiate them clearly from completion strings. When an action is selected, the ::action-activated signal is emitted.

Details

GtkEntryCompletion

```
typedef struct _GtkEntryCompletion GtkEntryCompletion;
```

The GtkEntryCompletion struct contains only private data.

GtkEntryCompletionMatchFunc ()

```
gboolean    (*GtkEntryCompletionMatchFunc)  (GtkEntryCompletion *completion,
                                             const gchar *key,
                                             GtkTreeIter *iter,
                                             gpointer user_data);
```

A function which decides whether the row indicated by *iter* matches a given *key*, and should be displayed as a possible completion for *key*. Note that *key* is normalized and case-folded (see g_utf8_normalize() and g_utf8_casefold()). If this is not appropriate, match functions have access to the unmodified key via gtk_entry_get_text (GTK_ENTRY (gtk_entry_completion_get_entry ())).

completion :	the GtkEntryCompletion
key :	the string to match, normalized and case-folded
iter :	a GtkTreeIter indicating the row to match
user_data :	user data given to gtk_entry_completion_set_match_func()
Returns :	TRUE if *iter* should be displayed as a possible completion for *key*

---

gtk_entry_completion_new ()

```
GtkEntryCompletion* gtk_entry_completion_new
                                            (void);
```

Creates a new GtkEntryCompletion object.

Returns : A newly created GtkEntryCompletion object.

Since 2.4

---

gtk_entry_completion_get_entry ()

```
GtkWidget*  gtk_entry_completion_get_entry  (GtkEntryCompletion *completion);
```

Gets the entry *completion* has been attached to.

completion : A GtkEntryCompletion.
Returns : The entry *completion* has been attached to.

Since 2.4

---

gtk_entry_completion_set_model ()

```
void        gtk_entry_completion_set_model  (GtkEntryCompletion *completion,
                                             GtkTreeModel *model);
```

Sets the model for a GtkEntryCompletion. If *completion* already has a model set, it will remove it before setting the new model. If model is NULL, then it will unset the model.

completion : A GtkEntryCompletion.
model : The GtkTreeModel.

Since 2.4

---

gtk_entry_completion_get_model ()

```
GtkTreeModel* gtk_entry_completion_get_model
                                                    (GtkEntryCompletion *completion);
```

Returns the model the GtkEntryCompletion is using as data source. Returns NULL if the model is unset.

completion : A GtkEntryCompletion.

Returns : A GtkTreeModel, or NULL if none is currently being used.

Since 2.4

---

gtk_entry_completion_set_match_func ()

```
void        gtk_entry_completion_set_match_func
                                                    (GtkEntryCompletion *completion,
                                                     GtkEntryCompletionMatchFunc func,
                                                     gpointer func_data,
                                                     GDestroyNotify func_notify);
```

Sets the match function for *completion* to be *func*. The match function is used to determine if a row should or should not be in the completion list.

completion : A GtkEntryCompletion.

func : The GtkEntryCompletionMatchFunc to use.

func_data : The user data for *func*.

func_notify : Destroy notifier for *func_data*.

Since 2.4.

---

gtk_entry_completion_set_minimum_key_length ()

```
void        gtk_entry_completion_set_minimum_key_length
                                                    (GtkEntryCompletion *completion,
                                                     gint length);
```

Requires the length of the search key for *completion* to be at least *length*. This is useful for long lists, where

completing using a small key takes a lot of time and will come up with meaningless results anyway (ie, a too large dataset).

completion : A GtkEntryCompletion.
length : The minimum length of the key in order to start completing.

Since 2.4

---

gtk_entry_completion_get_minimum_key_length ()

```
gint        gtk_entry_completion_get_minimum_key_length
                                            (GtkEntryCompletion *completion);
```

Returns the minimum key length as set for *completion*.

completion : A GtkEntryCompletion.
Returns : The currently used minimum key length.

Since 2.4

---

gtk_entry_completion_complete ()

```
void        gtk_entry_completion_complete   (GtkEntryCompletion *completion);
```

Requests a completion operation, or in other words a refiltering of the current list with completions, using the current key. The completion list view will be updated accordingly.

completion : A GtkEntryCompletion.

Since 2.4

---

gtk_entry_completion_insert_prefix ()

```
void        gtk_entry_completion_insert_prefix
                                            (GtkEntryCompletion *completion);
```

Requests a prefix insertion.

completion : a GtkEntryCompletion

Since 2.6

---

gtk_entry_completion_insert_action_text ()

```
void        gtk_entry_completion_insert_action_text
                                            (GtkEntryCompletion *completion,
                                             gint index_,
                                             const gchar *text);
```

Inserts an action in *completion*'s action item list at position *index_* with text *text*. If you want the action item to have markup, use gtk_entry_completion_insert_action_markup().

completion : A GtkEntryCompletion.
*index_* : The index of the item to insert.
text : Text of the item to insert.

Since 2.4

---

gtk_entry_completion_insert_action_markup ()

```
void        gtk_entry_completion_insert_action_markup
                                            (GtkEntryCompletion *completion,
                                             gint index_,
                                             const gchar *markup);
```

Inserts an action in *completion*'s action item list at position *index_* with markup *markup*.

completion : A GtkEntryCompletion.

*index_* : The index of the item to insert.
markup : Markup of the item to insert.

Since 2.4

---

gtk_entry_completion_delete_action ()

```
void        gtk_entry_completion_delete_action
                                            (GtkEntryCompletion *completion,
                                             gint index_);
```

Deletes the action at *index_* from *completion*'s action list.

completion : A GtkEntryCompletion.
*index_* : The index of the item to Delete.

Since 2.4

---

gtk_entry_completion_set_text_column ()

```
void        gtk_entry_completion_set_text_column
                                            (GtkEntryCompletion *completion,
                                             gint column);
```

Convenience function for setting up the most used case of this code: a completion list with just strings. This function will set up *completion* to have a list displaying all (and just) strings in the completion list, and to get those strings from *column* in the model of *completion*.

This functions creates and adds a GtkCellRendererText for the selected column. If you need to set the text column, but don't want the cell renderer, use `g_object_set()` to set the ::text_column property directly.

completion : A GtkEntryCompletion.
column : The column in the model of *completion* to get strings from.

Since 2.4

---

gtk_entry_completion_get_text_column ()

```
gint        gtk_entry_completion_get_text_column
                                            (GtkEntryCompletion *completion);
```

Returns the column in the model of *completion* to get strings from.

completion : a GtkEntryCompletion
Returns : the column containing the strings

Since 2.6

---

gtk_entry_completion_set_inline_completion ()

```
void        gtk_entry_completion_set_inline_completion
                                            (GtkEntryCompletion *completion,
                                             gboolean inline_completion);
```

Sets whether the common prefix of the possible completions should be automatically inserted in the entry.

completion : a GtkEntryCompletion
inline_completion : TRUE to do inline completion

Since 2.6

---

gtk_entry_completion_get_inline_completion ()

```
gboolean    gtk_entry_completion_get_inline_completion
                                            (GtkEntryCompletion *completion);
```

Returns whether the common prefix of the possible completions should be automatically inserted in the entry.

completion : a GtkEntryCompletion

Returns : TRUE if inline completion is turned on

Since 2.6

---

gtk_entry_completion_set_popup_completion ()

```
void        gtk_entry_completion_set_popup_completion
                                            (GtkEntryCompletion *completion,
                                             gboolean popup_completion);
```

Sets whether the completions should be presented in a popup window.

completion : a GtkEntryCompletion

popup_completion : TRUE to do popup completion

Since 2.6

---

gtk_entry_completion_get_popup_completion ()

```
gboolean    gtk_entry_completion_get_popup_completion
                                            (GtkEntryCompletion *completion);
```

Returns whether the completions should be presented in a popup window.

completion : a GtkEntryCompletion

Returns : TRUE if popup completion is turned on

Since 2.6

Properties

The "inline-completion" property

```
  "inline-completion"    gboolean              : Read / Write
```

Whether the common prefix should be inserted automatically.

Default value: FALSE

The "minimum-key-length" property

```
  "minimum-key-length"   gint                  : Read / Write
```

Minimum length of the search key in order to look up matches.

Allowed values: >= -1

Default value: 1

The "model" property

```
  "model"                GtkTreeModel          : Read / Write
```

The model to find matches in.

The "popup-completion" property

```
  "popup-completion"     gboolean              : Read / Write
```

Whether the completions should be shown in a popup window.

Default value: TRUE

The "text-column" property

```
  "text-column"          gint                  : Read / Write
```

The column of the model containing the strings.

Allowed values: >= -1

Default value: -1

Since 2.6

Signals

The "action-activated" signal

```
void        user_function                 (GtkEntryCompletion *widget,
                                           gint index,
                                           gpointer user_data);
```

The ::action-activated signal is emitted when an action is activated.

widget : the object which received the signal
index : the index of the activated action
user_data : user data set when the signal handler was connected.

Since 2.4

The "insert-prefix" signal

```
gboolean    user_function                 (GtkEntryCompletion *widget,
                                           gchar *prefix,
                                           gpointer user_data);
```

The ::insert-prefix signal is emitted when the inline autocompletion is triggered. The default behaviour is to make the entry display the whole prefix and select the newly inserted part.

Applications may connect to this signal in order to insert only a smaller part of the *prefix* into the entry - e.g. the entry used in the GtkFileChooser inserts only the part of the prefix up to the next '/'.

widget : the object which received the signal
prefix : the common prefix of all possible completions
user_data : user data set when the signal handler was connected.
Returns : TRUE if the signal has been handled

Since 2.6

The "match-selected" signal

```
gboolean    user_function                  (GtkEntryCompletion *widget,
                                            GtkTreeModel *model,
                                            GtkTreeIter *iter,
                                            gpointer user_data);
```

The ::match-selected signal is emitted when a match from the list is selected. The default behaviour is to replace the contents of the entry with the contents of the text column in the row pointed to by *iter*.

widget : the object which received the signal
model : the GtkTreeModel containing the matches
iter : a GtkTreeIter positioned at the selected match
user_data : user data set when the signal handler was connected.
Returns : TRUE if the signal has been handled

Since 2.4

<< GtkEntry **GtkHScale >>**

GtkHScale

GtkHScale — A horizontal slider widget for selecting a value from a range

Synopsis

```
#include <gtk/gtk.h>


            GtkHScale;
GtkWidget*  gtk_hscale_new                  (GtkAdjustment *adjustment);
GtkWidget*  gtk_hscale_new_with_range       (gdouble min,
                                             gdouble max,
                                             gdouble step);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkWidget
               +----GtkRange
                     +----GtkScale
                           +----GtkHScale
```

Implemented Interfaces

GtkHScale implements AtkImplementorIface.

Description

The GtkHScale widget is used to allow the user to select a value using a horizontal slider. To create one, use `gtk_hscale_new_with_range()`.

The position to show the current value, and the number of decimal places shown can be set using the parent GtkScale class's functions.

Details

GtkHScale

```
typedef struct _GtkHScale GtkHScale;
```

The GtkHScale-struct struct contains private data only, and should be accessed using the functions below.

---

gtk_hscale_new ()

```
GtkWidget*  gtk_hscale_new                  (GtkAdjustment *adjustment);
```

Creates a new GtkHScale.

adjustment : the GtkAdjustment which sets the range of the scale.

Returns : a new GtkHScale.

---

gtk_hscale_new_with_range ()

```
GtkWidget*  gtk_hscale_new_with_range       (gdouble min,
                                             gdouble max,
                                             gdouble step);
```

Creates a new horizontal scale widget that lets the user input a number between *min* and *max* (including *min* and *max*) with the increment *step*. *step* must be nonzero; it's the distance the slider moves when using the

arrow keys to adjust the scale value.

min :	minimum value
max :	maximum value
step :	step increment (tick size) used with keyboard shortcuts
Returns :	a new GtkHScale

<< GtkEntryCompletion **GtkVScale >>**

GtkVScale

GtkVScale — A vertical slider widget for selecting a value from a range

Synopsis

```
#include <gtk/gtk.h>


            GtkVScale;
GtkWidget*  gtk_vscale_new                  (GtkAdjustment *adjustment);
GtkWidget*  gtk_vscale_new_with_range       (gdouble min,
                                             gdouble max,
                                             gdouble step);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkWidget
               +----GtkRange
                     +----GtkScale
                           +----GtkVScale
```

Implemented Interfaces

GtkVScale implements AtkImplementorIface.

Description

The GtkVScale widget is used to allow the user to select a value using a vertical slider. To create one, use `gtk_hscale_new_with_range()`.

The position to show the current value, and the number of decimal places shown can be set using the parent GtkScale class's functions.

Details

GtkVScale

```
typedef struct _GtkVScale GtkVScale;
```

The GtkVScale-struct struct contains private data only, and should be accessed using the functions below.

---

gtk_vscale_new ()

```
GtkWidget*  gtk_vscale_new                  (GtkAdjustment *adjustment);
```

Creates a new GtkVScale.

adjustment : the GtkAdjustment which sets the range of the scale.

Returns : a new GtkVScale.

---

gtk_vscale_new_with_range ()

```
GtkWidget*  gtk_vscale_new_with_range       (gdouble min,
                                             gdouble max,
                                             gdouble step);
```

Creates a new vertical scale widget that lets the user input a number between *min* and *max* (including *min* and *max*) with the increment *step*. *step* must be nonzero; it's the distance the slider moves when using the

arrow keys to adjust the scale value.

min : minimum value
max : maximum value
step : step increment (tick size) used with keyboard shortcuts
Returns : a new GtkVScale

<< GtkHScale **GtkSpinButton >>**

GtkSpinButton

GtkSpinButton — Retrieve an integer or floating-point number from the user

Synopsis

```
#include <gtk/gtk.h>

            GtkSpinButton;
enum        GtkSpinButtonUpdatePolicy;
enum        GtkSpinType;
void        gtk_spin_button_configure       (GtkSpinButton *spin_button,
                                             GtkAdjustment *adjustment,
                                             gdouble climb_rate,
                                             guint digits);
GtkWidget*  gtk_spin_button_new             (GtkAdjustment *adjustment,
                                             gdouble climb_rate,
                                             guint digits);
GtkWidget*  gtk_spin_button_new_with_range  (gdouble min,
                                             gdouble max,
                                             gdouble step);
void        gtk_spin_button_set_adjustment  (GtkSpinButton *spin_button,
                                             GtkAdjustment *adjustment);
GtkAdjustment* gtk_spin_button_get_adjustment
                                            (GtkSpinButton *spin_button);
void        gtk_spin_button_set_digits      (GtkSpinButton *spin_button,
                                             guint digits);
void        gtk_spin_button_set_increments  (GtkSpinButton *spin_button,
                                             gdouble step,
                                             gdouble page);
void        gtk_spin_button_set_range       (GtkSpinButton *spin_button,
                                             gdouble min,
                                             gdouble max);
#define     gtk_spin_button_get_value_as_float
gint        gtk_spin_button_get_value_as_int
                                            (GtkSpinButton *spin_button);
void        gtk_spin_button_set_value       (GtkSpinButton *spin_button,
                                             gdouble value);
void        gtk_spin_button_set_update_policy
```

```
                                                (GtkSpinButton *spin_button,
                                                 GtkSpinButtonUpdatePolicy policy);
void        gtk_spin_button_set_numeric         (GtkSpinButton *spin_button,
                                                 gboolean numeric);
void        gtk_spin_button_spin                (GtkSpinButton *spin_button,
                                                 GtkSpinType direction,
                                                 gdouble increment);
void        gtk_spin_button_set_wrap            (GtkSpinButton *spin_button,
                                                 gboolean wrap);
void        gtk_spin_button_set_snap_to_ticks
                                                (GtkSpinButton *spin_button,
                                                 gboolean snap_to_ticks);
void        gtk_spin_button_update              (GtkSpinButton *spin_button);
guint       gtk_spin_button_get_digits          (GtkSpinButton *spin_button);
void        gtk_spin_button_get_increments      (GtkSpinButton *spin_button,
                                                 gdouble *step,
                                                 gdouble *page);
gboolean    gtk_spin_button_get_numeric         (GtkSpinButton *spin_button);
void        gtk_spin_button_get_range           (GtkSpinButton *spin_button,
                                                 gdouble *min,
                                                 gdouble *max);
gboolean    gtk_spin_button_get_snap_to_ticks
                                                (GtkSpinButton *spin_button);
GtkSpinButtonUpdatePolicy gtk_spin_button_get_update_policy
                                                (GtkSpinButton *spin_button);
gdouble     gtk_spin_button_get_value           (GtkSpinButton *spin_button);
gboolean    gtk_spin_button_get_wrap            (GtkSpinButton *spin_button);
#define     GTK_INPUT_ERROR
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkWidget
               +----GtkEntry
                     +----GtkSpinButton
```

Implemented Interfaces

GtkSpinButton implements AtkImplementorIface, GtkCellEditable and GtkEditable.

Properties

```
  "adjustment"           GtkAdjustment         : Read / Write
  "climb-rate"           gdouble               : Read / Write
  "digits"               guint                 : Read / Write
  "numeric"              gboolean              : Read / Write
  "snap-to-ticks"        gboolean              : Read / Write
  "update-policy"        GtkSpinButtonUpdatePolicy  : Read / Write
  "value"                gdouble               : Read / Write
  "wrap"                 gboolean              : Read / Write
```

Style Properties

```
  "shadow-type"          GtkShadowType         : Read
```

Signal Prototypes

```
"change-value"
            void        user_function      (GtkSpinButton *spinbutton,
                                            GtkScrollType arg1,
                                            gpointer user_data);
"input"     gint        user_function      (GtkSpinButton *spinbutton,
                                            gpointer arg1,
                                            gpointer user_data);
"output"    gboolean    user_function      (GtkSpinButton *spinbutton,
                                            gpointer user_data);
"value-changed"
            void        user_function      (GtkSpinButton *spinbutton,
                                            gpointer user_data);
```

Description

A GtkSpinButton is an ideal way to allow the user to set the value of some attribute. Rather than having to directly type a number into a GtkEntry, GtkSpinButton allows the user to click on one of two arrows to increment or decrement the displayed value. A value can still be typed in, with the bonus that it can be checked to ensure it is in a given range.

The main properties of a GtkSpinButton are through a GtkAdjustment. See the GtkAdjustment section for more details about an adjustment's properties.

Example 1. Using a GtkSpinButton to get an integer.

```
/* Provides a function to retrieve an integer value from a GtkSpinButton
 * and creates a spin button to model percentage values.
 */

gint grab_int_value (GtkSpinButton *a_spinner, gpointer user_data) {
   return gtk_spin_button_get_value_as_int (a_spinner);
}

void create_integer_spin_button (void) {

   GtkWidget *window, *spinner;
   GtkAdjustment *spinner_adj;

   spinner_adj = (GtkAdjustment *) gtk_adjustment_new (50.0, 0.0, 100.0, 1.0, 5.0,
5.0);

   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
   gtk_container_set_border_width (GTK_CONTAINER (window), 5);

   /* creates the spinner, with no decimal places */
   spinner = gtk_spin_button_new (spinner_adj, 1.0, 0);
   gtk_container_add (GTK_CONTAINER (window), spinner);

   gtk_widget_show_all (window);
   return;
}
```

Example 2. Using a GtkSpinButton to get a floating point value.

```
/* Provides a function to retrieve a floating point value from a
 * GtkSpinButton, and creates a high precision spin button.
 */

gfloat grab_int_value (GtkSpinButton *a_spinner, gpointer user_data) {
   return gtk_spin_button_get_value (a_spinner);
}

void create_floating_spin_button (void) {

   GtkWidget *window, *spinner;
   GtkAdjustment *spinner_adj;

   spinner_adj = (GtkAdjustment *) gtk_adjustment_new (2.500, 0.0, 5.0, 0.001, 0.1,
0.1);

   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
```

```
   gtk_container_set_border_width (GTK_CONTAINER (window), 5);

   /* creates the spinner, with three decimal places */
   spinner = gtk_spin_button_new (spinner_adj, 0.001, 3);
   gtk_container_add (GTK_CONTAINER (window), spinner);

   gtk_widget_show_all (window);
   return;
}
```

Details

GtkSpinButton

```
typedef struct _GtkSpinButton GtkSpinButton;
```

entry is the GtkEntry part of the GtkSpinButton widget, and can be used accordingly. All other fields contain private data and should only be modified using the functions below.

---

enum GtkSpinButtonUpdatePolicy

```
typedef enum
{
  GTK_UPDATE_ALWAYS,
  GTK_UPDATE_IF_VALID
} GtkSpinButtonUpdatePolicy;
```

GTK_UPDATE_ALWAYS	When refreshing your GtkSpinButton, the value is always displayed.
GTK_UPDATE_IF_VALID	When refreshing your GtkSpinButton, the value is only displayed if it is valid within the bounds of the spin button's GtkAdjustment.

---

enum GtkSpinType

```
typedef enum
{
  GTK_SPIN_STEP_FORWARD,
  GTK_SPIN_STEP_BACKWARD,
  GTK_SPIN_PAGE_FORWARD,
  GTK_SPIN_PAGE_BACKWARD,
```

```
  GTK_SPIN_HOME,
  GTK_SPIN_END,
  GTK_SPIN_USER_DEFINED
} GtkSpinType;
```

GTK_SPIN_STEP_FORWARD, GTK_SPIN_STEP_BACKWARD, GTK_SPIN_PAGE_FORWARD, GTK_SPIN_PAGE_BACKWARD	These values spin a GtkSpinButton by the relevant values of the spin button's GtkAdjustment.
GTK_SPIN_HOME, GTK_SPIN_END	These set the spin button's value to the minimum or maxmimum possible values, (set by its GtkAdjustment), respectively.
GTK_SPIN_USER_DEFINED	The programmer must specify the exact amount to spin the GtkSpinButton.

---

gtk_spin_button_configure ()

```
void        gtk_spin_button_configure       (GtkSpinButton *spin_button,
                                             GtkAdjustment *adjustment,
                                             gdouble climb_rate,
                                             guint digits);
```

Changes the properties of an existing spin button. The adjustment, climb rate, and number of decimal places are all changed accordingly, after this function call.

spin_button : a GtkSpinButton.
adjustment : a GtkAdjustment.
climb_rate : the new climb rate.
digits : the number of decimal places to display in the spin button.

---

gtk_spin_button_new ()

```
GtkWidget*  gtk_spin_button_new             (GtkAdjustment *adjustment,
                                             gdouble climb_rate,
                                             guint digits);
```

Creates a new GtkSpinButton.

adjustment : the GtkAdjustment object that this spin button should use.

climb_rate : specifies how much the spin button changes when an arrow is clicked on.
digits : the number of decimal places to display.
Returns : The new spin button as a GtkWidget.

---

gtk_spin_button_new_with_range ()

```
GtkWidget*  gtk_spin_button_new_with_range  (gdouble min,
                                             gdouble max,
                                             gdouble step);
```

This is a convenience constructor that allows creation of a numeric GtkSpinButton without manually creating an adjustment. The value is initially set to the minimum value and a page increment of 10 * *step* is the default. The precision of the spin button is equivalent to the precision of *step*.

min : Minimum allowable value
max : Maximum allowable value
step : Increment added or subtracted by spinning the widget
Returns : The new spin button as a GtkWidget.

---

gtk_spin_button_set_adjustment ()

```
void        gtk_spin_button_set_adjustment  (GtkSpinButton *spin_button,
                                             GtkAdjustment *adjustment);
```

Replaces the GtkAdjustment associated with *spin_button*.

spin_button : a GtkSpinButton
adjustment : a GtkAdjustment to replace the existing adjustment

---

gtk_spin_button_get_adjustment ()

```
GtkAdjustment* gtk_spin_button_get_adjustment
                                             (GtkSpinButton *spin_button);
```

Get the adjustment associated with a GtkSpinButton

spin_button :

Returns : the GtkAdjustment of *spin_button*

---

gtk_spin_button_set_digits ()

```
void        gtk_spin_button_set_digits      (GtkSpinButton *spin_button,
                                             guint digits);
```

Set the precision to be displayed by *spin_button*. Up to 20 digit precision is allowed.

spin_button : a GtkSpinButton

digits : the number of digits after the decimal point to be displayed for the spin button's value

---

gtk_spin_button_set_increments ()

```
void        gtk_spin_button_set_increments  (GtkSpinButton *spin_button,
                                             gdouble step,
                                             gdouble page);
```

Sets the step and page increments for spin_button. This affects how quickly the value changes when the spin button's arrows are activated.

spin_button : a GtkSpinButton

step : increment applied for a button 1 press.

page : increment applied for a button 2 press.

---

gtk_spin_button_set_range ()

```
void        gtk_spin_button_set_range       (GtkSpinButton *spin_button,
                                             gdouble min,
                                             gdouble max);
```

Sets the minimum and maximum allowable values for *spin_button*

spin_button : a GtkSpinButton

min : minimum allowable value

max : maximum allowable value

---

gtk_spin_button_get_value_as_float

```
#define gtk_spin_button_get_value_as_float gtk_spin_button_get_value
```

Warning

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

Gets the value in the *spin_button*. This function is deprecated, use `gtk_spin_button_get_value()` instead.

Returns : the value of *spin_button*.

---

gtk_spin_button_get_value_as_int ()

```
gint        gtk_spin_button_get_value_as_int
                                        (GtkSpinButton *spin_button);
```

Get the value *spin_button* represented as an integer.

spin_button : a GtkSpinButton
Returns : the value of *spin_button*

---

gtk_spin_button_set_value ()

```
void        gtk_spin_button_set_value       (GtkSpinButton *spin_button,
                                             gdouble value);
```

Set the value of *spin_button*.

spin_button : a GtkSpinButton
value : the new value

---

gtk_spin_button_set_update_policy ()

```
void        gtk_spin_button_set_update_policy
                                            (GtkSpinButton *spin_button,
                                             GtkSpinButtonUpdatePolicy policy);
```

Sets the update behavior of a spin button. This determines whether the spin button is always updated or only when a valid value is set.

spin_button : a GtkSpinButton
policy : a GtkSpinButtonUpdatePolicy value

---

gtk_spin_button_set_numeric ()

```
void        gtk_spin_button_set_numeric     (GtkSpinButton *spin_button,
                                             gboolean numeric);
```

Sets the flag that determines if non-numeric text can be typed into the spin button.

spin_button : a GtkSpinButton
numeric : flag indicating if only numeric entry is allowed.

---

gtk_spin_button_spin ()

```
void        gtk_spin_button_spin            (GtkSpinButton *spin_button,
                                             GtkSpinType direction,
                                             gdouble increment);
```

Increment or decrement a spin button's value in a specified direction by a specified amount.

spin_button : a GtkSpinButton
direction : a GtkSpinType indicating the direction to spin.
increment : step increment to apply in the specified direction.

---

gtk_spin_button_set_wrap ()

```
void        gtk_spin_button_set_wrap        (GtkSpinButton *spin_button,
                                             gboolean wrap);
```

Sets the flag that determines if a spin button value wraps around to the opposite limit when the upper or lower limit of the range is exceeded.

spin_button : a GtkSpinButton
wrap : a flag indicating if wrapping behavior is performed.

---

gtk_spin_button_set_snap_to_ticks ()

```
void        gtk_spin_button_set_snap_to_ticks
                                            (GtkSpinButton *spin_button,
                                             gboolean snap_to_ticks);
```

Sets the policy as to whether values are corrected to the nearest step increment when a spin button is activated after providing an invalid value.

spin_button : a GtkSpinButton
snap_to_ticks : a flag indicating if invalid values should be corrected.

---

gtk_spin_button_update ()

```
void        gtk_spin_button_update          (GtkSpinButton *spin_button);
```

Manually force an update of the spin button.

spin_button : a GtkSpinButton

---

gtk_spin_button_get_digits ()

```
guint       gtk_spin_button_get_digits      (GtkSpinButton *spin_button);
```

Fetches the precision of *spin_button*. See gtk_spin_button_set_digits().

spin_button : a GtkSpinButton
Returns : the current precision

---

gtk_spin_button_get_increments ()

```
void        gtk_spin_button_get_increments  (GtkSpinButton *spin_button,
                                             gdouble *step,
                                             gdouble *page);
```

Gets the current step and page the increments used by *spin_button*. See gtk_spin_button_set_increments().

spin_button : a GtkSpinButton
step : location to store step increment, or NULL
page : location to store page increment, or NULL

---

gtk_spin_button_get_numeric ()

```
gboolean    gtk_spin_button_get_numeric     (GtkSpinButton *spin_button);
```

Returns whether non-numeric text can be typed into the spin button. See gtk_spin_button_set_numeric().

spin_button : a GtkSpinButton
Returns : TRUE if only numeric text can be entered

---

gtk_spin_button_get_range ()

```
void        gtk_spin_button_get_range       (GtkSpinButton *spin_button,
                                             gdouble *min,
                                             gdouble *max);
```

Gets the range allowed for *spin_button*. See gtk_spin_button_set_range().

spin_button : a GtkSpinButton
min : location to store minimum allowed value, or NULL
max : location to store maximum allowed value, or NULL

---

gtk_spin_button_get_snap_to_ticks ()

```
gboolean    gtk_spin_button_get_snap_to_ticks
```

```
                                                (GtkSpinButton *spin_button);
```

Returns whether the values are corrected to the nearest step. See gtk_spin_button_set_snap_to_ticks().

spin_button : a GtkSpinButton
Returns : TRUE if values are snapped to the nearest step.

---

gtk_spin_button_get_update_policy ()

```
GtkSpinButtonUpdatePolicy gtk_spin_button_get_update_policy
                                                (GtkSpinButton *spin_button);
```

Gets the update behavior of a spin button. See gtk_spin_button_set_update_policy().

spin_button : a GtkSpinButton
Returns : the current update policy

---

gtk_spin_button_get_value ()

```
gdouble     gtk_spin_button_get_value       (GtkSpinButton *spin_button);
```

Get the value in the *spin_button*.

spin_button : a GtkSpinButton
Returns : the value of *spin_button*

---

gtk_spin_button_get_wrap ()

```
gboolean    gtk_spin_button_get_wrap        (GtkSpinButton *spin_button);
```

Returns whether the spin button's value wraps around to the opposite limit when the upper or lower limit of the range is exceeded. See gtk_spin_button_set_wrap().

spin_button : a GtkSpinButton
Returns : TRUE if the spin button wraps around

GTK_INPUT_ERROR

```
#define GTK_INPUT_ERROR -1
```

Properties

The "adjustment" property

```
  "adjustment"          GtkAdjustment         : Read / Write
```

The adjustment that holds the value of the spinbutton.

The "climb-rate" property

```
  "climb-rate"          gdouble               : Read / Write
```

The acceleration rate when you hold down a button.

Allowed values: >= 0

Default value: 0

The "digits" property

```
  "digits"              guint                 : Read / Write
```

The number of decimal places to display.

Allowed values: <= 20

Default value: 0

The "numeric" property

```
  "numeric"             gboolean              : Read / Write
```

Whether non-numeric characters should be ignored.

Default value: FALSE

The "snap-to-ticks" property

```
  "snap-to-ticks"       gboolean              : Read / Write
```

Whether erroneous values are automatically changed to a spin button's nearest step increment.

Default value: FALSE

The "update-policy" property

```
  "update-policy"       GtkSpinButtonUpdatePolicy  : Read / Write
```

Whether the spin button should update always, or only when the value is legal.

Default value: GTK_UPDATE_ALWAYS

The "value" property

```
  "value"               gdouble               : Read / Write
```

Reads the current value, or sets a new value.

Default value: 0

The "wrap" property

```
  "wrap"                gboolean              : Read / Write
```

Whether a spin button should wrap upon reaching its limits.

Default value: FALSE

Style Properties

The "shadow-type" style property

```
  "shadow-type"          GtkShadowType         : Read
```

the type of border that surrounds the arrows of a spin button.

Default value: GTK_SHADOW_IN

Signals

The "change-value" signal

```
void        user_function                  (GtkSpinButton *spinbutton,
                                            GtkScrollType arg1,
                                            gpointer user_data);
```

spinbutton : the object which received the signal.
arg1 :
user_data : user data set when the signal handler was connected.

The "input" signal

```
gint        user_function                  (GtkSpinButton *spinbutton,
                                            gpointer arg1,
                                            gpointer user_data);
```

spinbutton : the object which received the signal.
arg1 :
user_data : user data set when the signal handler was connected.
Returns :

The "output" signal

```
gboolean    user_function                   (GtkSpinButton *spinbutton,
                                             gpointer user_data);
```

spinbutton : the object which received the signal.

user_data : user data set when the signal handler was connected.

Returns :

The "value-changed" signal

```
void        user_function                   (GtkSpinButton *spinbutton,
                                             gpointer user_data);
```

spinbutton : the object which received the signal.

user_data : user data set when the signal handler was connected.

See Also

GtkEntry retrieve text rather than numbers.

<< GtkVScale **GtkEditable >>**

GtkEditable

GtkEditable — Interface for text-editing widgets

Synopsis

```
#include <gtk/gtk.h>

            GtkEditable;
void        gtk_editable_select_region      (GtkEditable *editable,
                                             gint start,
                                             gint end);
gboolean    gtk_editable_get_selection_bounds
                                            (GtkEditable *editable,
                                             gint *start,
                                             gint *end);
void        gtk_editable_insert_text        (GtkEditable *editable,
                                             const gchar *new_text,
                                             gint new_text_length,
                                             gint *position);
void        gtk_editable_delete_text        (GtkEditable *editable,
                                             gint start_pos,
                                             gint end_pos);
gchar*      gtk_editable_get_chars          (GtkEditable *editable,
                                             gint start_pos,
                                             gint end_pos);
void        gtk_editable_cut_clipboard      (GtkEditable *editable);
void        gtk_editable_copy_clipboard     (GtkEditable *editable);
void        gtk_editable_paste_clipboard    (GtkEditable *editable);
void        gtk_editable_delete_selection   (GtkEditable *editable);
void        gtk_editable_set_position       (GtkEditable *editable,
                                             gint position);
gint        gtk_editable_get_position       (GtkEditable *editable);
void        gtk_editable_set_editable       (GtkEditable *editable,
                                             gboolean is_editable);
```

```
gboolean    gtk_editable_get_editable       (GtkEditable *editable);
```

Object Hierarchy

```
  GInterface
   +----GtkEditable
```

Known Implementations

GtkEditable is implemented by GtkEntry, GtkOldEditable, GtkSpinButton and GtkText.

Signal Prototypes

```
"changed"   void        user_function      (GtkEditable *editable,
                                            gpointer user_data);
"delete-text"
            void        user_function      (GtkEditable *editable,
                                            gint start_pos,
                                            gint end_pos,
                                            gpointer user_data);
"insert-text"
            void        user_function      (GtkEditable *editable,
                                            gchar *new_text,
                                            gint new_text_length,
                                            gint *position,
                                            gpointer user_data);
```

Description

The GtkEditable interface is an interface which should be implemented by text editing widgets, such as GtkEntry and GtkText. It contains functions for generically manipulating an editable widget, a large number of action signals used for key bindings, and several signals that an application can connect to to modify the behavior of a widget.

As an example of the latter usage, by connecting the following handler to "insert_text", an application can convert all entry into a widget into uppercase.

Example 3. Forcing entry to uppercase.

```
include <ctype.h>

void
insert_text_handler (GtkEditable *editable,
                     const gchar *text,
                     gint         length,
                     gint        *position,
                     gpointer     data)
{
  int i;
  gchar *result = g_utf8_strup (text, length);

  g_signal_handlers_block_by_func (editable,
                                   (gpointer) insert_text_handler, data);
  gtk_editable_insert_text (editable, result, length, position);
  g_signal_handlers_unblock_by_func (editable,
                                     (gpointer) insert_text_handler, data);

  g_signal_stop_emission_by_name (editable, "insert_text");

  g_free (result);
}
```

Details

GtkEditable

```
typedef struct _GtkEditable GtkEditable;
```

The GtkEditable structure contains the following fields. (These fields should be considered read-only. They should never be set by an application.)

guint selection_start;	the starting position of the selected characters in the widget.
guint selection_end;	the end position of the selected characters in the widget.
guint editable;	a flag indicating whether or not the widget is editable by the user.

---

gtk_editable_select_region ()

```
void        gtk_editable_select_region      (GtkEditable *editable,
                                             gint start,
                                             gint end);
```

Selects a region of text. The characters that are selected are those characters at positions from *start_pos* up to, but not including *end_pos*. If *end_pos* is negative, then the the characters selected will be those characters from *start_pos* to the end of the text.

editable : a GtkEditable widget.
start : the starting position.
end : the end position.

---

gtk_editable_get_selection_bounds ()

```
gboolean    gtk_editable_get_selection_bounds
                                            (GtkEditable *editable,
                                             gint *start,
                                             gint *end);
```

Gets the current selection bounds, if there is a selection.

editable : a GtkEditable widget.
start : location to store the starting position, or NULL.
end : location to store the end position, or NULL.
Returns : TRUE if there is a selection.

---

gtk_editable_insert_text ()

```
void        gtk_editable_insert_text        (GtkEditable *editable,
                                             const gchar *new_text,
                                             gint new_text_length,
                                             gint *position);
```

Inserts text at a given position.

editable :	a GtkEditable widget.
new_text :	the text to insert.
new_text_length :	the length of the text to insert, in bytes
position :	an inout parameter. The caller initializes it to the position at which to insert the text. After the call it points at the position after the newly inserted text.

---

gtk_editable_delete_text ()

```
void        gtk_editable_delete_text        (GtkEditable *editable,
                                             gint start_pos,
                                             gint end_pos);
```

Deletes a sequence of characters. The characters that are deleted are those characters at positions from *start_pos* up to, but not including *end_pos*. If *end_pos* is negative, then the the characters deleted will be those characters from *start_pos* to the end of the text.

editable :	a GtkEditable widget.
start_pos :	the starting position.
end_pos :	the end position.

---

gtk_editable_get_chars ()

```
gchar*      gtk_editable_get_chars          (GtkEditable *editable,
                                             gint start_pos,
                                             gint end_pos);
```

Retrieves a sequence of characters. The characters that are retrieved are those characters at positions from *start_pos* up to, but not including *end_pos*. If *end_pos* is negative, then the the characters retrieved will be those characters from *start_pos* to the end of the text.

editable :	a GtkEditable widget.
start_pos :	the starting position.

end_pos : the end position.

Returns : the characters in the indicated region. The result must be freed with `g_free()` when the application is finished with it.

---

gtk_editable_cut_clipboard ()

```
void        gtk_editable_cut_clipboard      (GtkEditable *editable);
```

Causes the characters in the current selection to be copied to the clipboard and then deleted from the widget.

editable : a GtkEditable widget.

---

gtk_editable_copy_clipboard ()

```
void        gtk_editable_copy_clipboard     (GtkEditable *editable);
```

Causes the characters in the current selection to be copied to the clipboard.

editable : a GtkEditable widget.

---

gtk_editable_paste_clipboard ()

```
void        gtk_editable_paste_clipboard    (GtkEditable *editable);
```

Causes the contents of the clipboard to be pasted into the given widget at the current cursor position.

editable : a GtkEditable widget.

---

gtk_editable_delete_selection ()

```
void        gtk_editable_delete_selection   (GtkEditable *editable);
```

Deletes the current contents of the widgets selection and disclaims the selection.

editable : a GtkEditable widget.

---

gtk_editable_set_position ()

```
void        gtk_editable_set_position       (GtkEditable *editable,
                                             gint position);
```

Sets the cursor position.

editable : a GtkEditable widget.

position : the position of the cursor. The cursor is displayed before the character with the given (base 0) index in the widget. The value must be less than or equal to the number of characters in the widget. A value of -1 indicates that the position should be set after the last character in the entry. Note that this position is in characters, not in bytes.

---

gtk_editable_get_position ()

```
gint        gtk_editable_get_position       (GtkEditable *editable);
```

Retrieves the current cursor position.

editable : a GtkEditable widget.

Returns : the position of the cursor. The cursor is displayed before the character with the given (base 0) index in the widget. The value will be less than or equal to the number of characters in the widget. Note that this position is in characters, not in bytes.

---

gtk_editable_set_editable ()

```
void        gtk_editable_set_editable       (GtkEditable *editable,
                                             gboolean is_editable);
```

Determines if the user can edit the text in the editable widget or not.

editable : a GtkEditable widget.
is_editable : TRUE if the user is allowed to edit the text in the widget.

---

gtk_editable_get_editable ()

```
gboolean    gtk_editable_get_editable       (GtkEditable *editable);
```

Retrieves whether *editable* is editable. See gtk_editable_set_editable().

editable : a GtkEditable
Returns : TRUE if *editable* is editable.

Signals

The "changed" signal

```
void        user_function                   (GtkEditable *editable,
                                             gpointer user_data);
```

Indicates that the user has changed the contents of the widget.

editable : the object which received the signal.
user_data : user data set when the signal handler was connected.

The "delete-text" signal

```
void        user_function                   (GtkEditable *editable,
                                             gint start_pos,
                                             gint end_pos,
                                             gpointer user_data);
```

This signal is emitted when text is deleted from the widget by the user. The default handler for this signal will normally be responsible for inserting the text, so by connecting to this signal and then stopping the signal with `gtk_signal_emit_stop()`, it is possible to modify the inserted text, or prevent it from being inserted entirely. The *start_pos* and *end_pos* parameters are interpreted as for `gtk_editable_delete_text()`

editable : the object which received the signal.
start_pos : the starting position.
end_pos : the end position.
user_data : user data set when the signal handler was connected.

The "insert-text" signal

```
void        user_function                   (GtkEditable *editable,
                                             gchar *new_text,
                                             gint new_text_length,
                                             gint *position,
                                             gpointer user_data);
```

This signal is emitted when text is inserted into the widget by the user. The default handler for this signal will normally be responsible for inserting the text, so by connecting to this signal and then stopping the signal with `gtk_signal_emit_stop()`, it is possible to modify the inserted text, or prevent it from being inserted entirely.

editable : the object which received the signal.
new_text : the new text to insert.
new_text_length : the length of the new text.
position : the position at which to insert the new text. this is an in-out paramter. After the signal emission is finished, it should point after the newly inserted text.
user_data : user data set when the signal handler was connected.

<< GtkSpinButton **Multiline Text Editor >>**

Multiline Text Editor

Text Widget Overview - Overview of GtkTextBuffer, GtkTextView, and friends
GtkTextIter - Text buffer iterator
GtkTextMark - A position in the buffer preserved across buffer modifications
GtkTextBuffer - Stores attributed text for display in a GtkTextView
GtkTextTag - A tag that can be applied to text in a GtkTextBuffer
GtkTextTagTable - Collection of tags that can be used together
GtkTextView - Widget that displays a GtkTextBuffer

<< GtkEditable **Text Widget Overview >>**

Text Widget Overview

Text Widget Overview — Overview of GtkTextBuffer, GtkTextView, and friends

Conceptual Overview

GTK+ has an extremely powerful framework for multiline text editing. The primary objects involved in the process are GtkTextBuffer, which represents the text being edited, and GtkTextView, a widget which can display a GtkTextBuffer. Each buffer can be displayed by any number of views.

One of the important things to remember about text in GTK+ is that it's in the UTF-8 encoding. This means that one character can be encoded as multiple bytes. Character counts are usually referred to as *offsets*, while byte counts are called *indexes*. If you confuse these two, things will work fine with ASCII, but as soon as your buffer contains multibyte characters, bad things will happen.

Text in a buffer can be marked with *tags*. A tag is an attribute that can be applied to some range of text. For example, a tag might be called "bold" and make the text inside the tag bold. However, the tag concept is more general than that; tags don't have to affect appearance. They can instead affect the behavior of mouse and key presses, "lock" a range of text so the user can't edit it, or countless other things. A tag is represented by a GtkTextTag object. One GtkTextTag can be applied to any number of text ranges in any number of buffers.

Each tag is stored in a GtkTextTagTable. A tag table defines a set of tags that can be used together. Each buffer has one tag table associated with it; only tags from that tag table can be used with the buffer. A single tag table can be shared between multiple buffers, however.

Tags can have names, which is convenient sometimes (for example, you can name your tag that makes things bold "bold"), but they can also be anonymous (which is convenient if you're creating tags on-the-fly).

Most text manipulation is accomplished with *iterators*, represented by a GtkTextIter. An iterator represents a position between two characters in the text buffer. GtkTextIter is a struct designed to be allocated on the stack; it's guaranteed to be copiable by value and never contain any heap-allocated data. Iterators are not valid indefinitely; whenever the buffer is modified in a way that affects the number of characters in the buffer, all outstanding iterators become invalid. (Note that deleting 5 characters and

then reinserting 5 still invalidates iterators, though you end up with the same number of characters you pass through a state with a different number).

Because of this, iterators can't be used to preserve positions across buffer modifications. To preserve a position, the GtkTextMark object is ideal. You can think of a mark as an invisible cursor or insertion point; it floats in the buffer, saving a position. If the text surrounding the mark is deleted, the mark remains in the position the text once occupied; if text is inserted at the mark, the mark ends up either to the left or to the right of the new text, depending on its *gravity*. The standard text cursor in left-to-right languages is a mark with right gravity, because it stays to the right of inserted text.

Like tags, marks can be either named or anonymous. There are two marks built-in to GtkTextBuffer; these are named `"insert"` and `"selection_bound"` and refer to the insertion point and the boundary of the selection which is not the insertion point, respectively. If no text is selected, these two marks will be in the same position. You can manipulate what is selected and where the cursor appears by moving these marks around. [2]

Text buffers always contain at least one line, but may be empty (that is, buffers can contain zero characters). The last line in the text buffer never ends in a line separator (such as newline); the other lines in the buffer always end in a line separator. Line separators count as characters when computing character counts and character offsets. Note that some Unicode line separators are represented with multiple bytes in UTF-8, and the two-character sequence "\r\n" is also considered a line separator.

Simple Example

The simplest usage of GtkTextView might look like this:

```
GtkWidget *view;
GtkTextBuffer *buffer;

view = gtk_text_view_new ();

buffer = gtk_text_view_get_buffer (GTK_TEXT_VIEW (view));

gtk_text_buffer_set_text (buffer, "Hello, this is some text", -1);

/* Now you might put the view in a container and display it on the
 * screen; when the user edits the text, signals on the buffer
 * will be emitted, such as "changed", "insert_text", and so on.
 */
```

In many cases it's also convenient to first create the buffer with gtk_text_buffer_new(), then create a widget for that buffer with gtk_text_view_new_with_buffer(). Or you can change the buffer the widget displays after the widget is created with gtk_text_view_set_buffer().

Example of Changing Text Attributes

There are two ways to affect text attributes in GtkTextView. You can change the default attributes for a given GtkTextView, and you can apply tags that change the attributes for a region of text. For text features that come from the theme — such as font and foreground color — use standard GtkWidget functions such as gtk_widget_modify_font() or gtk_widget_modify_text(). For other attributes there are dedicated methods on GtkTextView such as gtk_text_view_set_tabs().

```
GtkWidget *view;
GtkTextBuffer *buffer;
GtkTextIter start, end;
PangoFontDescription *font_desc;
GdkColor color;
GtkTextTag *tag;

view = gtk_text_view_new ();

buffer = gtk_text_view_get_buffer (GTK_TEXT_VIEW (view));

gtk_text_buffer_set_text (buffer, "Hello, this is some text", -1);

/* Change default font throughout the widget */
font_desc = pango_font_description_from_string ("Serif 15");
gtk_widget_modify_font (view, font_desc);
pango_font_description_free (font_desc);

/* Change default color throughout the widget */
gdk_color_parse ("green", &color);
gtk_widget_modify_text (view, GTK_STATE_NORMAL, &color);

/* Change left margin throughout the widget */
gtk_text_view_set_left_margin (GTK_TEXT_VIEW (view), 30);

/* Use a tag to change the color for just one part of the widget */
tag = gtk_text_buffer_create_tag (buffer, "blue_foreground",
                                  "foreground", "blue", NULL);
gtk_text_buffer_get_iter_at_offset (buffer, &start, 7);
gtk_text_buffer_get_iter_at_offset (buffer, &end, 12);
```

```
gtk_text_buffer_apply_tag (buffer, tag, &start, &end);
```

The gtk-demo application that comes with GTK+ contains more example code for GtkTextView.

---

[2] If you want to place the cursor in response to a user action, be sure to use gtk_text_buffer_place_cursor(), which moves both at once without causing a temporary selection (moving one then the other temporarily selects the range in between the old and new positions).

<< Multiline Text Editor **GtkTextIter >>**

GtkTextIter

GtkTextIter — Text buffer iterator

Synopsis

```
#include <gtk/gtk.h>

            GtkTextIter;
GtkTextBuffer* gtk_text_iter_get_buffer     (const GtkTextIter *iter);
GtkTextIter* gtk_text_iter_copy             (const GtkTextIter *iter);
void        gtk_text_iter_free              (GtkTextIter *iter);
gint        gtk_text_iter_get_offset        (const GtkTextIter *iter);
gint        gtk_text_iter_get_line          (const GtkTextIter *iter);
gint        gtk_text_iter_get_line_offset   (const GtkTextIter *iter);
gint        gtk_text_iter_get_line_index    (const GtkTextIter *iter);
gint        gtk_text_iter_get_visible_line_index
                                            (const GtkTextIter *iter);
gint        gtk_text_iter_get_visible_line_offset
                                            (const GtkTextIter *iter);
gunichar    gtk_text_iter_get_char          (const GtkTextIter *iter);
gchar*      gtk_text_iter_get_slice         (const GtkTextIter *start,
                                             const GtkTextIter *end);
gchar*      gtk_text_iter_get_text          (const GtkTextIter *start,
                                             const GtkTextIter *end);
gchar*      gtk_text_iter_get_visible_slice (const GtkTextIter *start,
                                             const GtkTextIter *end);
gchar*      gtk_text_iter_get_visible_text  (const GtkTextIter *start,
                                             const GtkTextIter *end);
GdkPixbuf*  gtk_text_iter_get_pixbuf        (const GtkTextIter *iter);
GSList*     gtk_text_iter_get_marks         (const GtkTextIter *iter);
GSList*     gtk_text_iter_get_toggled_tags  (const GtkTextIter *iter,
                                             gboolean toggled_on);
GtkTextChildAnchor* gtk_text_iter_get_child_anchor
                                            (const GtkTextIter *iter);
```

```
gboolean    gtk_text_iter_begins_tag        (const GtkTextIter *iter,
                                             GtkTextTag *tag);
gboolean    gtk_text_iter_ends_tag          (const GtkTextIter *iter,
                                             GtkTextTag *tag);
gboolean    gtk_text_iter_toggles_tag       (const GtkTextIter *iter,
                                             GtkTextTag *tag);
gboolean    gtk_text_iter_has_tag           (const GtkTextIter *iter,
                                             GtkTextTag *tag);
GSList*     gtk_text_iter_get_tags          (const GtkTextIter *iter);
gboolean    gtk_text_iter_editable          (const GtkTextIter *iter,
                                             gboolean default_setting);
gboolean    gtk_text_iter_can_insert        (const GtkTextIter *iter,
                                             gboolean default_editability);
gboolean    gtk_text_iter_starts_word       (const GtkTextIter *iter);
gboolean    gtk_text_iter_ends_word         (const GtkTextIter *iter);
gboolean    gtk_text_iter_inside_word       (const GtkTextIter *iter);
gboolean    gtk_text_iter_starts_line       (const GtkTextIter *iter);
gboolean    gtk_text_iter_ends_line         (const GtkTextIter *iter);
gboolean    gtk_text_iter_starts_sentence   (const GtkTextIter *iter);
gboolean    gtk_text_iter_ends_sentence     (const GtkTextIter *iter);
gboolean    gtk_text_iter_inside_sentence   (const GtkTextIter *iter);
gboolean    gtk_text_iter_is_cursor_position
                                            (const GtkTextIter *iter);
gint        gtk_text_iter_get_chars_in_line (const GtkTextIter *iter);
gint        gtk_text_iter_get_bytes_in_line (const GtkTextIter *iter);
gboolean    gtk_text_iter_get_attributes    (const GtkTextIter *iter,
                                             GtkTextAttributes *values);
PangoLanguage* gtk_text_iter_get_language   (const GtkTextIter *iter);
gboolean    gtk_text_iter_is_end            (const GtkTextIter *iter);
gboolean    gtk_text_iter_is_start          (const GtkTextIter *iter);
gboolean    gtk_text_iter_forward_char      (GtkTextIter *iter);
gboolean    gtk_text_iter_backward_char     (GtkTextIter *iter);
gboolean    gtk_text_iter_forward_chars     (GtkTextIter *iter,
                                             gint count);
gboolean    gtk_text_iter_backward_chars    (GtkTextIter *iter,
                                             gint count);
gboolean    gtk_text_iter_forward_line      (GtkTextIter *iter);
gboolean    gtk_text_iter_backward_line     (GtkTextIter *iter);
gboolean    gtk_text_iter_forward_lines     (GtkTextIter *iter,
                                             gint count);
gboolean    gtk_text_iter_backward_lines    (GtkTextIter *iter,
                                             gint count);
```

```
gboolean    gtk_text_iter_forward_word_ends (GtkTextIter *iter,
                                             gint count);
gboolean    gtk_text_iter_backward_word_starts
                                            (GtkTextIter *iter,
                                             gint count);
gboolean    gtk_text_iter_forward_word_end  (GtkTextIter *iter);
gboolean    gtk_text_iter_backward_word_start
                                            (GtkTextIter *iter);
gboolean    gtk_text_iter_forward_cursor_position
                                            (GtkTextIter *iter);
gboolean    gtk_text_iter_backward_cursor_position
                                            (GtkTextIter *iter);
gboolean    gtk_text_iter_forward_cursor_positions
                                            (GtkTextIter *iter,
                                             gint count);
gboolean    gtk_text_iter_backward_cursor_positions
                                            (GtkTextIter *iter,
                                             gint count);
gboolean    gtk_text_iter_backward_sentence_start
                                            (GtkTextIter *iter);
gboolean    gtk_text_iter_backward_sentence_starts
                                            (GtkTextIter *iter,
                                             gint count);
gboolean    gtk_text_iter_forward_sentence_end
                                            (GtkTextIter *iter);
gboolean    gtk_text_iter_forward_sentence_ends
                                            (GtkTextIter *iter,
                                             gint count);
gboolean    gtk_text_iter_forward_visible_word_ends
                                            (GtkTextIter *iter,
                                             gint count);
gboolean    gtk_text_iter_backward_visible_word_starts
                                            (GtkTextIter *iter,
                                             gint count);
gboolean    gtk_text_iter_forward_visible_word_end
                                            (GtkTextIter *iter);
gboolean    gtk_text_iter_backward_visible_word_start
                                            (GtkTextIter *iter);
gboolean    gtk_text_iter_forward_visible_cursor_position
                                            (GtkTextIter *iter);
gboolean    gtk_text_iter_backward_visible_cursor_position
                                            (GtkTextIter *iter);
```

```
gboolean    gtk_text_iter_forward_visible_cursor_positions
                                            (GtkTextIter *iter,
                                             gint count);
gboolean    gtk_text_iter_backward_visible_cursor_positions
                                            (GtkTextIter *iter,
                                             gint count);
void        gtk_text_iter_set_offset        (GtkTextIter *iter,
                                             gint char_offset);
void        gtk_text_iter_set_line          (GtkTextIter *iter,
                                             gint line_number);
void        gtk_text_iter_set_line_offset   (GtkTextIter *iter,
                                             gint char_on_line);
void        gtk_text_iter_set_line_index    (GtkTextIter *iter,
                                             gint byte_on_line);
void        gtk_text_iter_set_visible_line_index
                                            (GtkTextIter *iter,
                                             gint byte_on_line);
void        gtk_text_iter_set_visible_line_offset
                                            (GtkTextIter *iter,
                                             gint char_on_line);
void        gtk_text_iter_forward_to_end    (GtkTextIter *iter);
gboolean    gtk_text_iter_forward_to_line_end
                                            (GtkTextIter *iter);
gboolean    gtk_text_iter_forward_to_tag_toggle
                                            (GtkTextIter *iter,
                                             GtkTextTag *tag);
gboolean    gtk_text_iter_backward_to_tag_toggle
                                            (GtkTextIter *iter,
                                             GtkTextTag *tag);
gboolean    (*GtkTextCharPredicate)         (gunichar ch,
                                             gpointer user_data);
gboolean    gtk_text_iter_forward_find_char (GtkTextIter *iter,
                                             GtkTextCharPredicate pred,
                                             gpointer user_data,
                                             const GtkTextIter *limit);
gboolean    gtk_text_iter_backward_find_char
                                            (GtkTextIter *iter,
                                             GtkTextCharPredicate pred,
                                             gpointer user_data,
                                             const GtkTextIter *limit);
enum        GtkTextSearchFlags;
gboolean    gtk_text_iter_forward_search    (const GtkTextIter *iter,
```

```
                                                const gchar *str,
                                                GtkTextSearchFlags flags,
                                                GtkTextIter *match_start,
                                                GtkTextIter *match_end,
                                                const GtkTextIter *limit);
gboolean    gtk_text_iter_backward_search   (const GtkTextIter *iter,
                                                const gchar *str,
                                                GtkTextSearchFlags flags,
                                                GtkTextIter *match_start,
                                                GtkTextIter *match_end,
                                                const GtkTextIter *limit);
gboolean    gtk_text_iter_equal             (const GtkTextIter *lhs,
                                                const GtkTextIter *rhs);
gint        gtk_text_iter_compare           (const GtkTextIter *lhs,
                                                const GtkTextIter *rhs);
gboolean    gtk_text_iter_in_range          (const GtkTextIter *iter,
                                                const GtkTextIter *start,
                                                const GtkTextIter *end);
void        gtk_text_iter_order             (GtkTextIter *first,
                                                GtkTextIter *second);
```

Description

You may wish to begin by reading the text widget conceptual overview which gives an overview of all the objects and data types related to the text widget and how they work together.

Details

GtkTextIter

```
typedef struct {
  /* GtkTextIter is an opaque datatype; ignore all these fields.
   * Initialize the iter with gtk_text_buffer_get_iter_*
   * functions
   */
} GtkTextIter;
```

---

gtk_text_iter_get_buffer ()

```
GtkTextBuffer* gtk_text_iter_get_buffer     (const GtkTextIter *iter);
```

Returns the GtkTextBuffer this iterator is associated with.

iter : an iterator
Returns : the buffer

---

gtk_text_iter_copy ()

```
GtkTextIter* gtk_text_iter_copy             (const GtkTextIter *iter);
```

Creates a dynamically-allocated copy of an iterator. This function is not useful in applications, because iterators can be copied with a simple assignment (`GtkTextIter i = j;`). The function is used by language bindings.

iter : an iterator
Returns : a copy of the *iter*, free with `gtk_text_iter_free()`

---

gtk_text_iter_free ()

```
void        gtk_text_iter_free              (GtkTextIter *iter);
```

Free an iterator allocated on the heap. This function is intended for use in language bindings, and is not especially useful for applications, because iterators can simply be allocated on the stack.

iter : a dynamically-allocated iterator

---

gtk_text_iter_get_offset ()

```
gint        gtk_text_iter_get_offset        (const GtkTextIter *iter);
```

Returns the character offset of an iterator. Each character in a GtkTextBuffer has an offset, starting with 0 for the first character in the buffer. Use `gtk_text_buffer_get_iter_at_offset()` to convert an offset back into an iterator.

iter : an iterator
Returns : a character offset

---

gtk_text_iter_get_line ()

```
gint        gtk_text_iter_get_line          (const GtkTextIter *iter);
```

Returns the line number containing the iterator. Lines in a GtkTextBuffer are numbered beginning with 0 for the first line in the buffer.

iter : an iterator
Returns : a line number

---

gtk_text_iter_get_line_offset ()

```
gint        gtk_text_iter_get_line_offset   (const GtkTextIter *iter);
```

Returns the character offset of the iterator, counting from the start of a newline-terminated line. The first character on the line has offset 0.

iter : an iterator
Returns : offset from start of line

---

gtk_text_iter_get_line_index ()

```
gint        gtk_text_iter_get_line_index    (const GtkTextIter *iter);
```

Returns the byte index of the iterator, counting from the start of a newline-terminated line. Remember that GtkTextBuffer encodes text in UTF-8, and that characters can require a variable number of bytes to represent.

iter : an iterator

Returns : distance from start of line, in bytes

---

gtk_text_iter_get_visible_line_index ()

```
gint        gtk_text_iter_get_visible_line_index
                                            (const GtkTextIter *iter);
```

Returns the number of bytes from the start of the line to the given *iter*, not counting bytes that are invisible due to tags with the "invisible" flag toggled on.

iter : a GtkTextIter

Returns : byte index of *iter* with respect to the start of the line

---

gtk_text_iter_get_visible_line_offset ()

```
gint        gtk_text_iter_get_visible_line_offset
                                            (const GtkTextIter *iter);
```

Returns the offset in characters from the start of the line to the given *iter*, not counting characters that are invisible due to tags with the "invisible" flag toggled on.

iter : a GtkTextIter

Returns : offset in visible characters from the start of the line

---

gtk_text_iter_get_char ()

```
gunichar    gtk_text_iter_get_char          (const GtkTextIter *iter);
```

Returns the Unicode character at this iterator. (Equivalent to operator* on a C++ iterator.) If the element at this iterator is a non-character element, such as an image embedded in the buffer, the Unicode "unknown" character 0xFFFC is returned. If invoked on the end iterator, zero is returned; zero is not a valid Unicode character. So you

can write a loop which ends when `gtk_text_iter_get_char()` returns 0.

iter : an iterator

Returns : a Unicode character, or 0 if *iter* is not dereferenceable

---

gtk_text_iter_get_slice ()

```
gchar*      gtk_text_iter_get_slice         (const GtkTextIter *start,
                                             const GtkTextIter *end);
```

Returns the text in the given range. A "slice" is an array of characters encoded in UTF-8 format, including the Unicode "unknown" character 0xFFFC for iterable non-character elements in the buffer, such as images. Because images are encoded in the slice, byte and character offsets in the returned array will correspond to byte offsets in the text buffer. Note that 0xFFFC can occur in normal text as well, so it is not a reliable indicator that a pixbuf or widget is in the buffer.

start : iterator at start of a range

end : iterator at end of a range

Returns : slice of text from the buffer

---

gtk_text_iter_get_text ()

```
gchar*      gtk_text_iter_get_text          (const GtkTextIter *start,
                                             const GtkTextIter *end);
```

Returns *text* in the given range. If the range contains non-text elements such as images, the character and byte offsets in the returned string will not correspond to character and byte offsets in the buffer. If you want offsets to correspond, see `gtk_text_iter_get_slice()`.

start : iterator at start of a range

end : iterator at end of a range

Returns : array of characters from the buffer

---

gtk_text_iter_get_visible_slice ()

```
gchar*      gtk_text_iter_get_visible_slice (const GtkTextIter *start,
                                             const GtkTextIter *end);
```

Like `gtk_text_iter_get_slice()`, but invisible text is not included. Invisible text is usually invisible because a GtkTextTag with the "invisible" attribute turned on has been applied to it.

start : iterator at start of range
end : iterator at end of range
Returns : slice of text from the buffer

---

gtk_text_iter_get_visible_text ()

```
gchar*      gtk_text_iter_get_visible_text  (const GtkTextIter *start,
                                             const GtkTextIter *end);
```

Like `gtk_text_iter_get_text()`, but invisible text is not included. Invisible text is usually invisible because a GtkTextTag with the "invisible" attribute turned on has been applied to it.

start : iterator at start of range
end : iterator at end of range
Returns : string containing visible text in the range

---

gtk_text_iter_get_pixbuf ()

```
GdkPixbuf*  gtk_text_iter_get_pixbuf        (const GtkTextIter *iter);
```

If the element at *iter* is a pixbuf, the pixbuf is returned (with no new reference count added). Otherwise, NULL is returned.

iter : an iterator
Returns : the pixbuf at *iter*

---

gtk_text_iter_get_marks ()

```
GSList*     gtk_text_iter_get_marks         (const GtkTextIter *iter);
```

Returns a list of all GtkTextMark at this location. Because marks are not iterable (they don't take up any "space" in the buffer, they are just marks in between iterable locations), multiple marks can exist in the same place. The returned list is not in any meaningful order.

iter : an iterator
Returns : list of GtkTextMark

---

gtk_text_iter_get_toggled_tags ()

```
GSList*     gtk_text_iter_get_toggled_tags  (const GtkTextIter *iter,
                                             gboolean toggled_on);
```

Returns a list of GtkTextTag that are toggled on or off at this point. (If *toggled_on* is TRUE, the list contains tags that are toggled on.) If a tag is toggled on at *iter*, then some non-empty range of characters following *iter* has that tag applied to it. If a tag is toggled off, then some non-empty range following *iter* does *not* have the tag applied to it.

iter : an iterator
toggled_on : TRUE to get toggled-on tags
Returns : tags toggled at this point

---

gtk_text_iter_get_child_anchor ()

```
GtkTextChildAnchor* gtk_text_iter_get_child_anchor
                                            (const GtkTextIter *iter);
```

If the location at *iter* contains a child anchor, the anchor is returned (with no new reference count added). Otherwise, NULL is returned.

iter : an iterator
Returns : the anchor at *iter*

---

gtk_text_iter_begins_tag ()

```
gboolean    gtk_text_iter_begins_tag        (const GtkTextIter *iter,
                                             GtkTextTag *tag);
```

Returns TRUE if *tag* is toggled on at exactly this point. If *tag* is NULL, returns TRUE if any tag is toggled on at this point. Note that the gtk_text_iter_begins_tag() returns TRUE if *iter* is the *start* of the tagged range; gtk_text_iter_has_tag() tells you whether an iterator is *within* a tagged range.

iter : an iterator

tag : a GtkTextTag, or NULL

Returns : whether *iter* is the start of a range tagged with *tag*

---

gtk_text_iter_ends_tag ()

```
gboolean    gtk_text_iter_ends_tag          (const GtkTextIter *iter,
                                             GtkTextTag *tag);
```

Returns TRUE if *tag* is toggled off at exactly this point. If *tag* is NULL, returns TRUE if any tag is toggled off at this point. Note that the gtk_text_iter_ends_tag() returns TRUE if *iter* is the *end* of the tagged range; gtk_text_iter_has_tag() tells you whether an iterator is *within* a tagged range.

iter : an iterator

tag : a GtkTextTag, or NULL

Returns : whether *iter* is the end of a range tagged with *tag*

---

gtk_text_iter_toggles_tag ()

```
gboolean    gtk_text_iter_toggles_tag       (const GtkTextIter *iter,
                                             GtkTextTag *tag);
```

This is equivalent to (gtk_text_iter_begins_tag() || gtk_text_iter_ends_tag()), i.e. it tells you whether a range with *tag* applied to it begins *or* ends at *iter*.

iter : an iterator

tag : a GtkTextTag, or NULL

Returns : whether *tag* is toggled on or off at *iter*

---

gtk_text_iter_has_tag ()

```
gboolean    gtk_text_iter_has_tag           (const GtkTextIter *iter,
                                             GtkTextTag *tag);
```

Returns TRUE if *iter* is within a range tagged with *tag*.

iter : an iterator

tag : a GtkTextTag

Returns : whether *iter* is tagged with *tag*

---

gtk_text_iter_get_tags ()

```
GSList*     gtk_text_iter_get_tags          (const GtkTextIter *iter);
```

Returns a list of tags that apply to *iter*, in ascending order of priority (highest-priority tags are last). The GtkTextTag in the list don't have a reference added, but you have to free the list itself.

iter : a GtkTextIter

Returns : list of GtkTextTag

---

gtk_text_iter_editable ()

```
gboolean    gtk_text_iter_editable          (const GtkTextIter *iter,
                                             gboolean default_setting);
```

Returns whether the character at *iter* is within an editable region of text. Non-editable text is "locked" and can't

be changed by the user via GtkTextView. This function is simply a convenience wrapper around gtk_text_iter_get_attributes(). If no tags applied to this text affect editability, *default_setting* will be returned.

You don't want to use this function to decide whether text can be inserted at *iter*, because for insertion you don't want to know whether the char at *iter* is inside an editable range, you want to know whether a new character inserted at *iter* would be inside an editable range. Use gtk_text_iter_can_insert() to handle this case.

iter :	an iterator
default_setting :	TRUE if text is editable by default
Returns :	whether *iter* is inside an editable range

---

gtk_text_iter_can_insert ()

```
gboolean    gtk_text_iter_can_insert        (const GtkTextIter *iter,
                                             gboolean default_editability);
```

Considering the default editability of the buffer, and tags that affect editability, determines whether text inserted at *iter* would be editable. If text inserted at *iter* would be editable then the user should be allowed to insert text at *iter*. gtk_text_buffer_insert_interactive() uses this function to decide whether insertions are allowed at a given position.

iter :	an iterator
default_editability :	TRUE if text is editable by default
Returns :	whether text inserted at *iter* would be editable

---

gtk_text_iter_starts_word ()

```
gboolean    gtk_text_iter_starts_word       (const GtkTextIter *iter);
```

Determines whether *iter* begins a natural-language word. Word breaks are determined by Pango and should be correct for nearly any language (if not, the correct fix would be to the Pango word break algorithms).

iter :	a GtkTextIter
Returns :	TRUE if *iter* is at the start of a word

gtk_text_iter_ends_word ()

```
gboolean    gtk_text_iter_ends_word         (const GtkTextIter *iter);
```

Determines whether *iter* ends a natural-language word. Word breaks are determined by Pango and should be correct for nearly any language (if not, the correct fix would be to the Pango word break algorithms).

iter : a GtkTextIter

Returns : TRUE if *iter* is at the end of a word

gtk_text_iter_inside_word ()

```
gboolean    gtk_text_iter_inside_word       (const GtkTextIter *iter);
```

Determines whether *iter* is inside a natural-language word (as opposed to say inside some whitespace). Word breaks are determined by Pango and should be correct for nearly any language (if not, the correct fix would be to the Pango word break algorithms).

iter : a GtkTextIter

Returns : TRUE if *iter* is inside a word

gtk_text_iter_starts_line ()

```
gboolean    gtk_text_iter_starts_line       (const GtkTextIter *iter);
```

Returns TRUE if *iter* begins a paragraph, i.e. if gtk_text_iter_get_line_offset() would return 0. However this function is potentially more efficient than gtk_text_iter_get_line_offset() because it doesn't have to compute the offset, it just has to see whether it's 0.

iter : an iterator

Returns : whether *iter* begins a line

gtk_text_iter_ends_line ()

```
gboolean    gtk_text_iter_ends_line         (const GtkTextIter *iter);
```

Returns TRUE if *iter* points to the start of the paragraph delimiter characters for a line (delimiters will be either a newline, a carriage return, a carriage return followed by a newline, or a Unicode paragraph separator character). Note that an iterator pointing to the \n of a \r\n pair will not be counted as the end of a line, the line ends before the \r. The end iterator is considered to be at the end of a line, even though there are no paragraph delimiter chars there.

iter : an iterator

Returns : whether *iter* is at the end of a line

---

gtk_text_iter_starts_sentence ()

```
gboolean    gtk_text_iter_starts_sentence   (const GtkTextIter *iter);
```

Determines whether *iter* begins a sentence. Sentence boundaries are determined by Pango and should be correct for nearly any language (if not, the correct fix would be to the Pango text boundary algorithms).

iter : a GtkTextIter

Returns : TRUE if *iter* is at the start of a sentence.

---

gtk_text_iter_ends_sentence ()

```
gboolean    gtk_text_iter_ends_sentence     (const GtkTextIter *iter);
```

Determines whether *iter* ends a sentence. Sentence boundaries are determined by Pango and should be correct for nearly any language (if not, the correct fix would be to the Pango text boundary algorithms).

iter : a GtkTextIter

Returns : TRUE if *iter* is at the end of a sentence.

---

gtk_text_iter_inside_sentence ()

```
gboolean    gtk_text_iter_inside_sentence   (const GtkTextIter *iter);
```

Determines whether *iter* is inside a sentence (as opposed to in between two sentences, e.g. after a period and before the first letter of the next sentence). Sentence boundaries are determined by Pango and should be correct for nearly any language (if not, the correct fix would be to the Pango text boundary algorithms).

iter : a GtkTextIter

Returns : TRUE if *iter* is inside a sentence.

---

gtk_text_iter_is_cursor_position ()

```
gboolean    gtk_text_iter_is_cursor_position
                                            (const GtkTextIter *iter);
```

See `gtk_text_iter_forward_cursor_position()` or PangoLogAttr or `pango_break()` for details on what a cursor position is.

iter : a GtkTextIter

Returns : TRUE if the cursor can be placed at *iter*

---

gtk_text_iter_get_chars_in_line ()

```
gint        gtk_text_iter_get_chars_in_line (const GtkTextIter *iter);
```

Returns the number of characters in the line containing *iter*, including the paragraph delimiters.

iter : an iterator

Returns : number of characters in the line

---

gtk_text_iter_get_bytes_in_line ()

```
gint        gtk_text_iter_get_bytes_in_line (const GtkTextIter *iter);
```

Returns the number of bytes in the line containing *iter*, including the paragraph delimiters.

iter : an iterator
Returns : number of bytes in the line

---

gtk_text_iter_get_attributes ()

```
gboolean    gtk_text_iter_get_attributes    (const GtkTextIter *iter,
                                             GtkTextAttributes *values);
```

Computes the effect of any tags applied to this spot in the text. The *values* parameter should be initialized to the default settings you wish to use if no tags are in effect. You'd typically obtain the defaults from gtk_text_view_get_default_attributes().

gtk_text_iter_get_attributes() will modify *values*, applying the effects of any tags present at *iter*. If any tags affected *values*, the function returns TRUE.

iter : an iterator
values : a GtkTextAttributes to be filled in
Returns : TRUE if *values* was modified

---

gtk_text_iter_get_language ()

```
PangoLanguage* gtk_text_iter_get_language   (const GtkTextIter *iter);
```

A convenience wrapper around gtk_text_iter_get_attributes(), which returns the language in effect at *iter*. If no tags affecting language apply to *iter*, the return value is identical to that of gtk_get_default_language().

iter : an iterator
Returns : language in effect at *iter*

---

gtk_text_iter_is_end ()

```
gboolean    gtk_text_iter_is_end            (const GtkTextIter *iter);
```

Returns TRUE if *iter* is the end iterator, i.e. one past the last dereferenceable iterator in the buffer. gtk_text_iter_is_end() is the most efficient way to check whether an iterator is the end iterator.

iter : an iterator

Returns : whether *iter* is the end iterator

---

gtk_text_iter_is_start ()

```
gboolean    gtk_text_iter_is_start          (const GtkTextIter *iter);
```

Returns TRUE if *iter* is the first iterator in the buffer, that is if *iter* has a character offset of 0.

iter : an iterator

Returns : whether *iter* is the first in the buffer

---

gtk_text_iter_forward_char ()

```
gboolean    gtk_text_iter_forward_char      (GtkTextIter *iter);
```

Moves *iter* forward by one character offset. Note that images embedded in the buffer occupy 1 character slot, so gtk_text_iter_forward_char() may actually move onto an image instead of a character, if you have images in your buffer. If *iter* is the end iterator or one character before it, *iter* will now point at the end iterator, and gtk_text_iter_forward_char() returns FALSE for convenience when writing loops.

iter : an iterator

Returns : whether *iter* moved and is dereferenceable

---

gtk_text_iter_backward_char ()

```
gboolean    gtk_text_iter_backward_char     (GtkTextIter *iter);
```

Moves backward by one character offset. Returns TRUE if movement was possible; if *iter* was the first in the buffer (character offset 0), gtk_text_iter_backward_char() returns FALSE for convenience when writing loops.

iter : an iterator

Returns : whether movement was possible

---

gtk_text_iter_forward_chars ()

```
gboolean    gtk_text_iter_forward_chars     (GtkTextIter *iter,
                                             gint count);
```

Moves *count* characters if possible (if *count* would move past the start or end of the buffer, moves to the start or end of the buffer). The return value indicates whether the new position of *iter* is different from its original position, and dereferenceable (the last iterator in the buffer is not dereferenceable). If *count* is 0, the function does nothing and returns FALSE.

iter : an iterator

count : number of characters to move, may be negative

Returns : whether *iter* moved and is dereferenceable

---

gtk_text_iter_backward_chars ()

```
gboolean    gtk_text_iter_backward_chars    (GtkTextIter *iter,
                                             gint count);
```

Moves *count* characters backward, if possible (if *count* would move past the start or end of the buffer, moves to the start or end of the buffer). The return value indicates whether the iterator moved onto a dereferenceable position; if the iterator didn't move, or moved onto the end iterator, then FALSE is returned. If *count* is 0, the function does nothing and returns FALSE.

iter : an iterator

count : number of characters to move

Returns : whether *iter* moved and is dereferenceable

---

gtk_text_iter_forward_line ()

```
gboolean    gtk_text_iter_forward_line      (GtkTextIter *iter);
```

Moves *iter* to the start of the next line. Returns TRUE if there was a next line to move to, and FALSE if *iter* was simply moved to the end of the buffer and is now not dereferenceable, or if *iter* was already at the end of the buffer.

iter : an iterator

Returns : whether *iter* can be dereferenced

---

gtk_text_iter_backward_line ()

```
gboolean    gtk_text_iter_backward_line     (GtkTextIter *iter);
```

Moves *iter* to the start of the previous line. Returns TRUE if *iter* could be moved; i.e. if *iter* was at character offset 0, this function returns FALSE. Therefore if *iter* was already on line 0, but not at the start of the line, *iter* is snapped to the start of the line and the function returns TRUE. (Note that this implies that in a loop calling this function, the line number may not change on every iteration, if your first iteration is on line 0.)

iter : an iterator

Returns : whether *iter* moved

---

gtk_text_iter_forward_lines ()

```
gboolean    gtk_text_iter_forward_lines     (GtkTextIter *iter,
                                             gint count);
```

Moves *count* lines forward, if possible (if *count* would move past the start or end of the buffer, moves to the start or end of the buffer). The return value indicates whether the iterator moved onto a dereferenceable position; if the iterator didn't move, or moved onto the end iterator, then FALSE is returned. If *count* is 0, the function does nothing and returns FALSE. If *count* is negative, moves backward by 0 - *count* lines.

iter : a GtkTextIter

count : number of lines to move forward

Returns : whether *iter* moved and is dereferenceable

---

gtk_text_iter_backward_lines ()

```
gboolean    gtk_text_iter_backward_lines    (GtkTextIter *iter,
                                             gint count);
```

Moves *count* lines backward, if possible (if *count* would move past the start or end of the buffer, moves to the start or end of the buffer). The return value indicates whether the iterator moved onto a dereferenceable position; if the iterator didn't move, or moved onto the end iterator, then FALSE is returned. If *count* is 0, the function does nothing and returns FALSE. If *count* is negative, moves forward by 0 - *count* lines.

iter : a GtkTextIter

count : number of lines to move backward

Returns : whether *iter* moved and is dereferenceable

---

gtk_text_iter_forward_word_ends ()

```
gboolean    gtk_text_iter_forward_word_ends (GtkTextIter *iter,
                                             gint count);
```

Calls gtk_text_iter_forward_word_end() up to *count* times.

iter : a GtkTextIter

count : number of times to move

Returns : TRUE if *iter* moved and is not the end iterator

---

gtk_text_iter_backward_word_starts ()

```
gboolean    gtk_text_iter_backward_word_starts
                                            (GtkTextIter *iter,
```

```
                                                gint count);
```

Calls gtk_text_iter_backward_word_start() up to *count* times.

iter : a GtkTextIter

count : number of times to move

Returns : TRUE if *iter* moved and is not the end iterator

---

gtk_text_iter_forward_word_end ()

```
gboolean    gtk_text_iter_forward_word_end  (GtkTextIter *iter);
```

Moves forward to the next word end. (If *iter* is currently on a word end, moves forward to the next one after that.) Word breaks are determined by Pango and should be correct for nearly any language (if not, the correct fix would be to the Pango word break algorithms).

iter : a GtkTextIter

Returns : TRUE if *iter* moved and is not the end iterator

---

gtk_text_iter_backward_word_start ()

```
gboolean    gtk_text_iter_backward_word_start
                                            (GtkTextIter *iter);
```

Moves backward to the previous word start. (If *iter* is currently on a word start, moves backward to the next one after that.) Word breaks are determined by Pango and should be correct for nearly any language (if not, the correct fix would be to the Pango word break algorithms).

iter : a GtkTextIter

Returns : TRUE if *iter* moved and is not the end iterator

---

gtk_text_iter_forward_cursor_position ()

```

```

```
gboolean    gtk_text_iter_forward_cursor_position
                                            (GtkTextIter *iter);
```

Moves *iter* forward by a single cursor position. Cursor positions are (unsurprisingly) positions where the cursor can appear. Perhaps surprisingly, there may not be a cursor position between all characters. The most common example for European languages would be a carriage return/newline sequence. For some Unicode characters, the equivalent of say the letter "a" with an accent mark will be represented as two characters, first the letter then a "combining mark" that causes the accent to be rendered; so the cursor can't go between those two characters. See also the PangoLogAttr structure and `pango_break()` function.

iter : a GtkTextIter

Returns : TRUE if we moved and the new position is dereferenceable

---

gtk_text_iter_backward_cursor_position ()

```
gboolean    gtk_text_iter_backward_cursor_position
                                            (GtkTextIter *iter);
```

Like `gtk_text_iter_forward_cursor_position()`, but moves backward.

iter : a GtkTextIter

Returns : TRUE if we moved

---

gtk_text_iter_forward_cursor_positions ()

```
gboolean    gtk_text_iter_forward_cursor_positions
                                            (GtkTextIter *iter,
                                             gint count);
```

Moves up to *count* cursor positions. See `gtk_text_iter_forward_cursor_position()` for details.

iter : a GtkTextIter

count : number of positions to move

Returns : TRUE if we moved and the new position is dereferenceable

---

gtk_text_iter_backward_cursor_positions ()

```
gboolean    gtk_text_iter_backward_cursor_positions
                                            (GtkTextIter *iter,
                                             gint count);
```

Moves up to *count* cursor positions. See gtk_text_iter_forward_cursor_position() for details.

iter : a GtkTextIter

count : number of positions to move

Returns : TRUE if we moved and the new position is dereferenceable

---

gtk_text_iter_backward_sentence_start ()

```
gboolean    gtk_text_iter_backward_sentence_start
                                            (GtkTextIter *iter);
```

Moves backward to the previous sentence start; if *iter* is already at the start of a sentence, moves backward to the next one. Sentence boundaries are determined by Pango and should be correct for nearly any language (if not, the correct fix would be to the Pango text boundary algorithms).

iter : a GtkTextIter

Returns : TRUE if *iter* moved and is not the end iterator

---

gtk_text_iter_backward_sentence_starts ()

```
gboolean    gtk_text_iter_backward_sentence_starts
                                            (GtkTextIter *iter,
                                             gint count);
```

Calls gtk_text_iter_backward_sentence_start() up to *count* times, or until it returns FALSE. If *count* is negative, moves forward instead of backward.

iter : a GtkTextIter

count : number of sentences to move

Returns : TRUE if *iter* moved and is not the end iterator

---

gtk_text_iter_forward_sentence_end ()

```
gboolean    gtk_text_iter_forward_sentence_end
                                            (GtkTextIter *iter);
```

Moves forward to the next sentence end. (If *iter* is at the end of a sentence, moves to the next end of sentence.) Sentence boundaries are determined by Pango and should be correct for nearly any language (if not, the correct fix would be to the Pango text boundary algorithms).

iter : a GtkTextIter

Returns : TRUE if *iter* moved and is not the end iterator

---

gtk_text_iter_forward_sentence_ends ()

```
gboolean    gtk_text_iter_forward_sentence_ends
                                            (GtkTextIter *iter,
                                             gint count);
```

Calls gtk_text_iter_forward_sentence_end() *count* times (or until gtk_text_iter_forward_sentence_end() returns FALSE). If *count* is negative, moves backward instead of forward.

iter : a GtkTextIter

count : number of sentences to move

Returns : TRUE if *iter* moved and is not the end iterator

---

gtk_text_iter_forward_visible_word_ends ()

```
gboolean    gtk_text_iter_forward_visible_word_ends
```

```
                                                    (GtkTextIter *iter,
                                                     gint count);
```

Calls `gtk_text_iter_forward_visible_word_end()` up to *`count`* times.

`iter` : a GtkTextIter
`count` : number of times to move
Returns : TRUE if *`iter`* moved and is not the end iterator

Since 2.4

---

gtk_text_iter_backward_visible_word_starts ()

```
gboolean    gtk_text_iter_backward_visible_word_starts
                                                    (GtkTextIter *iter,
                                                     gint count);
```

Calls `gtk_text_iter_backward_visible_word_start()` up to *`count`* times.

`iter` : a GtkTextIter
`count` : number of times to move
Returns : TRUE if *`iter`* moved and is not the end iterator

Since 2.4

---

gtk_text_iter_forward_visible_word_end ()

```
gboolean    gtk_text_iter_forward_visible_word_end
                                                    (GtkTextIter *iter);
```

Moves forward to the next visible word end. (If *`iter`* is currently on a word end, moves forward to the next one after that.) Word breaks are determined by Pango and should be correct for nearly any language (if not, the correct fix would be to the Pango word break algorithms).

iter : a GtkTextIter

Returns : TRUE if *iter* moved and is not the end iterator

Since 2.4

---

gtk_text_iter_backward_visible_word_start ()

```
gboolean    gtk_text_iter_backward_visible_word_start
                                            (GtkTextIter *iter);
```

Moves backward to the previous visible word start. (If *iter* is currently on a word start, moves backward to the next one after that.) Word breaks are determined by Pango and should be correct for nearly any language (if not, the correct fix would be to the Pango word break algorithms).

iter : a GtkTextIter

Returns : TRUE if *iter* moved and is not the end iterator

Since 2.4

---

gtk_text_iter_forward_visible_cursor_position ()

```
gboolean    gtk_text_iter_forward_visible_cursor_position
                                            (GtkTextIter *iter);
```

Moves *iter* forward to the next visible cursor position. See gtk_text_iter_forward_cursor_position() for details.

iter : a GtkTextIter

Returns : TRUE if we moved and the new position is dereferenceable

Since 2.4

---

gtk_text_iter_backward_visible_cursor_position ()

```
gboolean    gtk_text_iter_backward_visible_cursor_position
                                            (GtkTextIter *iter);
```

Moves *iter* forward to the previous visible cursor position. See gtk_text_iter_backward_cursor_position() for details.

iter : a GtkTextIter

Returns : TRUE if we moved and the new position is dereferenceable

Since 2.4

---

gtk_text_iter_forward_visible_cursor_positions ()

```
gboolean    gtk_text_iter_forward_visible_cursor_positions
                                            (GtkTextIter *iter,
                                             gint count);
```

Moves up to *count* visible cursor positions. See gtk_text_iter_forward_cursor_position() for details.

iter : a GtkTextIter

count : number of positions to move

Returns : TRUE if we moved and the new position is dereferenceable

Since 2.4

---

gtk_text_iter_backward_visible_cursor_positions ()

```
gboolean    gtk_text_iter_backward_visible_cursor_positions
                                            (GtkTextIter *iter,
```

```
                                                gint count);
```

Moves up to *count* visible cursor positions. See gtk_text_iter_forward_cursor_position() for details.

iter : a GtkTextIter
count : number of positions to move
Returns : TRUE if we moved and the new position is dereferenceable

Since 2.4

---

gtk_text_iter_set_offset ()

```
void        gtk_text_iter_set_offset        (GtkTextIter *iter,
                                             gint char_offset);
```

Sets *iter* to point to *char_offset*. *char_offset* counts from the start of the entire text buffer, starting with 0.

iter : a GtkTextIter
char_offset : a character number

---

gtk_text_iter_set_line ()

```
void        gtk_text_iter_set_line          (GtkTextIter *iter,
                                             gint line_number);
```

Moves iterator *iter* to the start of the line *line_number*. If *line_number* is negative or larger than the number of lines in the buffer, moves *iter* to the start of the last line in the buffer.

iter : a GtkTextIter
line_number : line number (counted from 0)

---

gtk_text_iter_set_line_offset ()

```
void        gtk_text_iter_set_line_offset   (GtkTextIter *iter,
                                             gint char_on_line);
```

Moves *iter* within a line, to a new *character* (not byte) offset. The given character offset must be less than or equal to the number of characters in the line; if equal, *iter* moves to the start of the next line. See gtk_text_iter_set_line_index() if you have a byte index rather than a character offset.

iter : a GtkTextIter

char_on_line : a character offset relative to the start of *iter*'s current line

---

gtk_text_iter_set_line_index ()

```
void        gtk_text_iter_set_line_index    (GtkTextIter *iter,
                                             gint byte_on_line);
```

Same as gtk_text_iter_set_line_offset(), but works with a *byte* index. The given byte index must be at the start of a character, it can't be in the middle of a UTF-8 encoded character.

iter : a GtkTextIter

byte_on_line : a byte index relative to the start of *iter*'s current line

---

gtk_text_iter_set_visible_line_index ()

```
void        gtk_text_iter_set_visible_line_index
                                            (GtkTextIter *iter,
                                             gint byte_on_line);
```

Like gtk_text_iter_set_line_index(), but the index is in visible bytes, i.e. text with a tag making it invisible is not counted in the index.

iter : a GtkTextIter

byte_on_line : a byte index

gtk_text_iter_set_visible_line_offset ()

```
void        gtk_text_iter_set_visible_line_offset
                                            (GtkTextIter *iter,
                                             gint char_on_line);
```

Like `gtk_text_iter_set_line_offset()`, but the offset is in visible characters, i.e. text with a tag making it invisible is not counted in the offset.

iter : a GtkTextIter
char_on_line : a character offset

gtk_text_iter_forward_to_end ()

```
void        gtk_text_iter_forward_to_end    (GtkTextIter *iter);
```

Moves *iter* forward to the "end iterator," which points one past the last valid character in the buffer. `gtk_text_iter_get_char()` called on the end iterator returns 0, which is convenient for writing loops.

iter : a GtkTextIter

gtk_text_iter_forward_to_line_end ()

```
gboolean    gtk_text_iter_forward_to_line_end
                                            (GtkTextIter *iter);
```

Moves the iterator to point to the paragraph delimiter characters, which will be either a newline, a carriage return, a carriage return/newline in sequence, or the Unicode paragraph separator character. If the iterator is already at the paragraph delimiter characters, moves to the paragraph delimiter characters for the next line. If *iter* is on the last line in the buffer, which does not end in paragraph delimiters, moves to the end iterator (end of the last line), and returns FALSE.

iter : a GtkTextIter

Returns : TRUE if we moved and the new location is not the end iterator

---

gtk_text_iter_forward_to_tag_toggle ()

```
gboolean    gtk_text_iter_forward_to_tag_toggle
                                            (GtkTextIter *iter,
                                             GtkTextTag *tag);
```

Moves forward to the next toggle (on or off) of the GtkTextTag *tag*, or to the next toggle of any tag if *tag* is NULL. If no matching tag toggles are found, returns FALSE, otherwise TRUE. Does not return toggles located at *iter*, only toggles after *iter*. Sets *iter* to the location of the toggle, or to the end of the buffer if no toggle is found.

iter : a GtkTextIter

tag : a GtkTextTag, or NULL

Returns : whether we found a tag toggle after *iter*

---

gtk_text_iter_backward_to_tag_toggle ()

```
gboolean    gtk_text_iter_backward_to_tag_toggle
                                            (GtkTextIter *iter,
                                             GtkTextTag *tag);
```

Moves backward to the next toggle (on or off) of the GtkTextTag *tag*, or to the next toggle of any tag if *tag* is NULL. If no matching tag toggles are found, returns FALSE, otherwise TRUE. Does not return toggles located at *iter*, only toggles before *iter*. Sets *iter* to the location of the toggle, or the start of the buffer if no toggle is found.

iter : a GtkTextIter

tag : a GtkTextTag, or NULL

Returns : whether we found a tag toggle before *iter*

GtkTextCharPredicate ()

```
gboolean    (*GtkTextCharPredicate)         (gunichar ch,
                                             gpointer user_data);
```

ch :
user_data :
Returns :

---

gtk_text_iter_forward_find_char ()

```
gboolean    gtk_text_iter_forward_find_char (GtkTextIter *iter,
                                             GtkTextCharPredicate pred,
                                             gpointer user_data,
                                             const GtkTextIter *limit);
```

Advances *iter*, calling *pred* on each character. If *pred* returns TRUE, returns TRUE and stops scanning. If *pred* never returns TRUE, *iter* is set to *limit* if *limit* is non-NULL, otherwise to the end iterator.

iter : a GtkTextIter
pred : a function to be called on each character
user_data : user data for *pred*
limit : search limit, or NULL for none
Returns : whether a match was found

---

gtk_text_iter_backward_find_char ()

```
gboolean    gtk_text_iter_backward_find_char
                                            (GtkTextIter *iter,
                                             GtkTextCharPredicate pred,
                                             gpointer user_data,
                                             const GtkTextIter *limit);
```

Same as gtk_text_iter_forward_find_char(), but goes backward from *iter*.

iter : a GtkTextIter

pred : function to be called on each character
user_data : user data for *pred*
limit : search limit, or NULL for none
Returns : whether a match was found

---

enum GtkTextSearchFlags

```
typedef enum {
  GTK_TEXT_SEARCH_VISIBLE_ONLY = 1 << 0,
  GTK_TEXT_SEARCH_TEXT_ONLY    = 1 << 1
  /* Possible future plans: SEARCH_CASE_INSENSITIVE, SEARCH_REGEXP */
} GtkTextSearchFlags;
```

---

gtk_text_iter_forward_search ()

```
gboolean    gtk_text_iter_forward_search    (const GtkTextIter *iter,
                                             const gchar *str,
                                             GtkTextSearchFlags flags,
                                             GtkTextIter *match_start,
                                             GtkTextIter *match_end,
                                             const GtkTextIter *limit);
```

Searches forward for *str*. Any match is returned by setting *match_start* to the first character of the match and *match_end* to the first character after the match. The search will not continue past *limit*. Note that a search is a linear or O(n) operation, so you may wish to use *limit* to avoid locking up your UI on large buffers.

If the GTK_TEXT_SEARCH_VISIBLE_ONLY flag is present, the match may have invisible text interspersed in *str*. i.e. *str* will be a possibly-noncontiguous subsequence of the matched range. similarly, if you specify GTK_TEXT_SEARCH_TEXT_ONLY, the match may have pixbufs or child widgets mixed inside the matched range. If these flags are not given, the match must be exact; the special 0xFFFC character in *str* will match embedded pixbufs or child widgets.

iter : start of search
str : a search string
flags : flags affecting how the search is done
match_start : return location for start of match, or NULL

match_end : return location for end of match, or NULL
limit : bound for the search, or NULL for the end of the buffer
Returns : whether a match was found

---

gtk_text_iter_backward_search ()

```
gboolean    gtk_text_iter_backward_search   (const GtkTextIter *iter,
                                             const gchar *str,
                                             GtkTextSearchFlags flags,
                                             GtkTextIter *match_start,
                                             GtkTextIter *match_end,
                                             const GtkTextIter *limit);
```

Same as gtk_text_iter_forward_search(), but moves backward.

iter : a GtkTextIter where the search begins
str : search string
flags : bitmask of flags affecting the search
match_start : return location for start of match, or NULL
match_end : return location for end of match, or NULL
limit : location of last possible *match_start*, or NULL for start of buffer
Returns : whether a match was found

---

gtk_text_iter_equal ()

```
gboolean    gtk_text_iter_equal             (const GtkTextIter *lhs,
                                             const GtkTextIter *rhs);
```

Tests whether two iterators are equal, using the fastest possible mechanism. This function is very fast; you can expect it to perform better than e.g. getting the character offset for each iterator and comparing the offsets yourself. Also, it's a bit faster than gtk_text_iter_compare().

lhs : a GtkTextIter
rhs : another GtkTextIter

Returns : TRUE if the iterators point to the same place in the buffer

---

gtk_text_iter_compare ()

```
gint        gtk_text_iter_compare           (const GtkTextIter *lhs,
                                             const GtkTextIter *rhs);
```

A `qsort()`-style function that returns negative if *`lhs`* is less than *`rhs`*, positive if *`lhs`* is greater than *`rhs`*, and 0 if they're equal. Ordering is in character offset order, i.e. the first character in the buffer is less than the second character in the buffer.

`lhs` : a GtkTextIter

`rhs` : another GtkTextIter

Returns : -1 if *`lhs`* is less than *`rhs`*, 1 if *`lhs`* is greater, 0 if they are equal

---

gtk_text_iter_in_range ()

```
gboolean    gtk_text_iter_in_range          (const GtkTextIter *iter,
                                             const GtkTextIter *start,
                                             const GtkTextIter *end);
```

Checks whether *`iter`* falls in the range [*`start`*, *`end`*). *`start`* and *`end`* must be in ascending order.

`iter` : a GtkTextIter

`start` : start of range

`end` : end of range

Returns : TRUE if *`iter`* is in the range

---

gtk_text_iter_order ()

```
void        gtk_text_iter_order             (GtkTextIter *first,
                                             GtkTextIter *second);
```

Swaps the value of *first* and *second* if *second* comes before *first* in the buffer. That is, ensures that *first* and *second* are in sequence. Most text buffer functions that take a range call this automatically on your behalf, so there's no real reason to call it yourself in those cases. There are some exceptions, such as gtk_text_iter_in_range(), that expect a pre-sorted range.

first : a GtkTextIter

second : another GtkTextIter

<< Text Widget Overview **GtkTextMark >>**

GtkTextMark

GtkTextMark — A position in the buffer preserved across buffer modifications

Synopsis

```
#include <gtk/gtk.h>


            GtkTextMark;
void        gtk_text_mark_set_visible       (GtkTextMark *mark,
                                             gboolean setting);
gboolean    gtk_text_mark_get_visible       (GtkTextMark *mark);
gboolean    gtk_text_mark_get_deleted       (GtkTextMark *mark);
G_CONST_RETURN gchar* gtk_text_mark_get_name
                                            (GtkTextMark *mark);
GtkTextBuffer* gtk_text_mark_get_buffer     (GtkTextMark *mark);
gboolean    gtk_text_mark_get_left_gravity  (GtkTextMark *mark);
```

Object Hierarchy

```
  GObject
   +----GtkTextMark
```

Description

You may wish to begin by reading the text widget conceptual overview which gives an overview of all

the objects and data types related to the text widget and how they work together.

A GtkTextMark is like a bookmark in a text buffer; it preserves a position in the text. You can convert the mark to an iterator using `gtk_text_buffer_get_iter_at_mark()`. Unlike iterators, marks remain valid across buffer mutations, because their behavior is defined when text is inserted or deleted. When text containing a mark is deleted, the mark remains in the position originally occupied by the deleted text. When text is inserted at a mark, a mark with *left gravity* will be moved to the beginning of the newly-inserted text, and a mark with *right gravity* will be moved to the end. [3]

Marks are reference counted, but the reference count only controls the validity of the memory; marks can be deleted from the buffer at any time with `gtk_text_buffer_delete_mark()`. Once deleted from the buffer, a mark is essentially useless.

Marks optionally have names; these can be convenient to avoid passing the GtkTextMark object around.

Marks are typically created using the `gtk_text_buffer_create_mark()` function.

Details

GtkTextMark

```
typedef struct _GtkTextMark GtkTextMark;
```

---

gtk_text_mark_set_visible ()

```
void        gtk_text_mark_set_visible       (GtkTextMark *mark,
                                             gboolean setting);
```

Sets the visibility of *`mark`*; the insertion point is normally visible, i.e. you can see it as a vertical bar. Also, the text widget uses a visible mark to indicate where a drop will occur when dragging-and-dropping text. Most other marks are not visible. Marks are not visible by default.

`mark` : a GtkTextMark

`setting` : visibility of mark

gtk_text_mark_get_visible ()

```
gboolean    gtk_text_mark_get_visible       (GtkTextMark *mark);
```

Returns TRUE if the mark is visible (i.e. a cursor is displayed for it)

mark : a GtkTextMark

Returns : TRUE if visible

gtk_text_mark_get_deleted ()

```
gboolean    gtk_text_mark_get_deleted       (GtkTextMark *mark);
```

Returns TRUE if the mark has been removed from its buffer with `gtk_text_buffer_delete_mark()`. Marks can't be used once deleted.

mark : a GtkTextMark

Returns : whether the mark is deleted

gtk_text_mark_get_name ()

```
G_CONST_RETURN gchar* gtk_text_mark_get_name
                                            (GtkTextMark *mark);
```

Returns the mark name; returns NULL for anonymous marks.

mark : a GtkTextMark

Returns : mark name

gtk_text_mark_get_buffer ()

```
GtkTextBuffer* gtk_text_mark_get_buffer     (GtkTextMark *mark);
```

Gets the buffer this mark is located inside, or NULL if the mark is deleted.

mark : a GtkTextMark

Returns : the mark's GtkTextBuffer

---

gtk_text_mark_get_left_gravity ()

```
gboolean    gtk_text_mark_get_left_gravity  (GtkTextMark *mark);
```

Determines whether the mark has left gravity.

mark : a GtkTextMark

Returns : TRUE if the mark has left gravity, FALSE otherwise

---

[3] "left" and "right" here refer to logical direction (left is the toward the start of the buffer); in some languages such as Hebrew the logically-leftmost text is not actually on the left when displayed.

<< GtkTextIter **GtkTextBuffer >>**

GtkTextBuffer

GtkTextBuffer — Stores attributed text for display in a GtkTextView

Synopsis

```
#include <gtk/gtk.h>


            GtkTextBuffer;
GtkTextBuffer* gtk_text_buffer_new          (GtkTextTagTable *table);
gint        gtk_text_buffer_get_line_count  (GtkTextBuffer *buffer);
gint        gtk_text_buffer_get_char_count  (GtkTextBuffer *buffer);
GtkTextTagTable* gtk_text_buffer_get_tag_table
                                            (GtkTextBuffer *buffer);
void        gtk_text_buffer_insert          (GtkTextBuffer *buffer,
                                             GtkTextIter *iter,
                                             const gchar *text,
                                             gint len);
void        gtk_text_buffer_insert_at_cursor
                                            (GtkTextBuffer *buffer,
                                             const gchar *text,
                                             gint len);
gboolean    gtk_text_buffer_insert_interactive
                                            (GtkTextBuffer *buffer,
                                             GtkTextIter *iter,
                                             const gchar *text,
                                             gint len,
                                             gboolean default_editable);
gboolean    gtk_text_buffer_insert_interactive_at_cursor
                                            (GtkTextBuffer *buffer,
                                             const gchar *text,
                                             gint len,
                                             gboolean default_editable);
void        gtk_text_buffer_insert_range    (GtkTextBuffer *buffer,
                                             GtkTextIter *iter,
                                             const GtkTextIter *start,
```

```
                                                const GtkTextIter *end);
gboolean    gtk_text_buffer_insert_range_interactive
                                               (GtkTextBuffer *buffer,
                                                GtkTextIter *iter,
                                                const GtkTextIter *start,
                                                const GtkTextIter *end,
                                                gboolean default_editable);
void        gtk_text_buffer_insert_with_tags
                                               (GtkTextBuffer *buffer,
                                                GtkTextIter *iter,
                                                const gchar *text,
                                                gint len,
                                                GtkTextTag *first_tag,
                                                ...);
void        gtk_text_buffer_insert_with_tags_by_name
                                               (GtkTextBuffer *buffer,
                                                GtkTextIter *iter,
                                                const gchar *text,
                                                gint len,
                                                const gchar *first_tag_name,
                                                ...);
void        gtk_text_buffer_delete             (GtkTextBuffer *buffer,
                                                GtkTextIter *start,
                                                GtkTextIter *end);
gboolean    gtk_text_buffer_delete_interactive
                                               (GtkTextBuffer *buffer,
                                                GtkTextIter *start_iter,
                                                GtkTextIter *end_iter,
                                                gboolean default_editable);
gboolean    gtk_text_buffer_backspace          (GtkTextBuffer *buffer,
                                                GtkTextIter *iter,
                                                gboolean interactive,
                                                gboolean default_editable);
void        gtk_text_buffer_set_text           (GtkTextBuffer *buffer,
                                                const gchar *text,
                                                gint len);
gchar*      gtk_text_buffer_get_text           (GtkTextBuffer *buffer,
                                                const GtkTextIter *start,
                                                const GtkTextIter *end,
                                                gboolean include_hidden_chars);
gchar*      gtk_text_buffer_get_slice          (GtkTextBuffer *buffer,
                                                const GtkTextIter *start,
                                                const GtkTextIter *end,
                                                gboolean include_hidden_chars);
```

```
void        gtk_text_buffer_insert_pixbuf   (GtkTextBuffer *buffer,
                                             GtkTextIter *iter,
                                             GdkPixbuf *pixbuf);
void        gtk_text_buffer_insert_child_anchor
                                            (GtkTextBuffer *buffer,
                                             GtkTextIter *iter,
                                             GtkTextChildAnchor *anchor);
GtkTextChildAnchor* gtk_text_buffer_create_child_anchor
                                            (GtkTextBuffer *buffer,
                                             GtkTextIter *iter);
GtkTextMark* gtk_text_buffer_create_mark    (GtkTextBuffer *buffer,
                                             const gchar *mark_name,
                                             const GtkTextIter *where,
                                             gboolean left_gravity);
void        gtk_text_buffer_move_mark       (GtkTextBuffer *buffer,
                                             GtkTextMark *mark,
                                             const GtkTextIter *where);
void        gtk_text_buffer_move_mark_by_name
                                            (GtkTextBuffer *buffer,
                                             const gchar *name,
                                             const GtkTextIter *where);
void        gtk_text_buffer_delete_mark     (GtkTextBuffer *buffer,
                                             GtkTextMark *mark);
void        gtk_text_buffer_delete_mark_by_name
                                            (GtkTextBuffer *buffer,
                                             const gchar *name);
GtkTextMark* gtk_text_buffer_get_mark       (GtkTextBuffer *buffer,
                                             const gchar *name);
GtkTextMark* gtk_text_buffer_get_insert     (GtkTextBuffer *buffer);
GtkTextMark* gtk_text_buffer_get_selection_bound
                                            (GtkTextBuffer *buffer);
void        gtk_text_buffer_place_cursor    (GtkTextBuffer *buffer,
                                             const GtkTextIter *where);
void        gtk_text_buffer_select_range    (GtkTextBuffer *buffer,
                                             const GtkTextIter *ins,
                                             const GtkTextIter *bound);
void        gtk_text_buffer_apply_tag       (GtkTextBuffer *buffer,
                                             GtkTextTag *tag,
                                             const GtkTextIter *start,
                                             const GtkTextIter *end);
void        gtk_text_buffer_remove_tag      (GtkTextBuffer *buffer,
                                             GtkTextTag *tag,
                                             const GtkTextIter *start,
```

```
                                                const GtkTextIter *end);
void        gtk_text_buffer_apply_tag_by_name
                                               (GtkTextBuffer *buffer,
                                                const gchar *name,
                                                const GtkTextIter *start,
                                                const GtkTextIter *end);
void        gtk_text_buffer_remove_tag_by_name
                                               (GtkTextBuffer *buffer,
                                                const gchar *name,
                                                const GtkTextIter *start,
                                                const GtkTextIter *end);
void        gtk_text_buffer_remove_all_tags (GtkTextBuffer *buffer,
                                                const GtkTextIter *start,
                                                const GtkTextIter *end);
GtkTextTag* gtk_text_buffer_create_tag      (GtkTextBuffer *buffer,
                                                const gchar *tag_name,
                                                const gchar *first_property_name,
                                                ...);
void        gtk_text_buffer_get_iter_at_line_offset
                                               (GtkTextBuffer *buffer,
                                                GtkTextIter *iter,
                                                gint line_number,
                                                gint char_offset);
void        gtk_text_buffer_get_iter_at_offset
                                               (GtkTextBuffer *buffer,
                                                GtkTextIter *iter,
                                                gint char_offset);
void        gtk_text_buffer_get_iter_at_line
                                               (GtkTextBuffer *buffer,
                                                GtkTextIter *iter,
                                                gint line_number);
void        gtk_text_buffer_get_iter_at_line_index
                                               (GtkTextBuffer *buffer,
                                                GtkTextIter *iter,
                                                gint line_number,
                                                gint byte_index);
void        gtk_text_buffer_get_iter_at_mark
                                               (GtkTextBuffer *buffer,
                                                GtkTextIter *iter,
                                                GtkTextMark *mark);
void        gtk_text_buffer_get_iter_at_child_anchor
                                               (GtkTextBuffer *buffer,
                                                GtkTextIter *iter,
```

```
                                                GtkTextChildAnchor *anchor);
void        gtk_text_buffer_get_start_iter  (GtkTextBuffer *buffer,
                                             GtkTextIter *iter);
void        gtk_text_buffer_get_end_iter    (GtkTextBuffer *buffer,
                                             GtkTextIter *iter);
void        gtk_text_buffer_get_bounds      (GtkTextBuffer *buffer,
                                             GtkTextIter *start,
                                             GtkTextIter *end);
gboolean    gtk_text_buffer_get_modified    (GtkTextBuffer *buffer);
void        gtk_text_buffer_set_modified    (GtkTextBuffer *buffer,
                                             gboolean setting);
gboolean    gtk_text_buffer_delete_selection
                                            (GtkTextBuffer *buffer,
                                             gboolean interactive,
                                             gboolean default_editable);
void        gtk_text_buffer_paste_clipboard (GtkTextBuffer *buffer,
                                             GtkClipboard *clipboard,
                                             GtkTextIter *override_location,
                                             gboolean default_editable);
void        gtk_text_buffer_copy_clipboard  (GtkTextBuffer *buffer,
                                             GtkClipboard *clipboard);
void        gtk_text_buffer_cut_clipboard   (GtkTextBuffer *buffer,
                                             GtkClipboard *clipboard,
                                             gboolean default_editable);
gboolean    gtk_text_buffer_get_selection_bounds
                                            (GtkTextBuffer *buffer,
                                             GtkTextIter *start,
                                             GtkTextIter *end);
void        gtk_text_buffer_begin_user_action
                                            (GtkTextBuffer *buffer);
void        gtk_text_buffer_end_user_action (GtkTextBuffer *buffer);
void        gtk_text_buffer_add_selection_clipboard
                                            (GtkTextBuffer *buffer,
                                             GtkClipboard *clipboard);
void        gtk_text_buffer_remove_selection_clipboard
                                            (GtkTextBuffer *buffer,
                                             GtkClipboard *clipboard);
```

Object Hierarchy

```
  GObject
   +----GtkTextBuffer
```

Properties

```
  "tag-table"           GtkTextTagTable       : Read / Write / Construct Only
```

Signal Prototypes

```
"apply-tag" void        user_function      (GtkTextBuffer *textbuffer,
                                            GtkTextTag *arg1,
                                            GtkTextIter *arg2,
                                            GtkTextIter *arg3,
                                            gpointer user_data);
"begin-user-action"
            void        user_function      (GtkTextBuffer *textbuffer,
                                            gpointer user_data);
"changed"   void        user_function      (GtkTextBuffer *textbuffer,
                                            gpointer user_data);
"delete-range"
            void        user_function      (GtkTextBuffer *textbuffer,
                                            GtkTextIter *arg1,
                                            GtkTextIter *arg2,
                                            gpointer user_data);
"end-user-action"
            void        user_function      (GtkTextBuffer *textbuffer,
                                            gpointer user_data);
"insert-child-anchor"
            void        user_function      (GtkTextBuffer *textbuffer,
                                            GtkTextIter *arg1,
                                            GtkTextChildAnchor *arg2,
                                            gpointer user_data);
"insert-pixbuf"
            void        user_function      (GtkTextBuffer *textbuffer,
                                            GtkTextIter *arg1,
                                            GdkPixbuf *arg2,
                                            gpointer user_data);
"insert-text"
```

```
            void        user_function      (GtkTextBuffer *textbuffer,
                                            GtkTextIter *arg1,
                                            gchar *arg2,
                                            gint arg3,
                                            gpointer user_data);
"mark-deleted"
            void        user_function      (GtkTextBuffer *textbuffer,
                                            GtkTextMark *arg1,
                                            gpointer user_data);
"mark-set"  void        user_function      (GtkTextBuffer *textbuffer,
                                            GtkTextIter *arg1,
                                            GtkTextMark *arg2,
                                            gpointer user_data);
"modified-changed"
            void        user_function      (GtkTextBuffer *textbuffer,
                                            gpointer user_data);
"remove-tag"
            void        user_function      (GtkTextBuffer *textbuffer,
                                            GtkTextTag *arg1,
                                            GtkTextIter *arg2,
                                            GtkTextIter *arg3,
                                            gpointer user_data);
```

Description

You may wish to begin by reading the text widget conceptual overview which gives an overview of all the objects and data types related to the text widget and how they work together.

Details

GtkTextBuffer

```
typedef struct _GtkTextBuffer GtkTextBuffer;
```

---

gtk_text_buffer_new ()

```
GtkTextBuffer* gtk_text_buffer_new          (GtkTextTagTable *table);
```

Creates a new text buffer.

table : a tag table, or NULL to create a new one
Returns : a new text buffer

---

gtk_text_buffer_get_line_count ()

```
gint        gtk_text_buffer_get_line_count  (GtkTextBuffer *buffer);
```

Obtains the number of lines in the buffer. This value is cached, so the function is very fast.

buffer : a GtkTextBuffer
Returns : number of lines in the buffer

---

gtk_text_buffer_get_char_count ()

```
gint        gtk_text_buffer_get_char_count  (GtkTextBuffer *buffer);
```

Gets the number of characters in the buffer; note that characters and bytes are not the same, you can't e.g. expect the contents of the buffer in string form to be this many bytes long. The character count is cached, so this function is very fast.

buffer : a GtkTextBuffer
Returns : number of characters in the buffer

---

gtk_text_buffer_get_tag_table ()

```
GtkTextTagTable* gtk_text_buffer_get_tag_table
                                            (GtkTextBuffer *buffer);
```

Get the GtkTextTagTable associated with this buffer.

buffer : a GtkTextBuffer
Returns : the buffer's tag table

gtk_text_buffer_insert ()

```
void        gtk_text_buffer_insert          (GtkTextBuffer *buffer,
                                             GtkTextIter *iter,
                                             const gchar *text,
                                             gint len);
```

Inserts *len* bytes of *text* at position *iter*. If *len* is -1, *text* must be nul-terminated and will be inserted in its entirety. Emits the "insert_text" signal; insertion actually occurs in the default handler for the signal. *iter* is invalidated when insertion occurs (because the buffer contents change), but the default signal handler revalidates it to point to the end of the inserted text.

buffer : a GtkTextBuffer
iter : a position in the buffer
text : UTF-8 format text to insert
len : length of text in bytes, or -1

---

gtk_text_buffer_insert_at_cursor ()

```
void        gtk_text_buffer_insert_at_cursor
                                            (GtkTextBuffer *buffer,
                                             const gchar *text,
                                             gint len);
```

Simply calls gtk_text_buffer_insert(), using the current cursor position as the insertion point.

buffer : a GtkTextBuffer
text : some text in UTF-8 format
len : length of text, in bytes

---

gtk_text_buffer_insert_interactive ()

```
gboolean    gtk_text_buffer_insert_interactive
                                            (GtkTextBuffer *buffer,
```

```
                                                GtkTextIter *iter,
                                                const gchar *text,
                                                gint len,
                                                gboolean default_editable);
```

Like gtk_text_buffer_insert(), but the insertion will not occur if *iter* is at a non-editable location in the buffer. Usually you want to prevent insertions at ineditable locations if the insertion results from a user action (is interactive).

default_editable indicates the editability of text that doesn't have a tag affecting editability applied to it. Typically the result of gtk_text_view_get_editable() is appropriate here.

buffer :	a GtkTextBuffer
iter :	a position in *buffer*
text :	some UTF-8 text
len :	length of text in bytes, or -1
default_editable :	default editability of buffer
Returns :	whether text was actually inserted

---

gtk_text_buffer_insert_interactive_at_cursor ()

```
gboolean    gtk_text_buffer_insert_interactive_at_cursor
                                               (GtkTextBuffer *buffer,
                                                const gchar *text,
                                                gint len,
                                                gboolean default_editable);
```

Calls gtk_text_buffer_insert_interactive() at the cursor position.

default_editable indicates the editability of text that doesn't have a tag affecting editability applied to it. Typically the result of gtk_text_view_get_editable() is appropriate here.

buffer :	a GtkTextBuffer
text :	text in UTF-8 format
len :	length of text in bytes, or -1
default_editable :	default editability of buffer
Returns :	whether text was actually inserted

---

gtk_text_buffer_insert_range ()

```
void        gtk_text_buffer_insert_range    (GtkTextBuffer *buffer,
                                             GtkTextIter *iter,
                                             const GtkTextIter *start,
                                             const GtkTextIter *end);
```

Copies text, tags, and pixbufs between *start* and *end* (the order of *start* and *end* doesn't matter) and inserts the copy at *iter*. Used instead of simply getting/inserting text because it preserves images and tags. If *start* and *end* are in a different buffer from *buffer*, the two buffers must share the same tag table.

Implemented via emissions of the insert_text and apply_tag signals, so expect those.

buffer : a GtkTextBuffer
iter : a position in *buffer*
start : a position in a GtkTextBuffer
end : another position in the same buffer as *start*

---

gtk_text_buffer_insert_range_interactive ()

```
gboolean    gtk_text_buffer_insert_range_interactive
                                            (GtkTextBuffer *buffer,
                                             GtkTextIter *iter,
                                             const GtkTextIter *start,
                                             const GtkTextIter *end,
                                             gboolean default_editable);
```

Same as gtk_text_buffer_insert_range(), but does nothing if the insertion point isn't editable. The *default_editable* parameter indicates whether the text is editable at *iter* if no tags enclosing *iter* affect editability. Typically the result of gtk_text_view_get_editable() is appropriate here.

buffer : a GtkTextBuffer
iter : a position in *buffer*
start : a position in a GtkTextBuffer
end : another position in the same buffer as *start*
default_editable : default editability of the buffer
Returns : whether an insertion was possible at *iter*

---

gtk_text_buffer_insert_with_tags ()

```
void        gtk_text_buffer_insert_with_tags
                                            (GtkTextBuffer *buffer,
                                             GtkTextIter *iter,
                                             const gchar *text,
                                             gint len,
                                             GtkTextTag *first_tag,
                                             ...);
```

Inserts *text* into *buffer* at *iter*, applying the list of tags to the newly-inserted text. The last tag specified must be NULL to terminate the list. Equivalent to calling gtk_text_buffer_insert(), then gtk_text_buffer_apply_tag() on the inserted text; gtk_text_buffer_insert_with_tags() is just a convenience function.

buffer :	a GtkTextBuffer
iter :	an iterator in *buffer*
text :	UTF-8 text
len :	length of *text*, or -1
first_tag :	first tag to apply to *text*
... :	NULL-terminated list of tags to apply

---

gtk_text_buffer_insert_with_tags_by_name ()

```
void        gtk_text_buffer_insert_with_tags_by_name
                                            (GtkTextBuffer *buffer,
                                             GtkTextIter *iter,
                                             const gchar *text,
                                             gint len,
                                             const gchar *first_tag_name,
                                             ...);
```

Same as gtk_text_buffer_insert_with_tags(), but allows you to pass in tag names instead of tag objects.

buffer :	a GtkTextBuffer
iter :	position in *buffer*
text :	UTF-8 text

len : length of *text*, or -1
first_tag_name : name of a tag to apply to *text*
... : more tag names

---

gtk_text_buffer_delete ()

```
void        gtk_text_buffer_delete          (GtkTextBuffer *buffer,
                                             GtkTextIter *start,
                                             GtkTextIter *end);
```

Deletes text between *start* and *end*. The order of *start* and *end* is not actually relevant; gtk_text_buffer_delete() will reorder them. This function actually emits the "delete_range" signal, and the default handler of that signal deletes the text. Because the buffer is modified, all outstanding iterators become invalid after calling this function; however, the *start* and *end* will be re-initialized to point to the location where text was deleted.

buffer : a GtkTextBuffer
start : a position in *buffer*
end : another position in *buffer*

---

gtk_text_buffer_delete_interactive ()

```
gboolean    gtk_text_buffer_delete_interactive
                                            (GtkTextBuffer *buffer,
                                             GtkTextIter *start_iter,
                                             GtkTextIter *end_iter,
                                             gboolean default_editable);
```

Deletes all *editable* text in the given range. Calls gtk_text_buffer_delete() for each editable sub-range of [*start*,*end*). *start* and *end* are revalidated to point to the location of the last deleted range, or left untouched if no text was deleted.

buffer : a GtkTextBuffer
start_iter : start of range to delete
end_iter : end of range
default_editable : whether the buffer is editable by default
Returns : whether some text was actually deleted

gtk_text_buffer_backspace ()

```
gboolean    gtk_text_buffer_backspace       (GtkTextBuffer *buffer,
                                             GtkTextIter *iter,
                                             gboolean interactive,
                                             gboolean default_editable);
```

Performs the appropriate action as if the user hit the delete key with the cursor at the position specified by *iter*. In the normal case a single character will be deleted, but when combining accents are involved, more than one character can be deleted, and when precomposed character and accent combinations, less than one character will be deleted.

iter must be at a cursor position.

buffer :	a GtkTextBuffer
iter :	a position in *buffer*
interactive :	whether the deletion is caused by user interaction
default_editable :	whether the buffer is editable by default
Returns :	TRUE if tbe buffer was modified

Since 2.6

gtk_text_buffer_set_text ()

```
void        gtk_text_buffer_set_text        (GtkTextBuffer *buffer,
                                             const gchar *text,
                                             gint len);
```

Deletes current contents of *buffer*, and inserts *text* instead. If *len* is -1, *text* must be nul-terminated. *text* must be valid UTF-8.

buffer :	a GtkTextBuffer
text :	UTF-8 text to insert
len :	length of *text* in bytes

gtk_text_buffer_get_text ()

```
gchar*      gtk_text_buffer_get_text        (GtkTextBuffer *buffer,
                                             const GtkTextIter *start,
                                             const GtkTextIter *end,
                                             gboolean include_hidden_chars);
```

Returns the text in the range [*start*,*end*). Excludes undisplayed text (text marked with tags that set the invisibility attribute) if *include_hidden_chars* is FALSE. Does not include characters representing embedded images, so byte and character indexes into the returned string do *not* correspond to byte and character indexes into the buffer. Contrast with gtk_text_buffer_get_slice().

buffer :	a GtkTextBuffer
start :	start of a range
end :	end of a range
include_hidden_chars :	whether to include invisible text
Returns :	an allocated UTF-8 string

---

gtk_text_buffer_get_slice ()

```
gchar*      gtk_text_buffer_get_slice       (GtkTextBuffer *buffer,
                                             const GtkTextIter *start,
                                             const GtkTextIter *end,
                                             gboolean include_hidden_chars);
```

Returns the text in the range [*start*,*end*). Excludes undisplayed text (text marked with tags that set the invisibility attribute) if *include_hidden_chars* is FALSE. The returned string includes a 0xFFFC character whenever the buffer contains embedded images, so byte and character indexes into the returned string *do* correspond to byte and character indexes into the buffer. Contrast with gtk_text_buffer_get_text(). Note that 0xFFFC can occur in normal text as well, so it is not a reliable indicator that a pixbuf or widget is in the buffer.

buffer :	a GtkTextBuffer
start :	start of a range
end :	end of a range
include_hidden_chars :	whether to include invisible text
Returns :	an allocated UTF-8 string

---

gtk_text_buffer_insert_pixbuf ()

```
void        gtk_text_buffer_insert_pixbuf   (GtkTextBuffer *buffer,
                                             GtkTextIter *iter,
                                             GdkPixbuf *pixbuf);
```

Inserts an image into the text buffer at *iter*. The image will be counted as one character in character counts, and when obtaining the buffer contents as a string, will be represented by the Unicode "object replacement character" 0xFFFC. Note that the "slice" variants for obtaining portions of the buffer as a string include this character for pixbufs, but the "text" variants do not. e.g. see gtk_text_buffer_get_slice() and gtk_text_buffer_get_text().

buffer : a GtkTextBuffer
iter : location to insert the pixbuf
pixbuf : a GdkPixbuf

---

gtk_text_buffer_insert_child_anchor ()

```
void        gtk_text_buffer_insert_child_anchor
                                            (GtkTextBuffer *buffer,
                                             GtkTextIter *iter,
                                             GtkTextChildAnchor *anchor);
```

Inserts a child widget anchor into the text buffer at *iter*. The anchor will be counted as one character in character counts, and when obtaining the buffer contents as a string, will be represented by the Unicode "object replacement character" 0xFFFC. Note that the "slice" variants for obtaining portions of the buffer as a string include this character for child anchors, but the "text" variants do not. e.g. see gtk_text_buffer_get_slice() and gtk_text_buffer_get_text(). Consider gtk_text_buffer_create_child_anchor() as a more convenient alternative to this function. The buffer will add a reference to the anchor, so you can unref it after insertion.

buffer : a GtkTextBuffer
iter : location to insert the anchor
anchor : a GtkTextChildAnchor

---

gtk_text_buffer_create_child_anchor ()

```
GtkTextChildAnchor* gtk_text_buffer_create_child_anchor
```

```
                                                (GtkTextBuffer *buffer,
                                                 GtkTextIter *iter);
```

This is a convenience function which simply creates a child anchor with gtk_text_child_anchor_new() and inserts it into the buffer with gtk_text_buffer_insert_child_anchor(). The new anchor is owned by the buffer; no reference count is returned to the caller of gtk_text_buffer_create_child_anchor().

buffer : a GtkTextBuffer
iter : location in the buffer
Returns : the created child anchor

---

gtk_text_buffer_create_mark ()

```
GtkTextMark* gtk_text_buffer_create_mark    (GtkTextBuffer *buffer,
                                             const gchar *mark_name,
                                             const GtkTextIter *where,
                                             gboolean left_gravity);
```

Creates a mark at position *where*. If *mark_name* is NULL, the mark is anonymous; otherwise, the mark can be retrieved by name using gtk_text_buffer_get_mark(). If a mark has left gravity, and text is inserted at the mark's current location, the mark will be moved to the left of the newly-inserted text. If the mark has right gravity (*left_gravity* = FALSE), the mark will end up on the right of newly-inserted text. The standard left-to-right cursor is a mark with right gravity (when you type, the cursor stays on the right side of the text you're typing).

The caller of this function does *not* own a reference to the returned GtkTextMark, so you can ignore the return value if you like. Marks are owned by the buffer and go away when the buffer does.

Emits the "mark_set" signal as notification of the mark's initial placement.

buffer : a GtkTextBuffer
mark_name : name for mark, or NULL
where : location to place mark
left_gravity : whether the mark has left gravity
Returns : the new GtkTextMark object

---

gtk_text_buffer_move_mark ()

```

```

```
void        gtk_text_buffer_move_mark       (GtkTextBuffer *buffer,
                                             GtkTextMark *mark,
                                             const GtkTextIter *where);
```

Moves *mark* to the new location *where*. Emits the "mark_set" signal as notification of the move.

buffer : a GtkTextBuffer
mark : a GtkTextMark
where : new location for *mark* in *buffer*

---

gtk_text_buffer_move_mark_by_name ()

```
void        gtk_text_buffer_move_mark_by_name
                                            (GtkTextBuffer *buffer,
                                             const gchar *name,
                                             const GtkTextIter *where);
```

Moves the mark named *name* (which must exist) to location *where*. See gtk_text_buffer_move_mark() for details.

buffer : a GtkTextBuffer
name : name of a mark
where : new location for mark

---

gtk_text_buffer_delete_mark ()

```
void        gtk_text_buffer_delete_mark     (GtkTextBuffer *buffer,
                                             GtkTextMark *mark);
```

Deletes *mark*, so that it's no longer located anywhere in the buffer. Removes the reference the buffer holds to the mark, so if you haven't called g_object_ref() on the mark, it will be freed. Even if the mark isn't freed, most operations on *mark* become invalid. There is no way to undelete a mark. gtk_text_mark_get_deleted() will return TRUE after this function has been called on a mark; gtk_text_mark_get_deleted() indicates that a mark no longer belongs to a buffer. The "mark_deleted" signal will be emitted as notification after the mark is deleted.

buffer : a GtkTextBuffer

mark : a GtkTextMark in *buffer*

---

gtk_text_buffer_delete_mark_by_name ()

```
void        gtk_text_buffer_delete_mark_by_name
                                            (GtkTextBuffer *buffer,
                                             const gchar *name);
```

Deletes the mark named *name*; the mark must exist. See `gtk_text_buffer_delete_mark()` for details.

buffer : a GtkTextBuffer
name : name of a mark in *buffer*

---

gtk_text_buffer_get_mark ()

```
GtkTextMark* gtk_text_buffer_get_mark       (GtkTextBuffer *buffer,
                                             const gchar *name);
```

Returns the mark named *name* in buffer *buffer*, or NULL if no such mark exists in the buffer.

buffer : a GtkTextBuffer
name : a mark name
Returns : a GtkTextMark, or NULL

---

gtk_text_buffer_get_insert ()

```
GtkTextMark* gtk_text_buffer_get_insert     (GtkTextBuffer *buffer);
```

Returns the mark that represents the cursor (insertion point). Equivalent to calling `gtk_text_buffer_get_mark()` to get the mark named "insert", but very slightly more efficient, and involves less typing.

buffer : a GtkTextBuffer
Returns : insertion point mark

gtk_text_buffer_get_selection_bound ()

```
GtkTextMark* gtk_text_buffer_get_selection_bound
                                                (GtkTextBuffer *buffer);
```

Returns the mark that represents the selection bound. Equivalent to calling `gtk_text_buffer_get_mark()` to get the mark named "selection_bound", but very slightly more efficient, and involves less typing.

The currently-selected text in *`buffer`* is the region between the "selection_bound" and "insert" marks. If "selection_bound" and "insert" are in the same place, then there is no current selection. `gtk_text_buffer_get_selection_bounds()` is another convenient function for handling the selection, if you just want to know whether there's a selection and what its bounds are.

`buffer` : a GtkTextBuffer

Returns : selection bound mark

---

gtk_text_buffer_place_cursor ()

```
void        gtk_text_buffer_place_cursor    (GtkTextBuffer *buffer,
                                             const GtkTextIter *where);
```

This function moves the "insert" and "selection_bound" marks simultaneously. If you move them to the same place in two steps with `gtk_text_buffer_move_mark()`, you will temporarily select a region in between their old and new locations, which can be pretty inefficient since the temporarily-selected region will force stuff to be recalculated. This function moves them as a unit, which can be optimized.

`buffer` : a GtkTextBuffer

`where` : where to put the cursor

---

gtk_text_buffer_select_range ()

```
void        gtk_text_buffer_select_range    (GtkTextBuffer *buffer,
                                             const GtkTextIter *ins,
                                             const GtkTextIter *bound);
```

This function moves the "insert" and "selection_bound" marks simultaneously. If you move them in two steps with `gtk_text_buffer_move_mark()`, you will temporarily select a region in between their old and new locations, which can be pretty inefficient since the temporarily-selected region will force stuff to be recalculated. This function moves them as a unit, which can be optimized.

buffer : a GtkTextBuffer
ins : where to put the "insert" mark
bound : where to put the "selection_bound" mark

Since 2.4

---

gtk_text_buffer_apply_tag ()

```
void        gtk_text_buffer_apply_tag       (GtkTextBuffer *buffer,
                                             GtkTextTag *tag,
                                             const GtkTextIter *start,
                                             const GtkTextIter *end);
```

Emits the "apply_tag" signal on *buffer*. The default handler for the signal applies *tag* to the given range. *start* and *end* do not have to be in order.

buffer : a GtkTextBuffer
tag : a GtkTextTag
start : one bound of range to be tagged
end : other bound of range to be tagged

---

gtk_text_buffer_remove_tag ()

```
void        gtk_text_buffer_remove_tag      (GtkTextBuffer *buffer,
                                             GtkTextTag *tag,
                                             const GtkTextIter *start,
                                             const GtkTextIter *end);
```

Emits the "remove_tag" signal. The default handler for the signal removes all occurrences of *tag* from the given range. *start* and *end* don't have to be in order.

buffer : a GtkTextBuffer
tag : a GtkTextTag
start : one bound of range to be untagged
end : other bound of range to be untagged

---

gtk_text_buffer_apply_tag_by_name ()

```
void        gtk_text_buffer_apply_tag_by_name
                                            (GtkTextBuffer *buffer,
                                             const gchar *name,
                                             const GtkTextIter *start,
                                             const GtkTextIter *end);
```

Calls `gtk_text_tag_table_lookup()` on the buffer's tag table to get a GtkTextTag, then calls `gtk_text_buffer_apply_tag()`.

buffer : a GtkTextBuffer
name : name of a named GtkTextTag
start : one bound of range to be tagged
end : other bound of range to be tagged

---

gtk_text_buffer_remove_tag_by_name ()

```
void        gtk_text_buffer_remove_tag_by_name
                                            (GtkTextBuffer *buffer,
                                             const gchar *name,
                                             const GtkTextIter *start,
                                             const GtkTextIter *end);
```

Calls `gtk_text_tag_table_lookup()` on the buffer's tag table to get a GtkTextTag, then calls `gtk_text_buffer_remove_tag()`.

buffer : a GtkTextBuffer
name : name of a GtkTextTag
start : one bound of range to be untagged
end : other bound of range to be untagged

gtk_text_buffer_remove_all_tags ()

```
void        gtk_text_buffer_remove_all_tags (GtkTextBuffer *buffer,
                                             const GtkTextIter *start,
                                             const GtkTextIter *end);
```

Removes all tags in the range between *start* and *end*. Be careful with this function; it could remove tags added in code unrelated to the code you're currently writing. That is, using this function is probably a bad idea if you have two or more unrelated code sections that add tags.

buffer : a GtkTextBuffer
start : one bound of range to be untagged
end : other bound of range to be untagged

gtk_text_buffer_create_tag ()

```
GtkTextTag* gtk_text_buffer_create_tag      (GtkTextBuffer *buffer,
                                             const gchar *tag_name,
                                             const gchar *first_property_name,
                                             ...);
```

Creates a tag and adds it to the tag table for *buffer*. Equivalent to calling gtk_text_tag_new() and then adding the tag to the buffer's tag table. The returned tag is owned by the buffer's tag table, so the ref count will be equal to one.

If *tag_name* is NULL, the tag is anonymous.

If *tag_name* is non-NULL, a tag called *tag_name* must not already exist in the tag table for this buffer.

The *first_property_name* argument and subsequent arguments are a list of properties to set on the tag, as with g_object_set().

buffer : a GtkTextBuffer
tag_name : name of the new tag, or NULL
first_property_name : name of first property to set, or NULL
... : NULL-terminated list of property names and values
Returns : a new tag

gtk_text_buffer_get_iter_at_line_offset ()

```
void        gtk_text_buffer_get_iter_at_line_offset
                                            (GtkTextBuffer *buffer,
                                             GtkTextIter *iter,
                                             gint line_number,
                                             gint char_offset);
```

Obtains an iterator pointing to *char_offset* within the given line. The *char_offset* must exist, offsets off the end of the line are not allowed. Note *characters*, not bytes; UTF-8 may encode one character as multiple bytes.

buffer :	a GtkTextBuffer
iter :	iterator to initialize
line_number :	line number counting from 0
char_offset :	char offset from start of line

gtk_text_buffer_get_iter_at_offset ()

```
void        gtk_text_buffer_get_iter_at_offset
                                            (GtkTextBuffer *buffer,
                                             GtkTextIter *iter,
                                             gint char_offset);
```

Initializes *iter* to a position *char_offset* chars from the start of the entire buffer. If *char_offset* is -1 or greater than the number of characters in the buffer, *iter* is initialized to the end iterator, the iterator one past the last valid character in the buffer.

buffer :	a GtkTextBuffer
iter :	iterator to initialize
char_offset :	char offset from start of buffer, counting from 0, or -1

gtk_text_buffer_get_iter_at_line ()

```
void        gtk_text_buffer_get_iter_at_line
                                            (GtkTextBuffer *buffer,
```

```
                                                GtkTextIter *iter,
                                                gint line_number);
```

Initializes *iter* to the start of the given line.

buffer : a GtkTextBuffer
iter : iterator to initialize
line_number : line number counting from 0

---

gtk_text_buffer_get_iter_at_line_index ()

```
void        gtk_text_buffer_get_iter_at_line_index
                                               (GtkTextBuffer *buffer,
                                                GtkTextIter *iter,
                                                gint line_number,
                                                gint byte_index);
```

Obtains an iterator pointing to *byte_index* within the given line. *byte_index* must be the start of a UTF-8 character, and must not be beyond the end of the line. Note *bytes*, not characters; UTF-8 may encode one character as multiple bytes.

buffer : a GtkTextBuffer
iter : iterator to initialize
line_number : line number counting from 0
byte_index : byte index from start of line

---

gtk_text_buffer_get_iter_at_mark ()

```
void        gtk_text_buffer_get_iter_at_mark
                                               (GtkTextBuffer *buffer,
                                                GtkTextIter *iter,
                                                GtkTextMark *mark);
```

Initializes *iter* with the current position of *mark*.

buffer : a GtkTextBuffer

iter : iterator to initialize
mark : a GtkTextMark in *buffer*

---

gtk_text_buffer_get_iter_at_child_anchor ()

```
void        gtk_text_buffer_get_iter_at_child_anchor
                                            (GtkTextBuffer *buffer,
                                             GtkTextIter *iter,
                                             GtkTextChildAnchor *anchor);
```

Obtains the location of *anchor* within *buffer*.

buffer : a GtkTextBuffer
iter : an iterator to be initialized
anchor : a child anchor that appears in *buffer*

---

gtk_text_buffer_get_start_iter ()

```
void        gtk_text_buffer_get_start_iter  (GtkTextBuffer *buffer,
                                             GtkTextIter *iter);
```

Initialized *iter* with the first position in the text buffer. This is the same as using gtk_text_buffer_get_iter_at_offset() to get the iter at character offset 0.

buffer : a GtkTextBuffer
iter : iterator to initialize

---

gtk_text_buffer_get_end_iter ()

```
void        gtk_text_buffer_get_end_iter    (GtkTextBuffer *buffer,
                                             GtkTextIter *iter);
```

Initializes *iter* with the "end iterator," one past the last valid character in the text buffer. If dereferenced with gtk_text_iter_get_char(), the end iterator has a character value of 0. The entire buffer lies in the range from the first position in the buffer (call gtk_text_buffer_get_start_iter() to get character position 0) to the

end iterator.

buffer : a GtkTextBuffer
iter : iterator to initialize

---

gtk_text_buffer_get_bounds ()

```
void        gtk_text_buffer_get_bounds      (GtkTextBuffer *buffer,
                                             GtkTextIter *start,
                                             GtkTextIter *end);
```

Retrieves the first and last iterators in the buffer, i.e. the entire buffer lies within the range [*start*,*end*).

buffer : a GtkTextBuffer
start : iterator to initialize with first position in the buffer
end : iterator to initialize with the end iterator

---

gtk_text_buffer_get_modified ()

```
gboolean    gtk_text_buffer_get_modified    (GtkTextBuffer *buffer);
```

Indicates whether the buffer has been modified since the last call to gtk_text_buffer_set_modified() set the modification flag to FALSE. Used for example to enable a "save" function in a text editor.

buffer : a GtkTextBuffer
Returns : TRUE if the buffer has been modified

---

gtk_text_buffer_set_modified ()

```
void        gtk_text_buffer_set_modified    (GtkTextBuffer *buffer,
                                             gboolean setting);
```

Used to keep track of whether the buffer has been modified since the last time it was saved. Whenever the buffer is saved to disk, call gtk_text_buffer_set_modified (*buffer*, FALSE). When the buffer is modified, it will

automatically toggled on the modified bit again. When the modified bit flips, the buffer emits a "modified_changed" signal.

buffer : a GtkTextBuffer
setting : modification flag setting

---

gtk_text_buffer_delete_selection ()

```
gboolean    gtk_text_buffer_delete_selection
                                            (GtkTextBuffer *buffer,
                                             gboolean interactive,
                                             gboolean default_editable);
```

Deletes the range between the "insert" and "selection_bound" marks, that is, the currently-selected text. If *interactive* is TRUE, the editability of the selection will be considered (users can't delete uneditable text).

buffer : a GtkTextBuffer
interactive : whether the deletion is caused by user interaction
default_editable : whether the buffer is editable by default
Returns : whether there was a non-empty selection to delete

---

gtk_text_buffer_paste_clipboard ()

```
void        gtk_text_buffer_paste_clipboard (GtkTextBuffer *buffer,
                                             GtkClipboard *clipboard,
                                             GtkTextIter *override_location,
                                             gboolean default_editable);
```

Pastes the contents of a clipboard at the insertion point, or at *override_location*. (Note: pasting is asynchronous, that is, we'll ask for the paste data and return, and at some point later after the main loop runs, the paste data will be inserted.)

buffer : a GtkTextBuffer
clipboard : the GtkClipboard to paste from
override_location : location to insert pasted text, or NULL for at the cursor
default_editable : whether the buffer is editable by default

---

gtk_text_buffer_copy_clipboard ()

```
void        gtk_text_buffer_copy_clipboard  (GtkTextBuffer *buffer,
                                             GtkClipboard *clipboard);
```

Copies the currently-selected text to a clipboard.

buffer : a GtkTextBuffer
clipboard : the GtkClipboard object to copy to.

---

gtk_text_buffer_cut_clipboard ()

```
void        gtk_text_buffer_cut_clipboard   (GtkTextBuffer *buffer,
                                             GtkClipboard *clipboard,
                                             gboolean default_editable);
```

Copies the currently-selected text to a clipboard, then deletes said text if it's editable.

buffer : a GtkTextBuffer
clipboard : the GtkClipboard object to cut to.
default_editable : default editability of the buffer

---

gtk_text_buffer_get_selection_bounds ()

```
gboolean    gtk_text_buffer_get_selection_bounds
                                            (GtkTextBuffer *buffer,
                                             GtkTextIter *start,
                                             GtkTextIter *end);
```

Returns TRUE if some text is selected; places the bounds of the selection in *start* and *end* (if the selection has length 0, then *start* and *end* are filled in with the same value). *start* and *end* will be in ascending order. If *start* and *end* are NULL, then they are not filled in, but the return value still indicates whether text is selected.

buffer : a GtkTextBuffer a GtkTextBuffer

start : iterator to initialize with selection start
end : iterator to initialize with selection end
Returns : whether the selection has nonzero length

---

gtk_text_buffer_begin_user_action ()

```
void        gtk_text_buffer_begin_user_action
                                            (GtkTextBuffer *buffer);
```

Called to indicate that the buffer operations between here and a call to `gtk_text_buffer_end_user_action()` are part of a single user-visible operation. The operations between `gtk_text_buffer_begin_user_action()` and `gtk_text_buffer_end_user_action()` can then be grouped when creating an undo stack. GtkTextBuffer maintains a count of calls to `gtk_text_buffer_begin_user_action()` that have not been closed with a call to `gtk_text_buffer_end_user_action()`, and emits the "begin_user_action" and "end_user_action" signals only for the outermost pair of calls. This allows you to build user actions from other user actions.

The "interactive" buffer mutation functions, such as `gtk_text_buffer_insert_interactive()`, automatically call begin/end user action around the buffer operations they perform, so there's no need to add extra calls if you user action consists solely of a single call to one of those functions.

buffer : a GtkTextBuffer

---

gtk_text_buffer_end_user_action ()

```
void        gtk_text_buffer_end_user_action (GtkTextBuffer *buffer);
```

Should be paired with a call to `gtk_text_buffer_begin_user_action()`. See that function for a full explanation.

buffer : a GtkTextBuffer

---

gtk_text_buffer_add_selection_clipboard ()

```
void        gtk_text_buffer_add_selection_clipboard
                                            (GtkTextBuffer *buffer,
```

```
                                                GtkClipboard *clipboard);
```

Adds *clipboard* to the list of clipboards in which the selection contents of *buffer* are available. In most cases, *clipboard* will be the GtkClipboard of type GDK_SELECTION_PRIMARY for a view of *buffer*.

buffer : a GtkTextBuffer

clipboard : a GtkClipboard

---

gtk_text_buffer_remove_selection_clipboard ()

```
void        gtk_text_buffer_remove_selection_clipboard
                                                (GtkTextBuffer *buffer,
                                                 GtkClipboard *clipboard);
```

Removes a GtkClipboard added with gtk_text_buffer_add_selection_clipboard()

buffer : a GtkTextBuffer

clipboard : a GtkClipboard added to *buffer* by gtk_text_buffer_add_selection_clipboard().

Properties

The "tag-table" property

```
  "tag-table"           GtkTextTagTable       : Read / Write / Construct Only
```

Text Tag Table.

Signals

The "apply-tag" signal

```
void        user_function                   (GtkTextBuffer *textbuffer,
                                             GtkTextTag *arg1,
                                             GtkTextIter *arg2,
                                             GtkTextIter *arg3,
```

```
                                        gpointer user_data);
```

textbuffer : the object which received the signal.
arg1 :
arg2 :
arg3 :
user_data : user data set when the signal handler was connected.

The "begin-user-action" signal

```
void        user_function               (GtkTextBuffer *textbuffer,
                                         gpointer user_data);
```

textbuffer : the object which received the signal.
user_data : user data set when the signal handler was connected.

The "changed" signal

```
void        user_function               (GtkTextBuffer *textbuffer,
                                         gpointer user_data);
```

textbuffer : the object which received the signal.
user_data : user data set when the signal handler was connected.

The "delete-range" signal

```
void        user_function               (GtkTextBuffer *textbuffer,
                                         GtkTextIter *arg1,
                                         GtkTextIter *arg2,
                                         gpointer user_data);
```

textbuffer : the object which received the signal.
arg1 :

arg2 :

user_data : user data set when the signal handler was connected.

The "end-user-action" signal

```
void        user_function                   (GtkTextBuffer *textbuffer,
                                             gpointer user_data);
```

textbuffer : the object which received the signal.

user_data : user data set when the signal handler was connected.

The "insert-child-anchor" signal

```
void        user_function                   (GtkTextBuffer *textbuffer,
                                             GtkTextIter *arg1,
                                             GtkTextChildAnchor *arg2,
                                             gpointer user_data);
```

textbuffer : the object which received the signal.

arg1 :

arg2 :

user_data : user data set when the signal handler was connected.

The "insert-pixbuf" signal

```
void        user_function                   (GtkTextBuffer *textbuffer,
                                             GtkTextIter *arg1,
                                             GdkPixbuf *arg2,
                                             gpointer user_data);
```

textbuffer : the object which received the signal.

arg1 :

arg2 :

user_data : user data set when the signal handler was connected.

The "insert-text" signal

```
void        user_function                   (GtkTextBuffer *textbuffer,
                                             GtkTextIter *arg1,
                                             gchar *arg2,
                                             gint arg3,
                                             gpointer user_data);
```

textbuffer : the object which received the signal.
arg1 :
arg2 :
arg3 :
user_data : user data set when the signal handler was connected.

The "mark-deleted" signal

```
void        user_function                   (GtkTextBuffer *textbuffer,
                                             GtkTextMark *arg1,
                                             gpointer user_data);
```

textbuffer : the object which received the signal.
arg1 :
user_data : user data set when the signal handler was connected.

The "mark-set" signal

```
void        user_function                   (GtkTextBuffer *textbuffer,
                                             GtkTextIter *arg1,
                                             GtkTextMark *arg2,
                                             gpointer user_data);
```

textbuffer : the object which received the signal.
arg1 :

arg2 :
user_data : user data set when the signal handler was connected.

The "modified-changed" signal

```
void        user_function                   (GtkTextBuffer *textbuffer,
                                             gpointer user_data);
```

textbuffer : the object which received the signal.
user_data : user data set when the signal handler was connected.

The "remove-tag" signal

```
void        user_function                   (GtkTextBuffer *textbuffer,
                                             GtkTextTag *arg1,
                                             GtkTextIter *arg2,
                                             GtkTextIter *arg3,
                                             gpointer user_data);
```

textbuffer : the object which received the signal.
arg1 :
arg2 :
arg3 :
user_data : user data set when the signal handler was connected.

See Also

GtkTextView, GtkTextIter, GtkTextMark

<< GtkTextMark **GtkTextTag >>**

GtkTextTag

GtkTextTag — A tag that can be applied to text in a GtkTextBuffer

Synopsis

```
#include <gtk/gtk.h>


            GtkTextTag;
enum        GtkWrapMode;
            GtkTextAttributes;
GtkTextTag* gtk_text_tag_new                (const gchar *name);
gint        gtk_text_tag_get_priority       (GtkTextTag *tag);
void        gtk_text_tag_set_priority       (GtkTextTag *tag,
                                             gint priority);
gboolean    gtk_text_tag_event              (GtkTextTag *tag,
                                             GObject *event_object,
                                             GdkEvent *event,
                                             const GtkTextIter *iter);
            GtkTextAppearance;
GtkTextAttributes* gtk_text_attributes_new   (void);
GtkTextAttributes* gtk_text_attributes_copy  (GtkTextAttributes *src);
void        gtk_text_attributes_copy_values (GtkTextAttributes *src,
                                             GtkTextAttributes *dest);
void        gtk_text_attributes_unref       (GtkTextAttributes *values);
void        gtk_text_attributes_ref         (GtkTextAttributes *values);
```

Object Hierarchy

```
  GObject
   +----GtkTextTag
```

Properties

```
"background"           gchararray           : Write
"background-full-height" gboolean             : Read / Write
"background-full-height-set" gboolean             : Read / Write
"background-gdk"       GdkColor             : Read / Write
"background-set"       gboolean             : Read / Write
"background-stipple"   GdkPixmap            : Read / Write
"background-stipple-set" gboolean             : Read / Write
"direction"            GtkTextDirection     : Read / Write
"editable"             gboolean             : Read / Write
"editable-set"         gboolean             : Read / Write
"family"               gchararray           : Read / Write
"family-set"           gboolean             : Read / Write
"font"                 gchararray           : Read / Write
"font-desc"            PangoFontDescription : Read / Write
"foreground"           gchararray           : Write
"foreground-gdk"       GdkColor             : Read / Write
"foreground-set"       gboolean             : Read / Write
"foreground-stipple"   GdkPixmap            : Read / Write
"foreground-stipple-set" gboolean             : Read / Write
"indent"               gint                 : Read / Write
"indent-set"           gboolean             : Read / Write
"invisible"            gboolean             : Read / Write
"invisible-set"        gboolean             : Read / Write
"justification"        GtkJustification     : Read / Write
"justification-set"    gboolean             : Read / Write
"language"             gchararray           : Read / Write
"language-set"         gboolean             : Read / Write
"left-margin"          gint                 : Read / Write
"left-margin-set"      gboolean             : Read / Write
"name"                 gchararray           : Read / Write / Construct Only
"pixels-above-lines"   gint                 : Read / Write
"pixels-above-lines-set" gboolean             : Read / Write
"pixels-below-lines"   gint                 : Read / Write
"pixels-below-lines-set" gboolean             : Read / Write
"pixels-inside-wrap"   gint                 : Read / Write
"pixels-inside-wrap-set" gboolean             : Read / Write
"right-margin"         gint                 : Read / Write
"right-margin-set"     gboolean             : Read / Write
"rise"                 gint                 : Read / Write
```

```
  "rise-set"            gboolean            : Read / Write
  "scale"               gdouble             : Read / Write
  "scale-set"           gboolean            : Read / Write
  "size"                gint                : Read / Write
  "size-points"         gdouble             : Read / Write
  "size-set"            gboolean            : Read / Write
  "stretch"             PangoStretch        : Read / Write
  "stretch-set"         gboolean            : Read / Write
  "strikethrough"       gboolean            : Read / Write
  "strikethrough-set"   gboolean            : Read / Write
  "style"               PangoStyle          : Read / Write
  "style-set"           gboolean            : Read / Write
  "tabs"                PangoTabArray       : Read / Write
  "tabs-set"            gboolean            : Read / Write
  "underline"           PangoUnderline      : Read / Write
  "underline-set"       gboolean            : Read / Write
  "variant"             PangoVariant        : Read / Write
  "variant-set"         gboolean            : Read / Write
  "weight"              gint                : Read / Write
  "weight-set"          gboolean            : Read / Write
  "wrap-mode"           GtkWrapMode         : Read / Write
  "wrap-mode-set"       gboolean            : Read / Write
```

Signal Prototypes

```
"event"     gboolean    user_function      (GtkTextTag *texttag,
                                            GObject *arg1,
                                            GdkEvent *event,
                                            GtkTextIter *arg2,
                                            gpointer user_data);
```

Description

You may wish to begin by reading the text widget conceptual overview which gives an overview of all the objects and data types related to the text widget and how they work together.

Tags should be in the GtkTextTagTable for a given GtkTextBuffer before using them with that buffer.

`gtk_text_buffer_create_tag()` is the best way to create tags. See gtk-demo for numerous examples.

The "invisible" property was not implemented for GTK+ 2.0; it's planned to be implemented in future releases.

Details

GtkTextTag

```
typedef struct _GtkTextTag GtkTextTag;
```

---

enum GtkWrapMode

```
typedef enum
{
  GTK_WRAP_NONE,
  GTK_WRAP_CHAR,
  GTK_WRAP_WORD,
  GTK_WRAP_WORD_CHAR
} GtkWrapMode;
```

Describes a type of line wrapping.

GTK_WRAP_NONE	do not wrap lines; just make the text area wider
GTK_WRAP_CHAR	wrap text, breaking lines anywhere the cursor can appear (between characters, usually - if you want to be technical, between graphemes, see pango_get_log_attrs())
GTK_WRAP_WORD	wrap text, breaking lines in between words
GTK_WRAP_WORD_CHAR	wrap text, breaking lines in between words, or if that is not enough, also between graphemes.

GtkTextAttributes

```
typedef struct {
  GtkTextAppearance appearance;

  GtkJustification justification;
  GtkTextDirection direction;

  /* Individual chunks of this can be set/unset as a group */
  PangoFontDescription *font;
```

```
  gdouble font_scale;

  gint left_margin;

  gint indent;

  gint right_margin;

  gint pixels_above_lines;

  gint pixels_below_lines;

  gint pixels_inside_wrap;

  PangoTabArray *tabs;

  GtkWrapMode wrap_mode;        /* How to handle wrap-around for this tag.
                                 * Must be GTK_WRAPMODE_CHAR,
                                 * GTK_WRAPMODE_NONE, GTK_WRAPMODE_WORD
                                 */

  PangoLanguage *language;

  /* hide the text  */
  guint invisible : 1;

  /* Background is fit to full line height rather than
   * baseline +/- ascent/descent (font height)
   */
  guint bg_full_height : 1;

  /* can edit this text */
  guint editable : 1;

  /* colors are allocated etc. */
  guint realized : 1;
} GtkTextAttributes;
```

Using GtkTextAttributes directly should rarely be necessary. It's primarily useful with `gtk_text_iter_get_attributes()`. As with most GTK+ structs, the fields in this struct should only be read, never modified directly.

GtkTextAppearance *appearance*; pointer to sub-struct containing certain attributes

GtkJustification *justification*;

GtkTextDirection *direction*;

PangoFontDescription **font*;

gdouble *font_scale*;

gint *left_margin*;
gint *indent*;
gint *right_margin*;
gint *pixels_above_lines*;
gint *pixels_below_lines*;
gint *pixels_inside_wrap*;
PangoTabArray **tabs*;
GtkWrapMode *wrap_mode*;
PangoLanguage **language*;
guint *invisible* : 1;
guint *bg_full_height* : 1;
guint *editable* : 1;
guint *realized* : 1;

---

gtk_text_tag_new ()

```
GtkTextTag* gtk_text_tag_new                (const gchar *name);
```

Creates a GtkTextTag. Configure the tag using object arguments, i.e. using `g_object_set()`.

name : tag name, or `NULL`
Returns : a new GtkTextTag

---

gtk_text_tag_get_priority ()

```
gint        gtk_text_tag_get_priority       (GtkTextTag *tag);
```

Get the tag priority.

tag : a GtkTextTag
Returns : The tag's priority.

---

gtk_text_tag_set_priority ()

```
void        gtk_text_tag_set_priority       (GtkTextTag *tag,
                                             gint priority);
```

Sets the priority of a GtkTextTag. Valid priorities are start at 0 and go to one less than `gtk_text_tag_table_get_size()`. Each tag in a table has a unique priority; setting the priority of one tag shifts the priorities of all the other tags in the table to maintain a unique priority for each tag. Higher priority tags "win" if two tags both set the same text attribute. When adding a tag to a tag table, it will be assigned the highest priority in the table by default; so normally the precedence of a set of tags is the order in which they were added to the table, or created with `gtk_text_buffer_create_tag()`, which adds the tag to the buffer's table automatically.

tag : a GtkTextTag
priority : the new priority

---

gtk_text_tag_event ()

```
gboolean    gtk_text_tag_event              (GtkTextTag *tag,
                                             GObject *event_object,
                                             GdkEvent *event,
                                             const GtkTextIter *iter);
```

Emits the "event" signal on the GtkTextTag.

tag : a GtkTextTag
event_object : object that received the event, such as a widget
event : the event
iter : location where the event was received
Returns : result of signal emission (whether the event was handled)

GtkTextAppearance

```
typedef struct {
  GdkColor bg_color;
  GdkColor fg_color;
  GdkBitmap *bg_stipple;
  GdkBitmap *fg_stipple;

  /* super/subscript rise, can be negative */
```

```
  gint rise;

  guint underline : 4;          /* PangoUnderline */
  guint strikethrough : 1;

  /* Whether to use background-related values; this is irrelevant for
   * the values struct when in a tag, but is used for the composite
   * values struct; it's true if any of the tags being composited
   * had background stuff set.
   */
  guint draw_bg : 1;

  /* These are only used when we are actually laying out and rendering
   * a paragraph; not when a GtkTextAppearance is part of a
   * GtkTextAttributes.
   */
  guint inside_selection : 1;
  guint is_text : 1;
} GtkTextAppearance;
```

---

gtk_text_attributes_new ()

```
GtkTextAttributes* gtk_text_attributes_new  (void);
```

Creates a GtkTextAttributes, which describes a set of properties on some text.

Returns : a new GtkTextAttributes

---

gtk_text_attributes_copy ()

```
GtkTextAttributes* gtk_text_attributes_copy (GtkTextAttributes *src);
```

Copies *src* and returns a new GtkTextAttributes.

src : a GtkTextAttributes to be copied

Returns : a copy of *src*

---

gtk_text_attributes_copy_values ()

```
void        gtk_text_attributes_copy_values (GtkTextAttributes *src,
                                             GtkTextAttributes *dest);
```

Copies the values from *src* to *dest* so that *dest* has the same values as *src*. Frees existing values in *dest*.

src : a GtkTextAttributes
dest : another GtkTextAttributes

---

gtk_text_attributes_unref ()

```
void        gtk_text_attributes_unref       (GtkTextAttributes *values);
```

Decrements the reference count on *values*, freeing the structure if the reference count reaches 0.

values : a GtkTextAttributes

---

gtk_text_attributes_ref ()

```
void        gtk_text_attributes_ref         (GtkTextAttributes *values);
```

Increments the reference count on *values*.

values : a GtkTextAttributes

Properties

The "background" property

```
  "background"           gchararray           : Write
```

Background color as a string.

Default value: NULL

The "background-full-height" property

```
"background-full-height" gboolean              : Read / Write
```

Whether the background color fills the entire line height or only the height of the tagged characters.

Default value: FALSE

The "background-full-height-set" property

```
"background-full-height-set" gboolean              : Read / Write
```

Whether this tag affects background height.

Default value: FALSE

The "background-gdk" property

```
"background-gdk"       GdkColor              : Read / Write
```

Background color as a (possibly unallocated) GdkColor.

The "background-set" property

```
"background-set"       gboolean              : Read / Write
```

Whether this tag affects the background color.

Default value: FALSE

The "background-stipple" property

```
"background-stipple"   GdkPixmap             : Read / Write
```

Bitmap to use as a mask when drawing the text background.

The "background-stipple-set" property

```
"background-stipple-set" gboolean              : Read / Write
```

Whether this tag affects the background stipple.

Default value: FALSE

The "direction" property

```
"direction"            GtkTextDirection      : Read / Write
```

Text direction, e.g. right-to-left or left-to-right.

Default value: GTK_TEXT_DIR_LTR

The "editable" property

```
"editable"             gboolean              : Read / Write
```

Whether the text can be modified by the user.

Default value: TRUE

The "editable-set" property

```
"editable-set"         gboolean              : Read / Write
```

Whether this tag affects text editability.

Default value: FALSE

The "family" property

```
"family"               gchararray            : Read / Write
```

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

Default value: NULL

The "family-set" property

```
"family-set"           gboolean              : Read / Write
```

Whether this tag affects the font family.

Default value: FALSE

The "font" property

```
"font"                 gchararray            : Read / Write
```

Font description as a string, e.g. "Sans Italic 12".

Default value: NULL

The "font-desc" property

```
"font-desc"           PangoFontDescription  : Read / Write
```

Font description as a PangoFontDescription struct.

The "foreground" property

```
"foreground"          gchararray            : Write
```

Foreground color as a string.

Default value: NULL

The "foreground-gdk" property

```
"foreground-gdk"      GdkColor              : Read / Write
```

Foreground color as a (possibly unallocated) GdkColor.

The "foreground-set" property

```
"foreground-set"      gboolean              : Read / Write
```

Whether this tag affects the foreground color.

Default value: FALSE

The "foreground-stipple" property

```
  "foreground-stipple"   GdkPixmap             : Read / Write
```

Bitmap to use as a mask when drawing the text foreground.

The "foreground-stipple-set" property

```
  "foreground-stipple-set" gboolean              : Read / Write
```

Whether this tag affects the foreground stipple.

Default value: FALSE

The "indent" property

```
  "indent"               gint                  : Read / Write
```

Amount to indent the paragraph, in pixels.

Default value: 0

The "indent-set" property

```
  "indent-set"           gboolean              : Read / Write
```

Whether this tag affects indentation.

Default value: FALSE

The "invisible" property

```
  "invisible"           gboolean            : Read / Write
```

Whether this text is hidden. Not implemented in GTK 2.0.

Default value: FALSE

The "invisible-set" property

```
  "invisible-set"       gboolean            : Read / Write
```

Whether this tag affects text visibility.

Default value: FALSE

The "justification" property

```
  "justification"       GtkJustification    : Read / Write
```

Left, right, or center justification.

Default value: GTK_JUSTIFY_LEFT

The "justification-set" property

```
  "justification-set"   gboolean            : Read / Write
```

Whether this tag affects paragraph justification.

Default value: FALSE

The "language" property

```
"language"              gchararray            : Read / Write
```

The language this text is in, as an ISO code. Pango can use this as a hint when rendering the text. If not set, an appropriate default will be used.

Default value: NULL

The "language-set" property

```
"language-set"          gboolean              : Read / Write
```

Whether this tag affects the language the text is rendered as.

Default value: FALSE

The "left-margin" property

```
"left-margin"           gint                  : Read / Write
```

Width of the left margin in pixels.

Allowed values: >= 0

Default value: 0

The "left-margin-set" property

```
"left-margin-set"       gboolean              : Read / Write
```

Whether this tag affects the left margin.

Default value: FALSE

The "name" property

```
  "name"                gchararray           : Read / Write / Construct Only
```

Name used to refer to the text tag. NULL for anonymous tags.

Default value: NULL

The "pixels-above-lines" property

```
  "pixels-above-lines"   gint                 : Read / Write
```

Pixels of blank space above paragraphs.

Allowed values: >= 0

Default value: 0

The "pixels-above-lines-set" property

```
  "pixels-above-lines-set" gboolean             : Read / Write
```

Whether this tag affects the number of pixels above lines.

Default value: FALSE

The "pixels-below-lines" property

```
  "pixels-below-lines"   gint                 : Read / Write
```

Pixels of blank space below paragraphs.

Allowed values: >= 0

Default value: 0

The "pixels-below-lines-set" property

```
"pixels-below-lines-set" gboolean              : Read / Write
```

Whether this tag affects the number of pixels above lines.

Default value: FALSE

The "pixels-inside-wrap" property

```
"pixels-inside-wrap"   gint                    : Read / Write
```

Pixels of blank space between wrapped lines in a paragraph.

Allowed values: >= 0

Default value: 0

The "pixels-inside-wrap-set" property

```
"pixels-inside-wrap-set" gboolean              : Read / Write
```

Whether this tag affects the number of pixels between wrapped lines.

Default value: FALSE

The "right-margin" property

```
"right-margin"          gint                : Read / Write
```

Width of the right margin in pixels.

Allowed values: >= 0

Default value: 0

The "right-margin-set" property

```
"right-margin-set"      gboolean            : Read / Write
```

Whether this tag affects the right margin.

Default value: FALSE

The "rise" property

```
"rise"                  gint                : Read / Write
```

Offset of text above the baseline (below the baseline if rise is negative) in pixels.

Default value: 0

The "rise-set" property

```
"rise-set"              gboolean            : Read / Write
```

Whether this tag affects the rise.

Default value: FALSE

The "scale" property

"scale"	gdouble	: Read / Write

Font size as a scale factor relative to the default font size. This properly adapts to theme changes etc. so is recommended. Pango predefines some scales such as PANGO_SCALE_X_LARGE.

Allowed values: >= 0

Default value: 1

The "scale-set" property

"scale-set"	gboolean	: Read / Write

Whether this tag scales the font size by a factor.

Default value: FALSE

The "size" property

"size"	gint	: Read / Write

Font size in Pango units.

Allowed values: >= 0

Default value: 0

The "size-points" property

"size-points"	gdouble	: Read / Write

Font size in points.

Allowed values: >= 0

Default value: 0

The "size-set" property

```
"size-set"            gboolean              : Read / Write
```

Whether this tag affects the font size.

Default value: FALSE

The "stretch" property

```
"stretch"             PangoStretch          : Read / Write
```

Font stretch as a PangoStretch, e.g. PANGO_STRETCH_CONDENSED.

Default value: PANGO_STRETCH_NORMAL

The "stretch-set" property

```
"stretch-set"         gboolean              : Read / Write
```

Whether this tag affects the font stretch.

Default value: FALSE

The "strikethrough" property

```
"strikethrough"        gboolean              : Read / Write
```

Whether to strike through the text.

Default value: FALSE

The "strikethrough-set" property

```
"strikethrough-set"    gboolean              : Read / Write
```

Whether this tag affects strikethrough.

Default value: FALSE

The "style" property

```
"style"                PangoStyle            : Read / Write
```

Font style as a PangoStyle, e.g. PANGO_STYLE_ITALIC.

Default value: PANGO_STYLE_NORMAL

The "style-set" property

```
"style-set"            gboolean              : Read / Write
```

Whether this tag affects the font style.

Default value: FALSE

The "tabs" property

```
"tabs"                PangoTabArray         : Read / Write
```

Custom tabs for this text.

The "tabs-set" property

```
"tabs-set"            gboolean              : Read / Write
```

Whether this tag affects tabs.

Default value: FALSE

The "underline" property

```
"underline"           PangoUnderline        : Read / Write
```

Style of underline for this text.

Default value: PANGO_UNDERLINE_NONE

The "underline-set" property

```
"underline-set"       gboolean              : Read / Write
```

Whether this tag affects underlining.

Default value: FALSE

The "variant" property

```
```

```
"variant"              PangoVariant          : Read / Write
```

Font variant as a PangoVariant, e.g. PANGO_VARIANT_SMALL_CAPS.

Default value: PANGO_VARIANT_NORMAL

The "variant-set" property

```
"variant-set"          gboolean              : Read / Write
```

Whether this tag affects the font variant.

Default value: FALSE

The "weight" property

```
"weight"               gint                  : Read / Write
```

Font weight as an integer, see predefined values in PangoWeight; for example, PANGO_WEIGHT_BOLD.

Allowed values: >= 0

Default value: 400

The "weight-set" property

```
"weight-set"           gboolean              : Read / Write
```

Whether this tag affects the font weight.

Default value: FALSE

The "wrap-mode" property

```
  "wrap-mode"           GtkWrapMode           : Read / Write
```

Whether to wrap lines never, at word boundaries, or at character boundaries.

Default value: GTK_WRAP_NONE

The "wrap-mode-set" property

```
  "wrap-mode-set"       gboolean              : Read / Write
```

Whether this tag affects line wrap mode.

Default value: FALSE

Signals

The "event" signal

```
gboolean    user_function                  (GtkTextTag *texttag,
                                            GObject *arg1,
                                            GdkEvent *event,
                                            GtkTextIter *arg2,
                                            gpointer user_data);
```

texttag : the object which received the signal.
arg1 :
event :
arg2 :
user_data : user data set when the signal handler was connected.
Returns :

<< GtkTextBuffer **GtkTextTagTable >>**

GtkTextTagTable

GtkTextTagTable — Collection of tags that can be used together

Synopsis

```
#include <gtk/gtk.h>


            GtkTextTagTable;
void        (*GtkTextTagTableForeach)       (GtkTextTag *tag,
                                             gpointer data);
GtkTextTagTable* gtk_text_tag_table_new     (void);
void        gtk_text_tag_table_add          (GtkTextTagTable *table,
                                             GtkTextTag *tag);
void        gtk_text_tag_table_remove       (GtkTextTagTable *table,
                                             GtkTextTag *tag);
GtkTextTag* gtk_text_tag_table_lookup       (GtkTextTagTable *table,
                                             const gchar *name);
void        gtk_text_tag_table_foreach      (GtkTextTagTable *table,
                                             GtkTextTagTableForeach func,
                                             gpointer data);
gint        gtk_text_tag_table_get_size     (GtkTextTagTable *table);
```

Object Hierarchy

```
  GObject
   +----GtkTextTagTable
```

Signal Prototypes

```
"tag-added" void        user_function      (GtkTextTagTable *texttagtable,
                                            GtkTextTag *arg1,
                                            gpointer user_data);
"tag-changed"
            void        user_function      (GtkTextTagTable *texttagtable,
                                            GtkTextTag *arg1,
                                            gboolean arg2,
                                            gpointer user_data);
"tag-removed"
            void        user_function      (GtkTextTagTable *texttagtable,
                                            GtkTextTag *arg1,
                                            gpointer user_data);
```

Description

You may wish to begin by reading the text widget conceptual overview which gives an overview of all the objects and data types related to the text widget and how they work together.

Details

GtkTextTagTable

```
typedef struct _GtkTextTagTable GtkTextTagTable;
```

GtkTextTagTableForeach ()

```
void        (*GtkTextTagTableForeach)       (GtkTextTag *tag,
                                             gpointer data);
```

tag:

data:

---

gtk_text_tag_table_new ()

```
GtkTextTagTable* gtk_text_tag_table_new     (void);
```

Creates a new GtkTextTagTable. The table contains no tags by default.

Returns : a new GtkTextTagTable

---

gtk_text_tag_table_add ()

```
void        gtk_text_tag_table_add          (GtkTextTagTable *table,
                                             GtkTextTag *tag);
```

Add a tag to the table. The tag is assigned the highest priority in the table.

tag must not be in a tag table already, and may not have the same name as an already-added tag.

table : a GtkTextTagTable
tag : a GtkTextTag

---

gtk_text_tag_table_remove ()

```
void        gtk_text_tag_table_remove       (GtkTextTagTable *table,
                                             GtkTextTag *tag);
```

Remove a tag from the table. This will remove the table's reference to the tag, so be careful - the tag will end up destroyed if you don't have a reference to it.

table : a GtkTextTagTable
tag : a GtkTextTag

---

gtk_text_tag_table_lookup ()

```
GtkTextTag* gtk_text_tag_table_lookup       (GtkTextTagTable *table,
                                             const gchar *name);
```

Look up a named tag.

table : a GtkTextTagTable
name : name of a tag
Returns : The tag, or NULL if none by that name is in the table.

---

gtk_text_tag_table_foreach ()

```
void        gtk_text_tag_table_foreach      (GtkTextTagTable *table,
                                             GtkTextTagTableForeach func,
                                             gpointer data);
```

Calls *func* on each tag in *table*, with user data *data*.

table : a GtkTextTagTable
func : a function to call on each tag
data : user data

---

gtk_text_tag_table_get_size ()

```
gint        gtk_text_tag_table_get_size     (GtkTextTagTable *table);
```

Returns the size of the table (number of tags)

table : a GtkTextTagTable
Returns : number of tags in *table*

Signals

The "tag-added" signal

```
void        user_function                   (GtkTextTagTable *texttagtable,
                                             GtkTextTag *arg1,
                                             gpointer user_data);
```

texttagtable : the object which received the signal.
arg1 :
user_data : user data set when the signal handler was connected.

The "tag-changed" signal

```
void        user_function                   (GtkTextTagTable *texttagtable,
                                             GtkTextTag *arg1,
                                             gboolean arg2,
                                             gpointer user_data);
```

texttagtable : the object which received the signal.
arg1 :
arg2 :
user_data : user data set when the signal handler was connected.

The "tag-removed" signal

```
void        user_function                   (GtkTextTagTable *texttagtable,
                                             GtkTextTag *arg1,
                                             gpointer user_data);
```

texttagtable : the object which received the signal.
arg1 :
user_data : user data set when the signal handler was connected.

<< GtkTextTag **GtkTextView >>**

GtkTextView

GtkTextView — Widget that displays a GtkTextBuffer

Synopsis

```
#include <gtk/gtk.h>

            GtkTextView;
enum        GtkTextWindowType;
GtkWidget*  gtk_text_view_new               (void);
GtkWidget*  gtk_text_view_new_with_buffer   (GtkTextBuffer *buffer);
void        gtk_text_view_set_buffer        (GtkTextView *text_view,
                                             GtkTextBuffer *buffer);
GtkTextBuffer* gtk_text_view_get_buffer     (GtkTextView *text_view);
void        gtk_text_view_scroll_to_mark    (GtkTextView *text_view,
                                             GtkTextMark *mark,
                                             gdouble within_margin,
                                             gboolean use_align,
                                             gdouble xalign,
                                             gdouble yalign);
gboolean    gtk_text_view_scroll_to_iter    (GtkTextView *text_view,
                                             GtkTextIter *iter,
                                             gdouble within_margin,
                                             gboolean use_align,
                                             gdouble xalign,
                                             gdouble yalign);
void        gtk_text_view_scroll_mark_onscreen
                                            (GtkTextView *text_view,
                                             GtkTextMark *mark);
gboolean    gtk_text_view_move_mark_onscreen
                                            (GtkTextView *text_view,
                                             GtkTextMark *mark);
gboolean    gtk_text_view_place_cursor_onscreen
                                            (GtkTextView *text_view);
void        gtk_text_view_get_visible_rect  (GtkTextView *text_view,
```

```
                                                GdkRectangle *visible_rect);
void        gtk_text_view_get_iter_location (GtkTextView *text_view,
                                             const GtkTextIter *iter,
                                             GdkRectangle *location);
void        gtk_text_view_get_line_at_y     (GtkTextView *text_view,
                                             GtkTextIter *target_iter,
                                             gint y,
                                             gint *line_top);
void        gtk_text_view_get_line_yrange   (GtkTextView *text_view,
                                             const GtkTextIter *iter,
                                             gint *y,
                                             gint *height);
void        gtk_text_view_get_iter_at_location
                                            (GtkTextView *text_view,
                                             GtkTextIter *iter,
                                             gint x,
                                             gint y);
void        gtk_text_view_buffer_to_window_coords
                                            (GtkTextView *text_view,
                                             GtkTextWindowType win,
                                             gint buffer_x,
                                             gint buffer_y,
                                             gint *window_x,
                                             gint *window_y);
void        gtk_text_view_window_to_buffer_coords
                                            (GtkTextView *text_view,
                                             GtkTextWindowType win,
                                             gint window_x,
                                             gint window_y,
                                             gint *buffer_x,
                                             gint *buffer_y);
GdkWindow*  gtk_text_view_get_window        (GtkTextView *text_view,
                                             GtkTextWindowType win);
GtkTextWindowType gtk_text_view_get_window_type
                                            (GtkTextView *text_view,
                                             GdkWindow *window);
void        gtk_text_view_set_border_window_size
                                            (GtkTextView *text_view,
                                             GtkTextWindowType type,
                                             gint size);
gint        gtk_text_view_get_border_window_size
                                            (GtkTextView *text_view,
                                             GtkTextWindowType type);
```

```
gboolean    gtk_text_view_forward_display_line
                                            (GtkTextView *text_view,
                                             GtkTextIter *iter);
gboolean    gtk_text_view_backward_display_line
                                            (GtkTextView *text_view,
                                             GtkTextIter *iter);
gboolean    gtk_text_view_forward_display_line_end
                                            (GtkTextView *text_view,
                                             GtkTextIter *iter);
gboolean    gtk_text_view_backward_display_line_start
                                            (GtkTextView *text_view,
                                             GtkTextIter *iter);
gboolean    gtk_text_view_starts_display_line
                                            (GtkTextView *text_view,
                                             const GtkTextIter *iter);
gboolean    gtk_text_view_move_visually     (GtkTextView *text_view,
                                             GtkTextIter *iter,
                                             gint count);
void        gtk_text_view_add_child_at_anchor
                                            (GtkTextView *text_view,
                                             GtkWidget *child,
                                             GtkTextChildAnchor *anchor);
            GtkTextChildAnchor;
GtkTextChildAnchor* gtk_text_child_anchor_new
                                            (void);
GList*      gtk_text_child_anchor_get_widgets
                                            (GtkTextChildAnchor *anchor);
gboolean    gtk_text_child_anchor_get_deleted
                                            (GtkTextChildAnchor *anchor);
void        gtk_text_view_add_child_in_window
                                            (GtkTextView *text_view,
                                             GtkWidget *child,
                                             GtkTextWindowType which_window,
                                             gint xpos,
                                             gint ypos);
void        gtk_text_view_move_child        (GtkTextView *text_view,
                                             GtkWidget *child,
                                             gint xpos,
                                             gint ypos);
void        gtk_text_view_set_wrap_mode     (GtkTextView *text_view,
                                             GtkWrapMode wrap_mode);
GtkWrapMode gtk_text_view_get_wrap_mode     (GtkTextView *text_view);
void        gtk_text_view_set_editable      (GtkTextView *text_view,
```

```
                                                gboolean setting);
gboolean    gtk_text_view_get_editable      (GtkTextView *text_view);
void        gtk_text_view_set_cursor_visible
                                            (GtkTextView *text_view,
                                             gboolean setting);
gboolean    gtk_text_view_get_cursor_visible
                                            (GtkTextView *text_view);
void        gtk_text_view_set_overwrite     (GtkTextView *text_view,
                                             gboolean overwrite);
gboolean    gtk_text_view_get_overwrite     (GtkTextView *text_view);
void        gtk_text_view_set_pixels_above_lines
                                            (GtkTextView *text_view,
                                             gint pixels_above_lines);
gint        gtk_text_view_get_pixels_above_lines
                                            (GtkTextView *text_view);
void        gtk_text_view_set_pixels_below_lines
                                            (GtkTextView *text_view,
                                             gint pixels_below_lines);
gint        gtk_text_view_get_pixels_below_lines
                                            (GtkTextView *text_view);
void        gtk_text_view_set_pixels_inside_wrap
                                            (GtkTextView *text_view,
                                             gint pixels_inside_wrap);
gint        gtk_text_view_get_pixels_inside_wrap
                                            (GtkTextView *text_view);
void        gtk_text_view_set_justification (GtkTextView *text_view,
                                             GtkJustification justification);
GtkJustification gtk_text_view_get_justification
                                            (GtkTextView *text_view);
void        gtk_text_view_set_left_margin   (GtkTextView *text_view,
                                             gint left_margin);
gint        gtk_text_view_get_left_margin   (GtkTextView *text_view);
void        gtk_text_view_set_right_margin  (GtkTextView *text_view,
                                             gint right_margin);
gint        gtk_text_view_get_right_margin  (GtkTextView *text_view);
void        gtk_text_view_set_indent        (GtkTextView *text_view,
                                             gint indent);
gint        gtk_text_view_get_indent        (GtkTextView *text_view);
void        gtk_text_view_set_tabs          (GtkTextView *text_view,
                                             PangoTabArray *tabs);
PangoTabArray* gtk_text_view_get_tabs       (GtkTextView *text_view);
void        gtk_text_view_set_accepts_tab   (GtkTextView *text_view,
                                             gboolean accepts_tab);
```

```
gboolean    gtk_text_view_get_accepts_tab   (GtkTextView *text_view);
GtkTextAttributes* gtk_text_view_get_default_attributes
                                            (GtkTextView *text_view);
#define     GTK_TEXT_VIEW_PRIORITY_VALIDATE
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkWidget
               +----GtkContainer
                     +----GtkTextView
```

```
  GObject
   +----GtkTextChildAnchor
```

Implemented Interfaces

GtkTextView implements AtkImplementorIface.

Properties

```
  "accepts-tab"          gboolean             : Read / Write
  "buffer"               GtkTextBuffer        : Read / Write
  "cursor-visible"       gboolean             : Read / Write
  "editable"             gboolean             : Read / Write
  "indent"               gint                 : Read / Write
  "justification"        GtkJustification     : Read / Write
  "left-margin"          gint                 : Read / Write
  "overwrite"            gboolean             : Read / Write
  "pixels-above-lines"   gint                 : Read / Write
  "pixels-below-lines"   gint                 : Read / Write
  "pixels-inside-wrap"   gint                 : Read / Write
  "right-margin"         gint                 : Read / Write
```

```
  "tabs"                PangoTabArray         : Read / Write
  "wrap-mode"           GtkWrapMode           : Read / Write
```

Style Properties

```
  "error-underline-color" GdkColor             : Read
```

Signal Prototypes

```
"backspace" void         user_function      (GtkTextView *textview,
                                             gpointer user_data);
"copy-clipboard"
            void         user_function      (GtkTextView *textview,
                                             gpointer user_data);
"cut-clipboard"
            void         user_function      (GtkTextView *textview,
                                             gpointer user_data);
"delete-from-cursor"
            void         user_function      (GtkTextView *textview,
                                             GtkDeleteType arg1,
                                             gint arg2,
                                             gpointer user_data);
"insert-at-cursor"
            void         user_function      (GtkTextView *textview,
                                             gchar *arg1,
                                             gpointer user_data);
"move-cursor"
            void         user_function      (GtkTextView *widget,
                                             GtkMovementStep step,
                                             gint count,
                                             gboolean extend_selection,
                                             gpointer user_data);
"move-focus"
            void         user_function      (GtkTextView *textview,
                                             GtkDirectionType arg1,
                                             gpointer user_data);
"move-viewport"
            void         user_function      (GtkTextView *textview,
```

```
                                                GtkScrollStep arg1,
                                                gint arg2,
                                                gpointer user_data);
"page-horizontally"
            void        user_function      (GtkTextView *textview,
                                                gint arg1,
                                                gboolean arg2,
                                                gpointer user_data);
"paste-clipboard"
            void        user_function      (GtkTextView *textview,
                                                gpointer user_data);
"populate-popup"
            void        user_function      (GtkTextView *textview,
                                                GtkMenu *arg1,
                                                gpointer user_data);
"select-all"
            void        user_function      (GtkTextView *textview,
                                                gboolean arg1,
                                                gpointer user_data);
"set-anchor"
            void        user_function      (GtkTextView *textview,
                                                gpointer user_data);
"set-scroll-adjustments"
            void        user_function      (GtkTextView *textview,
                                                GtkAdjustment *arg1,
                                                GtkAdjustment *arg2,
                                                gpointer user_data);
"toggle-overwrite"
            void        user_function      (GtkTextView *textview,
                                                gpointer user_data);
```

Description

You may wish to begin by reading the text widget conceptual overview which gives an overview of all the objects and data types related to the text widget and how they work together.

Details

GtkTextView

```
typedef struct _GtkTextView GtkTextView;
```

---

enum GtkTextWindowType

```
typedef enum
{
  GTK_TEXT_WINDOW_PRIVATE,
  GTK_TEXT_WINDOW_WIDGET,
  GTK_TEXT_WINDOW_TEXT,
  GTK_TEXT_WINDOW_LEFT,
  GTK_TEXT_WINDOW_RIGHT,
  GTK_TEXT_WINDOW_TOP,
  GTK_TEXT_WINDOW_BOTTOM
} GtkTextWindowType;
```

---

gtk_text_view_new ()

```
GtkWidget*  gtk_text_view_new               (void);
```

Creates a new GtkTextView. If you don't call `gtk_text_view_set_buffer()` before using the text view, an empty default buffer will be created for you. Get the buffer with `gtk_text_view_get_buffer()`. If you want to specify your own buffer, consider `gtk_text_view_new_with_buffer()`.

Returns : a new GtkTextView

---

gtk_text_view_new_with_buffer ()

```
GtkWidget*  gtk_text_view_new_with_buffer   (GtkTextBuffer *buffer);
```

Creates a new GtkTextView widget displaying the buffer *`buffer`*. One buffer can be shared among many widgets. *`buffer`* may be NULL to create a default buffer, in which case this function is equivalent to `gtk_text_view_new()`. The text view adds its own reference count to the buffer; it does not take over an existing reference.

buffer : a GtkTextBuffer

Returns : a new GtkTextView.

---

gtk_text_view_set_buffer ()

```
void        gtk_text_view_set_buffer        (GtkTextView *text_view,
                                             GtkTextBuffer *buffer);
```

Sets *buffer* as the buffer being displayed by *text_view*. The previous buffer displayed by the text view is unreferenced, and a reference is added to *buffer*. If you owned a reference to *buffer* before passing it to this function, you must remove that reference yourself; GtkTextView will not "adopt" it.

text_view : a GtkTextView

buffer : a GtkTextBuffer

---

gtk_text_view_get_buffer ()

```
GtkTextBuffer* gtk_text_view_get_buffer     (GtkTextView *text_view);
```

Returns the GtkTextBuffer being displayed by this text view. The reference count on the buffer is not incremented; the caller of this function won't own a new reference.

text_view : a GtkTextView

Returns : a GtkTextBuffer

---

gtk_text_view_scroll_to_mark ()

```
void        gtk_text_view_scroll_to_mark    (GtkTextView *text_view,
                                             GtkTextMark *mark,
                                             gdouble within_margin,
                                             gboolean use_align,
                                             gdouble xalign,
                                             gdouble yalign);
```

Scrolls *text_view* so that *mark* is on the screen in the position indicated by *xalign* and *yalign*. An alignment of 0.0 indicates left or top, 1.0 indicates right or bottom, 0.5 means center. If *use_align* is FALSE, the text scrolls the minimal distance to get the mark onscreen, possibly not scrolling at all. The effective screen for purposes of this function is reduced by a margin of size *within_margin*.

text_view :	a GtkTextView
mark :	a GtkTextMark
within_margin :	margin as a [0.0,0.5) fraction of screen size
use_align :	whether to use alignment arguments (if FALSE, just get the mark onscreen)
xalign :	horizontal alignment of mark within visible area.
yalign :	vertical alignment of mark within visible area

---

gtk_text_view_scroll_to_iter ()

```
gboolean    gtk_text_view_scroll_to_iter    (GtkTextView *text_view,
                                             GtkTextIter *iter,
                                             gdouble within_margin,
                                             gboolean use_align,
                                             gdouble xalign,
                                             gdouble yalign);
```

Scrolls *text_view* so that *iter* is on the screen in the position indicated by *xalign* and *yalign*. An alignment of 0.0 indicates left or top, 1.0 indicates right or bottom, 0.5 means center. If *use_align* is FALSE, the text scrolls the minimal distance to get the mark onscreen, possibly not scrolling at all. The effective screen for purposes of this function is reduced by a margin of size *within_margin*. NOTE: This function uses the currently-computed height of the lines in the text buffer. Note that line heights are computed in an idle handler; so this function may not have the desired effect if it's called before the height computations. To avoid oddness, consider using `gtk_text_view_scroll_to_mark()` which saves a point to be scrolled to after line validation.

text_view :	a GtkTextView
iter :	a GtkTextIter
within_margin :	margin as a [0.0,0.5) fraction of screen size
use_align :	whether to use alignment arguments (if FALSE, just get the mark onscreen)
xalign :	horizontal alignment of mark within visible area.
yalign :	vertical alignment of mark within visible area
Returns :	TRUE if scrolling occurred

---

gtk_text_view_scroll_mark_onscreen ()

```
void        gtk_text_view_scroll_mark_onscreen
                                            (GtkTextView *text_view,
                                             GtkTextMark *mark);
```

Scrolls *text_view* the minimum distance such that *mark* is contained within the visible area of the widget.

text_view : a GtkTextView
mark : a mark in the buffer for *text_view*

---

gtk_text_view_move_mark_onscreen ()

```
gboolean    gtk_text_view_move_mark_onscreen
                                            (GtkTextView *text_view,
                                             GtkTextMark *mark);
```

Moves a mark within the buffer so that it's located within the currently-visible text area.

text_view : a GtkTextView
mark : a GtkTextMark
Returns : TRUE if the mark moved (wasn't already onscreen)

---

gtk_text_view_place_cursor_onscreen ()

```
gboolean    gtk_text_view_place_cursor_onscreen
                                            (GtkTextView *text_view);
```

Moves the cursor to the currently visible region of the buffer, it it isn't there already.

text_view : a GtkTextView
Returns : TRUE if the cursor had to be moved.

---

gtk_text_view_get_visible_rect ()

```
void        gtk_text_view_get_visible_rect  (GtkTextView *text_view,
                                             GdkRectangle *visible_rect);
```

Fills *visible_rect* with the currently-visible region of the buffer, in buffer coordinates. Convert to window coordinates with gtk_text_view_buffer_to_window_coords().

text_view: a GtkTextView
visible_rect: rectangle to fill

---

gtk_text_view_get_iter_location ()

```
void        gtk_text_view_get_iter_location (GtkTextView *text_view,
                                             const GtkTextIter *iter,
                                             GdkRectangle *location);
```

Gets a rectangle which roughly contains the character at *iter*. The rectangle position is in buffer coordinates; use gtk_text_view_buffer_to_window_coords() to convert these coordinates to coordinates for one of the windows in the text view.

text_view: a GtkTextView
iter: a GtkTextIter
location: bounds of the character at *iter*

---

gtk_text_view_get_line_at_y ()

```
void        gtk_text_view_get_line_at_y     (GtkTextView *text_view,
                                             GtkTextIter *target_iter,
                                             gint y,
                                             gint *line_top);
```

Gets the GtkTextIter at the start of the line containing the coordinate *y*. *y* is in buffer coordinates, convert from window coordinates with gtk_text_view_window_to_buffer_coords(). If non-NULL, *line_top* will be filled with the coordinate of the top edge of the line.

text_view: a GtkTextView

target_iter : a GtkTextIter
y : a y coordinate
line_top : return location for top coordinate of the line

---

gtk_text_view_get_line_yrange ()

```
void        gtk_text_view_get_line_yrange   (GtkTextView *text_view,
                                             const GtkTextIter *iter,
                                             gint *y,
                                             gint *height);
```

Gets the y coordinate of the top of the line containing *iter*, and the height of the line. The coordinate is a buffer coordinate; convert to window coordinates with gtk_text_view_buffer_to_window_coords().

text_view : a GtkTextView
iter : a GtkTextIter
y : return location for a y coordinate
height : return location for a height

---

gtk_text_view_get_iter_at_location ()

```
void        gtk_text_view_get_iter_at_location
                                            (GtkTextView *text_view,
                                             GtkTextIter *iter,
                                             gint x,
                                             gint y);
```

Retrieves the iterator at buffer coordinates *x* and *y*. Buffer coordinates are coordinates for the entire buffer, not just the currently-displayed portion. If you have coordinates from an event, you have to convert those to buffer coordinates with gtk_text_view_window_to_buffer_coords().

text_view : a GtkTextView
iter : a GtkTextIter
x : x position, in buffer coordinates
y : y position, in buffer coordinates

---

gtk_text_view_buffer_to_window_coords ()

```
void        gtk_text_view_buffer_to_window_coords
                                            (GtkTextView *text_view,
                                             GtkTextWindowType win,
                                             gint buffer_x,
                                             gint buffer_y,
                                             gint *window_x,
                                             gint *window_y);
```

Converts coordinate (*buffer_x*, *buffer_y*) to coordinates for the window *win*, and stores the result in (*window_x*, *window_y*).

Note that you can't convert coordinates for a nonexisting window (see gtk_text_view_set_border_window_size()).

text_view : a GtkTextView

win : a GtkTextWindowType except GTK_TEXT_WINDOW_PRIVATE

buffer_x : buffer x coordinate

buffer_y : buffer y coordinate

window_x : window x coordinate return location

window_y : window y coordinate return location

---

gtk_text_view_window_to_buffer_coords ()

```
void        gtk_text_view_window_to_buffer_coords
                                            (GtkTextView *text_view,
                                             GtkTextWindowType win,
                                             gint window_x,
                                             gint window_y,
                                             gint *buffer_x,
                                             gint *buffer_y);
```

Converts coordinates on the window identified by *win* to buffer coordinates, storing the result in (*buffer_x*,*buffer_y*).

Note that you can't convert coordinates for a nonexisting window (see gtk_text_view_set_border_window_size()).

text_view : a GtkTextView
win : a GtkTextWindowType except GTK_TEXT_WINDOW_PRIVATE
window_x : window x coordinate
window_y : window y coordinate
buffer_x : buffer x coordinate return location
buffer_y : buffer y coordinate return location

---

gtk_text_view_get_window ()

```
GdkWindow*  gtk_text_view_get_window        (GtkTextView *text_view,
                                             GtkTextWindowType win);
```

Retrieves the GdkWindow corresponding to an area of the text view; possible windows include the overall widget window, child windows on the left, right, top, bottom, and the window that displays the text buffer. Windows are NULL and nonexistent if their width or height is 0, and are nonexistent before the widget has been realized.

text_view : a GtkTextView
win : window to get
Returns : a GdkWindow, or NULL

---

gtk_text_view_get_window_type ()

```
GtkTextWindowType gtk_text_view_get_window_type
                                            (GtkTextView *text_view,
                                             GdkWindow *window);
```

Usually used to find out which window an event corresponds to. If you connect to an event signal on *text_view*, this function should be called on event->window to see which window it was.

text_view : a GtkTextView
window : a window type
Returns : the window type.

---

gtk_text_view_set_border_window_size ()

```
void        gtk_text_view_set_border_window_size
                                            (GtkTextView *text_view,
                                             GtkTextWindowType type,
                                             gint size);
```

Sets the width of GTK_TEXT_WINDOW_LEFT or GTK_TEXT_WINDOW_RIGHT, or the height of GTK_TEXT_WINDOW_TOP or GTK_TEXT_WINDOW_BOTTOM. Automatically destroys the corresponding window if the size is set to 0, and creates the window if the size is set to non-zero. This function can only be used for the "border windows," it doesn't work with GTK_TEXT_WINDOW_WIDGET, GTK_TEXT_WINDOW_TEXT, or GTK_TEXT_WINDOW_PRIVATE.

text_view : a GtkTextView
type : window to affect
size : width or height of the window

---

gtk_text_view_get_border_window_size ()

```
gint        gtk_text_view_get_border_window_size
                                            (GtkTextView *text_view,
                                             GtkTextWindowType type);
```

Gets the width of the specified border window. See gtk_text_view_set_border_window_size().

text_view : a GtkTextView
type : window to return size from
Returns : width of window

---

gtk_text_view_forward_display_line ()

```
gboolean    gtk_text_view_forward_display_line
                                            (GtkTextView *text_view,
                                             GtkTextIter *iter);
```

Moves the given *iter* forward by one display (wrapped) line. A display line is different from a paragraph.

Paragraphs are separated by newlines or other paragraph separator characters. Display lines are created by line-wrapping a paragraph. If wrapping is turned off, display lines and paragraphs will be the same. Display lines are divided differently for each view, since they depend on the view's width; paragraphs are the same in all views, since they depend on the contents of the GtkTextBuffer.

text_view : a GtkTextView
iter : a GtkTextIter
Returns : TRUE if *iter* was moved and is not on the end iterator

---

gtk_text_view_backward_display_line ()

```
gboolean    gtk_text_view_backward_display_line
                                            (GtkTextView *text_view,
                                             GtkTextIter *iter);
```

Moves the given *iter* backward by one display (wrapped) line. A display line is different from a paragraph. Paragraphs are separated by newlines or other paragraph separator characters. Display lines are created by line-wrapping a paragraph. If wrapping is turned off, display lines and paragraphs will be the same. Display lines are divided differently for each view, since they depend on the view's width; paragraphs are the same in all views, since they depend on the contents of the GtkTextBuffer.

text_view : a GtkTextView
iter : a GtkTextIter
Returns : TRUE if *iter* was moved and is not on the end iterator

---

gtk_text_view_forward_display_line_end ()

```
gboolean    gtk_text_view_forward_display_line_end
                                            (GtkTextView *text_view,
                                             GtkTextIter *iter);
```

Moves the given *iter* forward to the next display line end. A display line is different from a paragraph. Paragraphs are separated by newlines or other paragraph separator characters. Display lines are created by line-wrapping a paragraph. If wrapping is turned off, display lines and paragraphs will be the same. Display lines are divided differently for each view, since they depend on the view's width; paragraphs are the same in all views, since they depend on the contents of the GtkTextBuffer.

text_view : a GtkTextView

iter : a GtkTextIter

Returns : TRUE if *iter* was moved and is not on the end iterator

---

gtk_text_view_backward_display_line_start ()

```
gboolean    gtk_text_view_backward_display_line_start
                                            (GtkTextView *text_view,
                                             GtkTextIter *iter);
```

Moves the given *iter* backward to the next display line start. A display line is different from a paragraph. Paragraphs are separated by newlines or other paragraph separator characters. Display lines are created by line-wrapping a paragraph. If wrapping is turned off, display lines and paragraphs will be the same. Display lines are divided differently for each view, since they depend on the view's width; paragraphs are the same in all views, since they depend on the contents of the GtkTextBuffer.

text_view : a GtkTextView

iter : a GtkTextIter

Returns : TRUE if *iter* was moved and is not on the end iterator

---

gtk_text_view_starts_display_line ()

```
gboolean    gtk_text_view_starts_display_line
                                            (GtkTextView *text_view,
                                             const GtkTextIter *iter);
```

Determines whether *iter* is at the start of a display line. See gtk_text_view_forward_display_line() for an explanation of display lines vs. paragraphs.

text_view : a GtkTextView

iter : a GtkTextIter

Returns : TRUE if *iter* begins a wrapped line

---

gtk_text_view_move_visually ()

```
gboolean    gtk_text_view_move_visually     (GtkTextView *text_view,
                                             GtkTextIter *iter,
                                             gint count);
```

Move the iterator a given number of characters visually, treating it as the strong cursor position. If *count* is positive, then the new strong cursor position will be *count* positions to the right of the old cursor position. If *count* is negative then the new strong cursor position will be *count* positions to the left of the old cursor position.

In the presence of bidirection text, the correspondence between logical and visual order will depend on the direction of the current run, and there may be jumps when the cursor is moved off of the end of a run.

text_view : a GtkTextView
iter : a GtkTextIter
count : number of characters to move (negative moves left, positive moves right)
Returns : TRUE if *iter* moved and is not on the end iterator

---

gtk_text_view_add_child_at_anchor ()

```
void        gtk_text_view_add_child_at_anchor
                                            (GtkTextView *text_view,
                                             GtkWidget *child,
                                             GtkTextChildAnchor *anchor);
```

Adds a child widget in the text buffer, at the given *anchor*.

text_view : a GtkTextView
child : a GtkWidget
anchor : a GtkTextChildAnchor in the GtkTextBuffer for *text_view*

GtkTextChildAnchor

```
typedef struct _GtkTextChildAnchor GtkTextChildAnchor;
```

A GtkTextChildAnchor is a spot in the buffer where child widgets can be "anchored" (inserted inline, as if they were characters). The anchor can have multiple widgets anchored, to allow for multiple views.

gtk_text_child_anchor_new ()

```
GtkTextChildAnchor* gtk_text_child_anchor_new
                                                    (void);
```

Creates a new GtkTextChildAnchor. Usually you would then insert it into a GtkTextBuffer with `gtk_text_buffer_insert_child_anchor()`. To perform the creation and insertion in one step, use the convenience function `gtk_text_buffer_create_child_anchor()`.

Returns : a new GtkTextChildAnchor

gtk_text_child_anchor_get_widgets ()

```
GList*      gtk_text_child_anchor_get_widgets
                                                    (GtkTextChildAnchor *anchor);
```

Gets a list of all widgets anchored at this child anchor. The returned list should be freed with `g_list_free()`.

anchor : a GtkTextChildAnchor

Returns : list of widgets anchored at *anchor*

gtk_text_child_anchor_get_deleted ()

```
gboolean    gtk_text_child_anchor_get_deleted
                                                    (GtkTextChildAnchor *anchor);
```

Determines whether a child anchor has been deleted from the buffer. Keep in mind that the child anchor will be unreferenced when removed from the buffer, so you need to hold your own reference (with `g_object_ref()`) if you plan to use this function — otherwise all deleted child anchors will also be finalized.

anchor : a GtkTextChildAnchor

Returns : TRUE if the child anchor has been deleted from its buffer

gtk_text_view_add_child_in_window ()

```
void        gtk_text_view_add_child_in_window
                                            (GtkTextView *text_view,
                                             GtkWidget *child,
                                             GtkTextWindowType which_window,
                                             gint xpos,
                                             gint ypos);
```

Adds a child at fixed coordinates in one of the text widget's windows. The window must have nonzero size (see gtk_text_view_set_border_window_size()). Note that the child coordinates are given relative to the GdkWindow in question, and that these coordinates have no sane relationship to scrolling. When placing a child in GTK_TEXT_WINDOW_WIDGET, scrolling is irrelevant, the child floats above all scrollable areas. But when placing a child in one of the scrollable windows (border windows or text window), you'll need to compute the child's correct position in buffer coordinates any time scrolling occurs or buffer changes occur, and then call gtk_text_view_move_child() to update the child's position. Unfortunately there's no good way to detect that scrolling has occurred, using the current API; a possible hack would be to update all child positions when the scroll adjustments change or the text buffer changes. See bug 64518 on bugzilla.gnome.org for status of fixing this issue.

text_view :	a GtkTextView
child :	a GtkWidget
which_window :	which window the child should appear in
xpos :	X position of child in window coordinates
ypos :	Y position of child in window coordinates

---

gtk_text_view_move_child ()

```
void        gtk_text_view_move_child        (GtkTextView *text_view,
                                             GtkWidget *child,
                                             gint xpos,
                                             gint ypos);
```

Updates the position of a child, as for gtk_text_view_add_child_in_window().

text_view :	a GtkTextView
child :	child widget already added to the text view
xpos :	new X position in window coordinates

ypos : new Y position in window coordinates

---

gtk_text_view_set_wrap_mode ()

```
void        gtk_text_view_set_wrap_mode     (GtkTextView *text_view,
                                             GtkWrapMode wrap_mode);
```

Sets the line wrapping for the view.

text_view : a GtkTextView
wrap_mode : a GtkWrapMode

---

gtk_text_view_get_wrap_mode ()

```
GtkWrapMode gtk_text_view_get_wrap_mode     (GtkTextView *text_view);
```

Gets the line wrapping for the view.

text_view : a GtkTextView
Returns : the line wrap setting

---

gtk_text_view_set_editable ()

```
void        gtk_text_view_set_editable      (GtkTextView *text_view,
                                             gboolean setting);
```

Sets the default editability of the GtkTextView. You can override this default setting with tags in the buffer, using the "editable" attribute of tags.

text_view : a GtkTextView
setting : whether it's editable

---

gtk_text_view_get_editable ()

```
gboolean    gtk_text_view_get_editable      (GtkTextView *text_view);
```

Returns the default editability of the GtkTextView. Tags in the buffer may override this setting for some ranges of text.

text_view : a GtkTextView

Returns : whether text is editable by default

---

gtk_text_view_set_cursor_visible ()

```
void        gtk_text_view_set_cursor_visible
                                            (GtkTextView *text_view,
                                             gboolean setting);
```

Toggles whether the insertion point is displayed. A buffer with no editable text probably shouldn't have a visible cursor, so you may want to turn the cursor off.

text_view : a GtkTextView

setting : whether to show the insertion cursor

---

gtk_text_view_get_cursor_visible ()

```
gboolean    gtk_text_view_get_cursor_visible
                                            (GtkTextView *text_view);
```

Find out whether the cursor is being displayed.

text_view : a GtkTextView

Returns : whether the insertion mark is visible

---

gtk_text_view_set_overwrite ()

```
void        gtk_text_view_set_overwrite     (GtkTextView *text_view,
                                             gboolean overwrite);
```

Changes the GtkTextView overwrite mode.

text_view : a GtkTextView
overwrite : TRUE to turn on overwrite mode, FALSE to turn it off

Since 2.4

---

gtk_text_view_get_overwrite ()

```
gboolean    gtk_text_view_get_overwrite     (GtkTextView *text_view);
```

Returns whether the GtkTextView is in overwrite mode or not.

text_view : a GtkTextView
Returns : whether *text_view* is in overwrite mode or not.

Since 2.4

---

gtk_text_view_set_pixels_above_lines ()

```
void        gtk_text_view_set_pixels_above_lines
                                            (GtkTextView *text_view,
                                             gint pixels_above_lines);
```

Sets the default number of blank pixels above paragraphs in *text_view*. Tags in the buffer for *text_view* may override the defaults.

text_view : a GtkTextView
pixels_above_lines : pixels above paragraphs

---

gtk_text_view_get_pixels_above_lines ()

```
gint        gtk_text_view_get_pixels_above_lines
                                            (GtkTextView *text_view);
```

Gets the default number of pixels to put above paragraphs.

text_view : a GtkTextView

Returns : default number of pixels above paragraphs

---

gtk_text_view_set_pixels_below_lines ()

```
void        gtk_text_view_set_pixels_below_lines
                                            (GtkTextView *text_view,
                                             gint pixels_below_lines);
```

Sets the default number of pixels of blank space to put below paragraphs in *text_view*. May be overridden by tags applied to *text_view*'s buffer.

text_view : a GtkTextView

pixels_below_lines : pixels below paragraphs

---

gtk_text_view_get_pixels_below_lines ()

```
gint        gtk_text_view_get_pixels_below_lines
                                            (GtkTextView *text_view);
```

Gets the value set by gtk_text_view_set_pixels_below_lines().

text_view : a GtkTextView

Returns : default number of blank pixels below paragraphs

---

gtk_text_view_set_pixels_inside_wrap ()

```
void        gtk_text_view_set_pixels_inside_wrap
                                            (GtkTextView *text_view,
                                             gint pixels_inside_wrap);
```

Sets the default number of pixels of blank space to leave between display/wrapped lines within a paragraph. May be overridden by tags in *text_view*'s buffer.

text_view : a GtkTextView
pixels_inside_wrap : default number of pixels between wrapped lines

---

gtk_text_view_get_pixels_inside_wrap ()

```
gint        gtk_text_view_get_pixels_inside_wrap
                                            (GtkTextView *text_view);
```

Gets the value set by gtk_text_view_set_pixels_inside_wrap().

text_view : a GtkTextView
Returns : default number of pixels of blank space between wrapped lines

---

gtk_text_view_set_justification ()

```
void        gtk_text_view_set_justification (GtkTextView *text_view,
                                             GtkJustification justification);
```

Sets the default justification of text in *text_view*. Tags in the view's buffer may override the default.

text_view : a GtkTextView
justification : justification

---

gtk_text_view_get_justification ()

```
GtkJustification gtk_text_view_get_justification
```

```
                                                (GtkTextView *text_view);
```

Gets the default justification of paragraphs in *text_view*. Tags in the buffer may override the default.

text_view : a GtkTextView
Returns : default justification

---

gtk_text_view_set_left_margin ()

```
void        gtk_text_view_set_left_margin   (GtkTextView *text_view,
                                             gint left_margin);
```

Sets the default left margin for text in *text_view*. Tags in the buffer may override the default.

text_view : a GtkTextView
left_margin : left margin in pixels

---

gtk_text_view_get_left_margin ()

```
gint        gtk_text_view_get_left_margin   (GtkTextView *text_view);
```

Gets the default left margin size of paragraphs in the *text_view*. Tags in the buffer may override the default.

text_view : a GtkTextView
Returns : left margin in pixels

---

gtk_text_view_set_right_margin ()

```
void        gtk_text_view_set_right_margin  (GtkTextView *text_view,
                                             gint right_margin);
```

Sets the default right margin for text in the text view. Tags in the buffer may override the default.

text_view : a GtkTextView
right_margin : right margin in pixels

---

gtk_text_view_get_right_margin ()

```
gint        gtk_text_view_get_right_margin  (GtkTextView *text_view);
```

Gets the default right margin for text in *text_view*. Tags in the buffer may override the default.

text_view : a GtkTextView
Returns : right margin in pixels

---

gtk_text_view_set_indent ()

```
void        gtk_text_view_set_indent        (GtkTextView *text_view,
                                             gint indent);
```

Sets the default indentation for paragraphs in *text_view*. Tags in the buffer may override the default.

text_view : a GtkTextView
indent : indentation in pixels

---

gtk_text_view_get_indent ()

```
gint        gtk_text_view_get_indent        (GtkTextView *text_view);
```

Gets the default indentation of paragraphs in *text_view*. Tags in the view's buffer may override the default. The indentation may be negative.

text_view : a GtkTextView
Returns : number of pixels of indentation

---

gtk_text_view_set_tabs ()

```
void        gtk_text_view_set_tabs          (GtkTextView *text_view,
                                             PangoTabArray *tabs);
```

Sets the default tab stops for paragraphs in *text_view*. Tags in the buffer may override the default.

text_view : a GtkTextView

tabs : tabs as a PangoTabArray

---

gtk_text_view_get_tabs ()

```
PangoTabArray* gtk_text_view_get_tabs       (GtkTextView *text_view);
```

Gets the default tabs for *text_view*. Tags in the buffer may override the defaults. The returned array will be NULL if "standard" (8-space) tabs are used. Free the return value with pango_tab_array_free().

text_view : a GtkTextView

Returns : copy of default tab array, or NULL if "standard" tabs are used; must be freed with pango_tab_array_free().

---

gtk_text_view_set_accepts_tab ()

```
void        gtk_text_view_set_accepts_tab   (GtkTextView *text_view,
                                             gboolean accepts_tab);
```

Sets the behavior of the text widget when the Tab key is pressed. If *accepts_tab* is TRUE a tab character is inserted. If *accepts_tab* is FALSE the keyboard focus is moved to the next widget in the focus chain.

text_view : A GtkTextView

accepts_tab : TRUE if pressing the Tab key should insert a tab character, FALSE, if pressing the Tab key should move the keyboard focus.

Since 2.4

gtk_text_view_get_accepts_tab ()

```
gboolean    gtk_text_view_get_accepts_tab   (GtkTextView *text_view);
```

Returns whether pressing the Tab key inserts a tab characters. `gtk_text_view_set_accepts_tab()`.

text_view : A GtkTextView

Returns : TRUE if pressing the Tab key inserts a tab character, FALSE if pressing the Tab key moves the keyboard focus.

Since 2.4

gtk_text_view_get_default_attributes ()

```
GtkTextAttributes* gtk_text_view_get_default_attributes
                                                (GtkTextView *text_view);
```

Obtains a copy of the default text attributes. These are the attributes used for text unless a tag overrides them. You'd typically pass the default attributes in to `gtk_text_iter_get_attributes()` in order to get the attributes in effect at a given text position.

The return value is a copy owned by the caller of this function, and should be freed.

text_view : a GtkTextView

Returns : a new GtkTextAttributes

GTK_TEXT_VIEW_PRIORITY_VALIDATE

```
#define GTK_TEXT_VIEW_PRIORITY_VALIDATE (GDK_PRIORITY_REDRAW + 5)
```

The priority at which the text view validates onscreen lines in an idle job in the background.

Properties

The "accepts-tab" property

"accepts-tab"	gboolean	: Read / Write

Whether Tab will result in a tab character being entered.

Default value: TRUE

The "buffer" property

"buffer"	GtkTextBuffer	: Read / Write

The buffer which is displayed.

The "cursor-visible" property

"cursor-visible"	gboolean	: Read / Write

If the insertion cursor is shown.

Default value: TRUE

The "editable" property

"editable"	gboolean	: Read / Write

Whether the text can be modified by the user.

Default value: TRUE

The "indent" property

"indent"	gint	: Read / Write

Amount to indent the paragraph, in pixels.

Allowed values: >= 0

Default value: 0

The "justification" property

"justification"	GtkJustification	: Read / Write

Left, right, or center justification.

Default value: GTK_JUSTIFY_LEFT

The "left-margin" property

"left-margin"	gint	: Read / Write

Width of the left margin in pixels.

Allowed values: >= 0

Default value: 0

The "overwrite" property

"overwrite"	gboolean	: Read / Write

Whether entered text overwrites existing contents.

Default value: FALSE

The "pixels-above-lines" property

```
"pixels-above-lines"   gint                : Read / Write
```

Pixels of blank space above paragraphs.

Allowed values: >= 0

Default value: 0

The "pixels-below-lines" property

```
"pixels-below-lines"   gint                : Read / Write
```

Pixels of blank space below paragraphs.

Allowed values: >= 0

Default value: 0

The "pixels-inside-wrap" property

```
"pixels-inside-wrap"   gint                : Read / Write
```

Pixels of blank space between wrapped lines in a paragraph.

Allowed values: >= 0

Default value: 0

The "right-margin" property

```
"right-margin"         gint                  : Read / Write
```

Width of the right margin in pixels.

Allowed values: >= 0

Default value: 0

The "tabs" property

```
"tabs"                 PangoTabArray         : Read / Write
```

Custom tabs for this text.

The "wrap-mode" property

```
"wrap-mode"            GtkWrapMode           : Read / Write
```

Whether to wrap lines never, at word boundaries, or at character boundaries.

Default value: GTK_WRAP_NONE

Style Properties

The "error-underline-color" style property

```
"error-underline-color" GdkColor              : Read
```

Color with which to draw error-indication underlines.

Signals

The "backspace" signal

```
void        user_function                   (GtkTextView *textview,
                                             gpointer user_data);
```

textview : the object which received the signal.
user_data : user data set when the signal handler was connected.

The "copy-clipboard" signal

```
void        user_function                   (GtkTextView *textview,
                                             gpointer user_data);
```

textview : the object which received the signal.
user_data : user data set when the signal handler was connected.

The "cut-clipboard" signal

```
void        user_function                   (GtkTextView *textview,
                                             gpointer user_data);
```

textview : the object which received the signal.
user_data : user data set when the signal handler was connected.

The "delete-from-cursor" signal

```
void        user_function                   (GtkTextView *textview,
                                             GtkDeleteType arg1,
                                             gint arg2,
                                             gpointer user_data);
```

textview : the object which received the signal.
arg1 :
arg2 :
user_data : user data set when the signal handler was connected.

The "insert-at-cursor" signal

```
void        user_function                   (GtkTextView *textview,
                                             gchar *arg1,
                                             gpointer user_data);
```

textview : the object which received the signal.
arg1 :
user_data : user data set when the signal handler was connected.

The "move-cursor" signal

```
void        user_function                   (GtkTextView *widget,
                                             GtkMovementStep step,
                                             gint count,
                                             gboolean extend_selection,
                                             gpointer user_data);
```

The ::move-cursor signal is a keybinding signal which gets emitted when the user initiates a cursor movement.

Applications should not connect to it, but may emit it with `g_signal_emit_by_name()` if they need to control scrolling programmatically.

widget : the object which received the signal
step : the granularity of the move, as a GtkMovementStep
count : the number of *step* units to move
extend_selection : TRUE if the move should extend the selection
user_data : user data set when the signal handler was connected.

The "move-focus" signal

```
void        user_function                   (GtkTextView *textview,
                                             GtkDirectionType arg1,
                                             gpointer user_data);
```

textview : the object which received the signal.
arg1 :
user_data : user data set when the signal handler was connected.

The "move-viewport" signal

```
void        user_function                   (GtkTextView *textview,
                                             GtkScrollStep arg1,
                                             gint arg2,
                                             gpointer user_data);
```

textview : the object which received the signal.
arg1 :
arg2 :
user_data : user data set when the signal handler was connected.

The "page-horizontally" signal

```
void        user_function                   (GtkTextView *textview,
                                             gint arg1,
                                             gboolean arg2,
                                             gpointer user_data);
```

textview : the object which received the signal.
arg1 :
arg2 :
user_data : user data set when the signal handler was connected.

The "paste-clipboard" signal

```
void        user_function                   (GtkTextView *textview,
                                             gpointer user_data);
```

textview : the object which received the signal.
user_data : user data set when the signal handler was connected.

The "populate-popup" signal

```
void        user_function                   (GtkTextView *textview,
                                             GtkMenu *arg1,
                                             gpointer user_data);
```

textview : the object which received the signal.
arg1 :
user_data : user data set when the signal handler was connected.

The "select-all" signal

```
void        user_function                   (GtkTextView *textview,
                                             gboolean arg1,
                                             gpointer user_data);
```

textview : the object which received the signal.
arg1 :
user_data : user data set when the signal handler was connected.

The "set-anchor" signal

```
void        user_function                   (GtkTextView *textview,
                                             gpointer user_data);
```

textview : the object which received the signal.
user_data : user data set when the signal handler was connected.

The "set-scroll-adjustments" signal

```
void        user_function                   (GtkTextView *textview,
                                             GtkAdjustment *arg1,
                                             GtkAdjustment *arg2,
                                             gpointer user_data);
```

textview : the object which received the signal.
arg1 :
arg2 :
user_data : user data set when the signal handler was connected.

The "toggle-overwrite" signal

```
void        user_function                   (GtkTextView *textview,
                                             gpointer user_data);
```

textview : the object which received the signal.
user_data : user data set when the signal handler was connected.

See Also

GtkTextBuffer, GtkTextIter

<< GtkTextTagTable **Tree, List and Icon Grid Widgets >>**

Tree, List and Icon Grid Widgets

Tree and List Widget Overview - Overview of GtkTreeModel, GtkTreeView, and other associated widgets
GtkTreeModel - The tree interface used by GtkTreeView
GtkTreeSelection - The selection object for GtkTreeView
GtkTreeViewColumn - A visible column in a GtkTreeView widget
GtkTreeView - A widget for displaying both trees and lists
GtkTreeView drag-and-drop - Interfaces for drag-and-drop support in GtkTreeView
GtkCellView - A widget displaying a single row of a GtkTreeModel
GtkIconView - A widget which displays a list of icons in a grid
GtkTreeSortable - The interface for sortable models used by GtkTreeView
GtkTreeModelSort - A GtkTreeModel which makes an underlying tree model sortable
GtkTreeModelFilter - A GtkTreeModel which hides parts of an underlying tree model
GtkCellLayout - An interface for packing cells
GtkCellRenderer - An object for rendering a single cell on a GdkDrawable
GtkCellEditable - Interface for widgets which can are used for editing cells
GtkCellRendererCombo -
GtkCellRendererPixbuf - Renders a pixbuf in a cell
GtkCellRendererProgress - Renders numbers as progress bars
GtkCellRendererText - Renders text in a cell
GtkCellRendererToggle - Renders a toggle button in a cell
GtkListStore - A list-like data structure that can be used with the GtkTreeView
GtkTreeStore - A tree-like data structure that can be used with the GtkTreeView

Tree and List Widget Overview

Tree and List Widget Overview — Overview of GtkTreeModel, GtkTreeView, and other associated widgets

Overview

To create a tree or list in GTK+, use the GtkTreeModel interface in conjunction with the GtkTreeView widget. This widget is designed around a *Model/View/Controller* design and consists of four major parts:

The tree view widget (GtkTreeView)
The view column (GtkTreeViewColumn)
The cell renderers (GtkCellRenderer etc.)
The model interface (GtkTreeModel)

The *View* is composed of the first three objects, while the last is the *Model*. One of the prime benefits of the MVC design is that multiple views can be created of a single model. For example, a model mapping the file system could be created for a file manager. Many views could be created to display various parts of the file system, but only one copy need be kept in memory.

The purpose of the cell renderers is to provide extensibility to the widget and to allow multiple ways of rendering the same type of data. For example, consider how to render a boolean variable. Should it render it as a string of "True" or "False", "On" or "Off", or should it be rendered as a checkbox?

Creating a model

GTK+ provides two simple models that can be used: the GtkListStore and the GtkTreeStore. GtkListStore is used to model list widgets, while the GtkTreeStore models trees. It is possible to develop a new type of model, but the existing models should be satisfactory for all but the most specialized of situations. Creating the model is quite simple:

```
GtkListStore *store = gtk_list_store_new (2, G_TYPE_STRING, G_TYPE_BOOLEAN);
```

This creates a list store with two columns: a string column and a boolean column. Typically the 2 is never passed directly like that; usually an enum is created wherein the different columns are enumerated, followed by a token that represents the total number of columns. The next example will illustrate this, only using a tree store instead of a list store. Creating a tree store operates almost exactly the same.

```
enum
{
   TITLE_COLUMN,
   AUTHOR_COLUMN,
   CHECKED_COLUMN,
   N_COLUMNS
```

```
};

GtkTreeStore *store = gtk_tree_store_new (N_COLUMNS,       /* Total number of columns
*/
                                          G_TYPE_STRING,   /* Book title
*/
                                          G_TYPE_STRING,   /* Author
*/
                                          G_TYPE_BOOLEAN); /* Is checked out?
*/
```

Adding data to the model is done using gtk_tree_store_set() or gtk_list_store_set(), depending upon which sort of model was created. To do this, a GtkTreeIter must be acquired. The iterator points to the location where data will be added.

Once an iterator has been acquired, gtk_tree_store_set() is used to apply data to the part of the model that the iterator points to. Consider the following example:

```
GtkTreeIter   iter;

gtk_tree_store_append (store, &iter, NULL);  /* Acquire an iterator */

gtk_tree_store_set (store, &iter,
                    TITLE_COLUMN, "The Principle of Reason",
                    AUTHOR_COLUMN, "Martin Heidegger",
                    CHECKED_COLUMN, FALSE,
                    -1);
```

Notice that the last argument is -1. This is always done because this is a variable-argument function and it needs to know when to stop processing arguments. It can be used to set the data in any or all columns in a given row.

The third argument to gtk_tree_store_append() is the parent iterator. It is used to add a row to a GtkTreeStore as a child of an existing row. This means that the new row will only be visible when its parent is visible and in its expanded state. Consider the following example:

```
GtkTreeIter iter1;  /* Parent iter */
GtkTreeIter iter2;  /* Child iter  */

gtk_tree_store_append (store, &iter1, NULL);  /* Acquire a top-level iterator */
gtk_tree_store_set (store, &iter1,
                    TITLE_COLUMN, "The Art of Computer Programming",
                    AUTHOR_COLUMN, "Donald E. Knuth",
                    CHECKED_COLUMN, FALSE,
                    -1);

gtk_tree_store_append (store, &iter2, &iter1);  /* Acquire a child iterator */
gtk_tree_store_set (store, &iter2,
                    TITLE_COLUMN, "Volume 1: Fundamental Algorithms",
                    -1);

gtk_tree_store_append (store, &iter2, &iter1);
```

```
gtk_tree_store_set (store, &iter2,
                    TITLE_COLUMN, "Volume 2: Seminumerical Algorithms",
                    -1);

gtk_tree_store_append (store, &iter2, &iter1);
gtk_tree_store_set (store, &iter2,
                    TITLE_COLUMN, "Volume 3: Sorting and Searching",
                    -1);
```

Creating the view component

While there are several different models to choose from, there is only one view widget to deal with. It works with either the list or the tree store. Setting up a GtkTreeView is not a difficult matter. It needs a GtkTreeModel to know where to retrieve its data from.

```
GtkWidget *tree;

tree = gtk_tree_view_new_with_model (GTK_TREE_MODEL (store));
```

Columns and cell renderers

Once the GtkTreeView widget has a model, it will need to know how to display the model. It does this with columns and cell renderers.

Cell renderers are used to draw the data in the tree model in a way. There are a number of cell renderers that come with GTK+ 2.x, including the GtkCellRendererText, GtkCellRendererPixbuf and the GtkCellRendererToggle. It is relatively easy to write a custom renderer.

A GtkTreeViewColumn is the object that GtkTreeView uses to organize the vertical columns in the tree view. It needs to know the name of the column to label for the user, what type of cell renderer to use, and which piece of data to retrieve from the model for a given row.

```
GtkCellRenderer *renderer;
GtkTreeViewColumn *column;

renderer = gtk_cell_renderer_text_new ();
column = gtk_tree_view_column_new_with_attributes ("Author",
                                                   renderer,
                                                   "text", AUTHOR_COLUMN,
                                                   NULL);
gtk_tree_view_append_column (GTK_TREE_VIEW (tree), column);
```

At this point, all the steps in creating a displayable tree have been covered. The model is created, data is stored in it, a tree view is created and columns are added to it.

Selection handling

Most applications will need to not only deal with displaying data, but also receiving input events from users. To do this, simply get a reference to a selection object and connect to the "changed" signal.

```
/* Prototype for selection handler callback */
static void tree_selection_changed_cb (GtkTreeSelection *selection, gpointer data);

/* Setup the selection handler */
GtkTreeSelection *select;

select = gtk_tree_view_get_selection (GTK_TREE_VIEW (tree));
gtk_tree_selection_set_mode (select, GTK_SELECTION_SINGLE);
g_signal_connect (G_OBJECT (select), "changed",
                  G_CALLBACK (tree_selection_changed_cb),
                  NULL);
```

Then to retrieve data for the row selected:

```
static void
tree_selection_changed_cb (GtkTreeSelection *selection, gpointer data)
{
        GtkTreeIter iter;
        GtkTreeModel *model;
        gchar *author;

        if (gtk_tree_selection_get_selected (selection, &model, &iter))
        {
                gtk_tree_model_get (model, &iter, AUTHOR_COLUMN, &author, -1);

                g_print ("You selected a book by %s\n", author);

                g_free (author);
        }
}
```

Simple Example

Here is a simple example of using a GtkTreeView widget in context of the other widgets. It simply creates a simple model and view, and puts them together. Note that the model is never populated with data — that is left as an exercise for the reader. More information can be found on this in the GtkTreeModel section.

```
enum
{
   TITLE_COLUMN,
   AUTHOR_COLUMN,
   CHECKED_COLUMN,
   N_COLUMNS
```

```
};

void
setup_tree (void)
{
   GtkTreeStore *store;
   GtkWidget *tree;
   GtkTreeViewColumn *column;
   GtkCellRenderer *renderer;

   /* Create a model.  We are using the store model for now, though we
    * could use any other GtkTreeModel */
   store = gtk_tree_store_new (N_COLUMNS,
                               G_TYPE_STRING,
                               G_TYPE_STRING,
                               G_TYPE_BOOLEAN);

   /* custom function to fill the model with data */
   populate_tree_model (store);

   /* Create a view */
   tree = gtk_tree_view_new_with_model (GTK_TREE_MODEL (store));

   /* The view now holds a reference.  We can get rid of our own
    * reference */
   g_object_unref (G_OBJECT (store));

   /* Create a cell render and arbitrarily make it red for demonstration
    * purposes */
   renderer = gtk_cell_renderer_text_new ();
   g_object_set (G_OBJECT (renderer),
                 "foreground", "red",
                 NULL);

   /* Create a column, associating the "text" attribute of the
    * cell_renderer to the first column of the model */
   column = gtk_tree_view_column_new_with_attributes ("Author", renderer,
                                                      "text", AUTHOR_COLUMN,
                                                      NULL);

   /* Add the column to the view. */
   gtk_tree_view_append_column (GTK_TREE_VIEW (tree), column);

   /* Second column.. title of the book. */
   renderer = gtk_cell_renderer_text_new ();
   column = gtk_tree_view_column_new_with_attributes ("Title",
                                                      renderer,
                                                      "text", TITLE_COLUMN,
                                                      NULL);
   gtk_tree_view_append_column (GTK_TREE_VIEW (tree), column);

   /* Last column.. whether a book is checked out. */
   renderer = gtk_cell_renderer_toggle_new ();
   column = gtk_tree_view_column_new_with_attributes ("Checked out",
                                                      renderer,
```

```
                                                    "active", CHECKED_COLUMN,
                                                    NULL);
   gtk_tree_view_append_column (GTK_TREE_VIEW (tree), column);

   /* Now we can manipulate the view just like any other GTK widget */
   ...
}
```

<< Tree, List and Icon Grid Widgets **GtkTreeModel >>**

GtkTreeModel

GtkTreeModel — The tree interface used by GtkTreeView

Synopsis

```
#include <gtk/gtk.h>

            GtkTreeModel;
            GtkTreeIter;
            GtkTreePath;
            GtkTreeRowReference;
            GtkTreeModelIface;
gboolean    (*GtkTreeModelForeachFunc)      (GtkTreeModel *model,
                                             GtkTreePath *path,
                                             GtkTreeIter *iter,
                                             gpointer data);
enum        GtkTreeModelFlags;
GtkTreePath* gtk_tree_path_new              (void);
GtkTreePath* gtk_tree_path_new_from_string  (const gchar *path);
GtkTreePath* gtk_tree_path_new_from_indices (gint first_index,
                                             ...);
gchar*      gtk_tree_path_to_string         (GtkTreePath *path);
GtkTreePath* gtk_tree_path_new_first        (void);
#define     gtk_tree_path_new_root          ()
void        gtk_tree_path_append_index      (GtkTreePath *path,
                                             gint index_);
void        gtk_tree_path_prepend_index     (GtkTreePath *path,
                                             gint index_);
gint        gtk_tree_path_get_depth         (GtkTreePath *path);
gint*       gtk_tree_path_get_indices       (GtkTreePath *path);
void        gtk_tree_path_free              (GtkTreePath *path);
GtkTreePath* gtk_tree_path_copy             (const GtkTreePath *path);
gint        gtk_tree_path_compare           (const GtkTreePath *a,
                                             const GtkTreePath *b);
void        gtk_tree_path_next              (GtkTreePath *path);
```

```
gboolean    gtk_tree_path_prev              (GtkTreePath *path);
gboolean    gtk_tree_path_up                (GtkTreePath *path);
void        gtk_tree_path_down              (GtkTreePath *path);
gboolean    gtk_tree_path_is_ancestor       (GtkTreePath *path,
                                             GtkTreePath *descendant);
gboolean    gtk_tree_path_is_descendant     (GtkTreePath *path,
                                             GtkTreePath *ancestor);
GtkTreeRowReference* gtk_tree_row_reference_new
                                            (GtkTreeModel *model,
                                             GtkTreePath *path);
GtkTreeRowReference* gtk_tree_row_reference_new_proxy
                                            (GObject *proxy,
                                             GtkTreeModel *model,
                                             GtkTreePath *path);
GtkTreePath* gtk_tree_row_reference_get_path
                                            (GtkTreeRowReference *reference);
gboolean    gtk_tree_row_reference_valid    (GtkTreeRowReference *reference);
void        gtk_tree_row_reference_free     (GtkTreeRowReference *reference);
GtkTreeRowReference* gtk_tree_row_reference_copy
                                            (GtkTreeRowReference *reference);
void        gtk_tree_row_reference_inserted (GObject *proxy,
                                             GtkTreePath *path);
void        gtk_tree_row_reference_deleted  (GObject *proxy,
                                             GtkTreePath *path);
void        gtk_tree_row_reference_reordered
                                            (GObject *proxy,
                                             GtkTreePath *path,
                                             GtkTreeIter *iter,
                                             gint *new_order);
GtkTreeIter* gtk_tree_iter_copy             (GtkTreeIter *iter);
void        gtk_tree_iter_free              (GtkTreeIter *iter);
GtkTreeModelFlags gtk_tree_model_get_flags  (GtkTreeModel *tree_model);
gint        gtk_tree_model_get_n_columns    (GtkTreeModel *tree_model);
GType       gtk_tree_model_get_column_type  (GtkTreeModel *tree_model,
                                             gint index_);
gboolean    gtk_tree_model_get_iter         (GtkTreeModel *tree_model,
                                             GtkTreeIter *iter,
                                             GtkTreePath *path);
gboolean    gtk_tree_model_get_iter_from_string
                                            (GtkTreeModel *tree_model,
                                             GtkTreeIter *iter,
                                             const gchar *path_string);
gboolean    gtk_tree_model_get_iter_first   (GtkTreeModel *tree_model,
```

```
                                                GtkTreeIter *iter);
#define     gtk_tree_model_get_iter_root        (tree_model, iter)
GtkTreePath* gtk_tree_model_get_path            (GtkTreeModel *tree_model,
                                                GtkTreeIter *iter);
void        gtk_tree_model_get_value            (GtkTreeModel *tree_model,
                                                GtkTreeIter *iter,
                                                gint column,
                                                GValue *value);
gboolean    gtk_tree_model_iter_next            (GtkTreeModel *tree_model,
                                                GtkTreeIter *iter);
gboolean    gtk_tree_model_iter_children        (GtkTreeModel *tree_model,
                                                GtkTreeIter *iter,
                                                GtkTreeIter *parent);
gboolean    gtk_tree_model_iter_has_child       (GtkTreeModel *tree_model,
                                                GtkTreeIter *iter);
gint        gtk_tree_model_iter_n_children      (GtkTreeModel *tree_model,
                                                GtkTreeIter *iter);
gboolean    gtk_tree_model_iter_nth_child       (GtkTreeModel *tree_model,
                                                GtkTreeIter *iter,
                                                GtkTreeIter *parent,
                                                gint n);
gboolean    gtk_tree_model_iter_parent          (GtkTreeModel *tree_model,
                                                GtkTreeIter *iter,
                                                GtkTreeIter *child);
gchar*      gtk_tree_model_get_string_from_iter
                                               (GtkTreeModel *tree_model,
                                                GtkTreeIter *iter);
void        gtk_tree_model_ref_node             (GtkTreeModel *tree_model,
                                                GtkTreeIter *iter);
void        gtk_tree_model_unref_node           (GtkTreeModel *tree_model,
                                                GtkTreeIter *iter);
void        gtk_tree_model_get                  (GtkTreeModel *tree_model,
                                                GtkTreeIter *iter,
                                                ...);
void        gtk_tree_model_get_valist           (GtkTreeModel *tree_model,
                                                GtkTreeIter *iter,
                                                va_list var_args);
void        gtk_tree_model_foreach              (GtkTreeModel *model,
                                                GtkTreeModelForeachFunc func,
                                                gpointer user_data);
void        gtk_tree_model_row_changed          (GtkTreeModel *tree_model,
                                                GtkTreePath *path,
                                                GtkTreeIter *iter);
```

```
void        gtk_tree_model_row_inserted     (GtkTreeModel *tree_model,
                                             GtkTreePath *path,
                                             GtkTreeIter *iter);
void        gtk_tree_model_row_has_child_toggled
                                            (GtkTreeModel *tree_model,
                                             GtkTreePath *path,
                                             GtkTreeIter *iter);
void        gtk_tree_model_row_deleted      (GtkTreeModel *tree_model,
                                             GtkTreePath *path);
void        gtk_tree_model_rows_reordered   (GtkTreeModel *tree_model,
                                             GtkTreePath *path,
                                             GtkTreeIter *iter,
                                             gint *new_order);
```

Object Hierarchy

```
  GInterface
   +----GtkTreeModel
```

Prerequisites

GtkTreeModel requires GObject.

Known Derived Interfaces

GtkTreeModel is required by GtkTreeSortable.

Known Implementations

GtkTreeModel is implemented by GtkTreeModelSort, GtkTreeStore, GtkListStore and GtkTreeModelFilter.

Signal Prototypes

```
"row-changed"
            void        user_function      (GtkTreeModel *treemodel,
```

```
                                                GtkTreePath *arg1,
                                                GtkTreeIter *arg2,
                                                gpointer user_data);
"row-deleted"
            void        user_function      (GtkTreeModel *treemodel,
                                                GtkTreePath *arg1,
                                                gpointer user_data);
"row-has-child-toggled"
            void        user_function      (GtkTreeModel *treemodel,
                                                GtkTreePath *arg1,
                                                GtkTreeIter *arg2,
                                                gpointer user_data);
"row-inserted"
            void        user_function      (GtkTreeModel *treemodel,
                                                GtkTreePath *arg1,
                                                GtkTreeIter *arg2,
                                                gpointer user_data);
"rows-reordered"
            void        user_function      (GtkTreeModel *treemodel,
                                                GtkTreePath *arg1,
                                                GtkTreeIter *arg2,
                                                gpointer arg3,
                                                gpointer user_data);
```

Description

The GtkTreeModel interface defines a generic tree interface for use by the GtkTreeView widget. It is an abstract interface, and is designed to be usable with any appropriate data structure. The programmer just has to implement this interface on their own data type for it to be viewable by a GtkTreeView widget.

The model is represented as a hierarchical tree of strongly-typed, columned data. In other words, the model can be seen as a tree where every node has different values depending on which column is being queried. The type of data found in a column is determined by using the GType system (ie. G_TYPE_INT, GTK_TYPE_BUTTON, G_TYPE_POINTER, etc.). The types are homogeneous per column across all nodes. It is important to note that this interface only provides a way of examining a model and observing changes. The implementation of each individual model decides how and if changes are made.

In order to make life simpler for programmers who do not need to write their own specialized model, two generic models are provided — the GtkTreeStore and the GtkListStore. To use these, the developer simply pushes data into these models as necessary. These models provide the data structure as well as all appropriate tree interfaces. As a result, implementing drag and drop, sorting, and storing data is trivial. For the vast majority of trees and lists, these two models are sufficient.

Models are accessed on a node/column level of granularity. One can query for the value of a model at a certain node and a certain column on that node. There are two structures used to reference a particular node in a model. They are the GtkTreePath and the GtkTreeIter [4] Most of the interface consists of operations on a GtkTreeIter.

A path is essentially a potential node. It is a location on a model that may or may not actually correspond to a node on a specific model. The GtkTreePath struct can be converted into either an array of unsigned integers or a string. The string form is a list of numbers separated by a colon. Each number refers to the offset at that level. Thus, the path “0” refers to the root node and the path “2:4” refers to the fifth child of the third node.

By contrast, a GtkTreeIter is a reference to a specific node on a specific model. It is a generic struct with an integer and three generic pointers. These are filled in by the model in a model-specific way. One can convert a path to an iterator by calling `gtk_tree_model_get_iter()`. These iterators are the primary way of accessing a model and are similar to the iterators used by GtkTextBuffer. They are generally statically allocated on the heap and only used for a short time. The model interface defines a set of operations using them for navigating the model.

It is expected that models fill in the iterator with private data. For example, the GtkListStore model, which is internally a simple linked list, stores a list node in one of the pointers. The GtkTreeModelSort stores an array and an offset in two of the pointers. Additionally, there is an integer field. This field is generally filled with a unique stamp per model. This stamp is for catching errors resulting from using invalid iterators with a model.

The lifecycle of an iterator can be a little confusing at first. Iterators are expected to always be valid for as long as the model is unchanged (and doesn't emit a signal). The model is considered to own all outstanding iterators and nothing needs to be done to free them from the user's point of view. Additionally, some models guarantee that an iterator is valid for as long as the node it refers to is valid (most notably the GtkTreeStore and GtkListStore). Although generally uninteresting, as one always has to allow for the case where iterators do not persist beyond a signal, some very important performance enhancements were made in the sort model. As a result, the GTK_TREE_MODEL_ITERS_PERSIST flag was added to indicate this behavior.

To help show some common operation of a model, some examples are provided. The first example shows three ways of getting the iter at the location “3:2:5”. While the first method shown is easier, the second is much more common, as you often get paths from callbacks.

Example 1. Acquiring a GtkTreeIter

```
/* Three ways of getting the iter pointing to the location
 */
{
  GtkTreePath *path;
  GtkTreeIter iter;
  GtkTreeIter parent_iter;

  /* get the iterator from a string */
  gtk_tree_model_get_iter_from_string (model, &iter, "3:2:5");

  /* get the iterator from a path */
  path = gtk_tree_path_new_from_string ("3:2:5");
```

```
  gtk_tree_model_get_iter (model, &iter, path);
  gtk_tree_path_free (path);


  /* walk the tree to find the iterator */
  gtk_tree_model_get_nth_child (model, &iter, NULL, 3);
  parent_iter = iter;
  gtk_tree_model_get_nth_child (model, &iter, &parent_iter, 2);
  parent_iter = iter;
  gtk_tree_model_get_nth_child (model, &iter, &parent_iter, 5);
}
```

This second example shows a quick way of iterating through a list and getting a string and an integer from each row. The `populate_model` function used below is not shown, as it is specific to the GtkListStore. For information on how to write such a function, see the GtkListStore documentation.

Example 2. Reading data from a GtkTreeModel

```
enum
{
  STRING_COLUMN,
  INT_COLUMN,
  N_COLUMNS
};

{
  GtkTreeModel *list_store;
  GtkTreeIter iter;
  gboolean valid;
  gint row_count = 0;

  /* make a new list_store */
  list_store = gtk_list_store_new (N_COLUMNS, G_TYPE_STRING, G_TYPE_INT);

  /* Fill the list store with data */
  populate_model (list_store);

  /* Get the first iter in the list */
  valid = gtk_tree_model_get_iter_first (list_store, &iter);

  while (valid)
    {
      /* Walk through the list, reading each row */
      gchar *str_data;
      gint   int_data;

      /* Make sure you terminate calls to gtk_tree_model_get()
```

```
        * with a '-1' value
        */
       gtk_tree_model_get (list_store, &iter,
                           STRING_COLUMN, &str_data,
                           INT_COLUMN, &int_data,
                           -1);

       /* Do something with the data */
       g_print ("Row %d: (%s,%d)\n", row_count, str_data, int_data);
       g_free (str_data);

       row_count ++;
       valid = gtk_tree_model_iter_next (list_store, &iter);
     }
}
```

Details

GtkTreeModel

```
typedef struct _GtkTreeModel GtkTreeModel;
```

GtkTreeIter

```
typedef struct {
  gint stamp;
  gpointer user_data;
  gpointer user_data2;
  gpointer user_data3;
} GtkTreeIter;
```

The GtkTreeIter is the primary structure for accessing a structure. Models are expected to put a unique integer in the *stamp* member, and put model-specific data in the three *user_data* members.

gint *stamp*;	A unique stamp to catch invalid iterators
gpointer *user_data*;	Model specific data
gpointer *user_data2*;	Model specific data
gpointer *user_data3*;	Model specific data

GtkTreePath

```
typedef struct _GtkTreePath GtkTreePath;
```

GtkTreeRowReference

```
typedef struct _GtkTreeRowReference GtkTreeRowReference;
```

GtkTreeModelIface

```
typedef struct {
  GTypeInterface g_iface;

  /* Signals */
  void         (* row_changed)          (GtkTreeModel *tree_model,
                                         GtkTreePath  *path,
                                         GtkTreeIter  *iter);
  void         (* row_inserted)         (GtkTreeModel *tree_model,
                                         GtkTreePath  *path,
                                         GtkTreeIter  *iter);
  void         (* row_has_child_toggled) (GtkTreeModel *tree_model,
                                         GtkTreePath  *path,
                                         GtkTreeIter  *iter);
  void         (* row_deleted)          (GtkTreeModel *tree_model,
                                         GtkTreePath  *path);
  void         (* rows_reordered)       (GtkTreeModel *tree_model,
                                         GtkTreePath  *path,
                                         GtkTreeIter  *iter,
                                         gint         *new_order);

  /* Virtual Table */
  GtkTreeModelFlags (* get_flags)  (GtkTreeModel *tree_model);

  gint         (* get_n_columns)   (GtkTreeModel *tree_model);
  GType        (* get_column_type) (GtkTreeModel *tree_model,
                                    gint          index_);
  gboolean     (* get_iter)        (GtkTreeModel *tree_model,
                                    GtkTreeIter  *iter,
```

```
                                        GtkTreePath  *path);
  GtkTreePath *(* get_path)        (GtkTreeModel *tree_model,
                                        GtkTreeIter  *iter);
  void         (* get_value)       (GtkTreeModel *tree_model,
                                        GtkTreeIter  *iter,
                                        gint          column,
                                        GValue       *value);
  gboolean     (* iter_next)       (GtkTreeModel *tree_model,
                                        GtkTreeIter  *iter);
  gboolean     (* iter_children)   (GtkTreeModel *tree_model,
                                        GtkTreeIter  *iter,
                                        GtkTreeIter  *parent);
  gboolean     (* iter_has_child)  (GtkTreeModel *tree_model,
                                        GtkTreeIter  *iter);
  gint         (* iter_n_children) (GtkTreeModel *tree_model,
                                        GtkTreeIter  *iter);
  gboolean     (* iter_nth_child)  (GtkTreeModel *tree_model,
                                        GtkTreeIter  *iter,
                                        GtkTreeIter  *parent,
                                        gint          n);
  gboolean     (* iter_parent)     (GtkTreeModel *tree_model,
                                        GtkTreeIter  *iter,
                                        GtkTreeIter  *child);
  void         (* ref_node)        (GtkTreeModel *tree_model,
                                        GtkTreeIter  *iter);
  void         (* unref_node)      (GtkTreeModel *tree_model,
                                        GtkTreeIter  *iter);
} GtkTreeModelIface;
```

GtkTreeModelForeachFunc ()

```
gboolean    (*GtkTreeModelForeachFunc)      (GtkTreeModel *model,
                                             GtkTreePath *path,
                                             GtkTreeIter *iter,
                                             gpointer data);
```

model :

path :

iter :

data :

Returns :

enum GtkTreeModelFlags

```
typedef enum
{
  GTK_TREE_MODEL_ITERS_PERSIST = 1 << 0,
  GTK_TREE_MODEL_LIST_ONLY = 1 << 1
} GtkTreeModelFlags;
```

These flags indicate various properties of a GtkTreeModel. They are returned by `gtk_tree_model_get_flags()`, and must be static for the lifetime of the object. A more complete description of GTK_TREE_MODEL_ITERS_PERSIST can be found in the overview of this section.

`GTK_TREE_MODEL_ITERS_PERSIST`	Iterators survive all signals emitted by the tree.
`GTK_TREE_MODEL_LIST_ONLY`	The model is a list only, and never has children

gtk_tree_path_new ()

```
GtkTreePath* gtk_tree_path_new              (void);
```

Creates a new GtkTreePath. This structure refers to a row.

Returns : A newly created GtkTreePath.

gtk_tree_path_new_from_string ()

```
GtkTreePath* gtk_tree_path_new_from_string  (const gchar *path);
```

Creates a new GtkTreePath initialized to *`path`*. *`path`* is expected to be a colon separated list of numbers. For example, the string "10:4:0" would create a path of depth 3 pointing to the 11th child of the root node, the 5th child of that 11th child, and the 1st child of that 5th child. If an invalid path string is passed in, `NULL` is returned.

`path` : The string representation of a path.

Returns : A newly-created GtkTreePath, or `NULL`

gtk_tree_path_new_from_indices ()

```
GtkTreePath* gtk_tree_path_new_from_indices (gint first_index,
                                             ...);
```

Creates a new path with *first_index* and *varargs* as indices.

first_index : first integer
... : list of integers terminated by -1
Returns : A newly created GtkTreePath.

Since 2.2

---

gtk_tree_path_to_string ()

```
gchar*      gtk_tree_path_to_string         (GtkTreePath *path);
```

Generates a string representation of the path. This string is a ':' separated list of numbers. For example, "4:10:0:3" would be an acceptable return value for this string.

path : A GtkTreePath
Returns : A newly-allocated string. Must be freed with g_free().

---

gtk_tree_path_new_first ()

```
GtkTreePath* gtk_tree_path_new_first        (void);
```

Creates a new GtkTreePath. The string representation of this path is "0"

Returns : A new GtkTreePath.

---

gtk_tree_path_new_root()

```
#define gtk_tree_path_new_root() gtk_tree_path_new_first()
```

Warning

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

An alternate name for `gtk_tree_path_new_iter()` provided for compatibility reasons; this macro will be deprecated in future versions of GTK+.

Returns : A new GtkTreePath.

---

gtk_tree_path_append_index ()

```
void        gtk_tree_path_append_index      (GtkTreePath *path,
                                             gint index_);
```

Appends a new index to a path. As a result, the depth of the path is increased.

path : A GtkTreePath.
*index_* : The index.

---

gtk_tree_path_prepend_index ()

```
void        gtk_tree_path_prepend_index     (GtkTreePath *path,
                                             gint index_);
```

Prepends a new index to a path. As a result, the depth of the path is increased.

path : A GtkTreePath.
*index_* : The index.

---

gtk_tree_path_get_depth ()

```
gint        gtk_tree_path_get_depth         (GtkTreePath *path);
```

Returns the current depth of *path*.

path : A GtkTreePath.

Returns : The depth of *path*

---

gtk_tree_path_get_indices ()

```
gint*       gtk_tree_path_get_indices       (GtkTreePath *path);
```

Returns the current indices of *path*. This is an array of integers, each representing a node in a tree. This value should not be freed.

path : A GtkTreePath.

Returns : The current indices, or NULL.

---

gtk_tree_path_free ()

```
void        gtk_tree_path_free              (GtkTreePath *path);
```

Frees *path*.

path : A GtkTreePath.

---

gtk_tree_path_copy ()

```
GtkTreePath* gtk_tree_path_copy             (const GtkTreePath *path);
```

Creates a new GtkTreePath as a copy of *path*.

path : A GtkTreePath.

Returns : A new GtkTreePath.

gtk_tree_path_compare ()

```
gint        gtk_tree_path_compare           (const GtkTreePath *a,
                                             const GtkTreePath *b);
```

Compares two paths. If *a* appears before *b* in a tree, then -1 is returned. If *b* appears before *a*, then 1 is returned. If the two nodes are equal, then 0 is returned.

a : A GtkTreePath.

b : A GtkTreePath to compare with.

Returns : The relative positions of *a* and *b*

gtk_tree_path_next ()

```
void        gtk_tree_path_next              (GtkTreePath *path);
```

Moves the *path* to point to the next node at the current depth.

path : A GtkTreePath.

gtk_tree_path_prev ()

```
gboolean    gtk_tree_path_prev              (GtkTreePath *path);
```

Moves the *path* to point to the previous node at the current depth, if it exists.

path : A GtkTreePath.

Returns : TRUE if *path* has a previous node, and the move was made.

gtk_tree_path_up ()

```
gboolean    gtk_tree_path_up                (GtkTreePath *path);
```

Moves the *path* to point to its parent node, if it has a parent.

path : A GtkTreePath.

Returns : TRUE if *path* has a parent, and the move was made.

---

gtk_tree_path_down ()

```
void        gtk_tree_path_down              (GtkTreePath *path);
```

Moves *path* to point to the first child of the current path.

path : A GtkTreePath.

---

gtk_tree_path_is_ancestor ()

```
gboolean    gtk_tree_path_is_ancestor       (GtkTreePath *path,
                                             GtkTreePath *descendant);
```

Returns TRUE if *descendant* is a descendant of *path*.

path : a GtkTreePath

descendant : another GtkTreePath

Returns : TRUE if *descendant* is contained inside *path*

---

gtk_tree_path_is_descendant ()

```
gboolean    gtk_tree_path_is_descendant     (GtkTreePath *path,
                                             GtkTreePath *ancestor);
```

Returns TRUE if *path* is a descendant of *ancestor*.

path : a GtkTreePath

ancestor : another GtkTreePath

Returns : TRUE if *ancestor* contains *path* somewhere below it

---

gtk_tree_row_reference_new ()

```
GtkTreeRowReference* gtk_tree_row_reference_new
                                                (GtkTreeModel *model,
                                                 GtkTreePath *path);
```

Creates a row reference based on *path*. This reference will keep pointing to the node pointed to by *path*, so long as it exists. It listens to all signals emitted by *model*, and updates its path appropriately. If *path* isn't a valid path in *model*, then NULL is returned.

model : A GtkTreeModel

path : A valid GtkTreePath to monitor

Returns : A newly allocated GtkTreeRowReference, or NULL

---

gtk_tree_row_reference_new_proxy ()

```
GtkTreeRowReference* gtk_tree_row_reference_new_proxy
                                                (GObject *proxy,
                                                 GtkTreeModel *model,
                                                 GtkTreePath *path);
```

You do not need to use this function. Creates a row reference based on *path*. This reference will keep pointing to the node pointed to by *path*, so long as it exists. If *path* isn't a valid path in *model*, then NULL is returned. However, unlike references created with gtk_tree_row_reference_new(), it does not listen to the model for changes. The creator of the row reference must do this explicitly using gtk_tree_row_reference_inserted(), gtk_tree_row_reference_deleted(), gtk_tree_row_reference_reordered().

These functions must be called exactly once per proxy when the corresponding signal on the model is emitted. This single call updates all row references for that proxy. Since built-in GTK+ objects like GtkTreeView already use this mechanism internally, using them as the proxy object will produce unpredictable results. Further more, passing the same object as *model* and *proxy* doesn't work for reasons of internal implementation.

This type of row reference is primarily meant by structures that need to carefully monitor exactly when a row_reference updates itself, and is not generally needed by most applications.

proxy : A proxy GObject
model : A GtkTreeModel
path : A valid GtkTreePath to monitor
Returns : A newly allocated GtkTreeRowReference, or NULL

---

gtk_tree_row_reference_get_path ()

```
GtkTreePath* gtk_tree_row_reference_get_path
                                        (GtkTreeRowReference *reference);
```

Returns a path that the row reference currently points to, or NULL if the path pointed to is no longer valid.

reference : A GtkTreeRowReference
Returns : A current path, or NULL.

---

gtk_tree_row_reference_valid ()

```
gboolean    gtk_tree_row_reference_valid    (GtkTreeRowReference *reference);
```

Returns TRUE if the reference is non-NULL and refers to a current valid path.

reference : A GtkTreeRowReference, or NULL
Returns : TRUE if reference points to a valid path.

---

gtk_tree_row_reference_free ()

```
void        gtk_tree_row_reference_free     (GtkTreeRowReference *reference);
```

Free's reference. reference may be NULL.

reference : A GtkTreeRowReference, or NULL

---

gtk_tree_row_reference_copy ()

```
GtkTreeRowReference* gtk_tree_row_reference_copy
                                                (GtkTreeRowReference *reference);
```

Copies a GtkTreeRowReference.

reference : a GtkTreeRowReference
Returns : a copy of *reference*.

Since 2.2

---

gtk_tree_row_reference_inserted ()

```
void        gtk_tree_row_reference_inserted (GObject *proxy,
                                             GtkTreePath *path);
```

Lets a set of row reference created by `gtk_tree_row_reference_new_proxy()` know that the model emitted the "row_inserted" signal.

proxy : A GObject
path : The row position that was inserted

---

gtk_tree_row_reference_deleted ()

```
void        gtk_tree_row_reference_deleted  (GObject *proxy,
                                             GtkTreePath *path);
```

Lets a set of row reference created by `gtk_tree_row_reference_new_proxy()` know that the model emitted the "row_deleted" signal.

proxy : A GObject

path : The path position that was deleted

---

gtk_tree_row_reference_reordered ()

```
void        gtk_tree_row_reference_reordered
                                            (GObject *proxy,
                                             GtkTreePath *path,
                                             GtkTreeIter *iter,
                                             gint *new_order);
```

Lets a set of row reference created by `gtk_tree_row_reference_new_proxy()` know that the model emitted the "rows_reordered" signal.

proxy : A GObject

path : The parent path of the reordered signal

iter : The iter pointing to the parent of the reordered

new_order : The new order of rows

---

gtk_tree_iter_copy ()

```
GtkTreeIter* gtk_tree_iter_copy             (GtkTreeIter *iter);
```

Creates a dynamically allocated tree iterator as a copy of *iter*. This function is not intended for use in applications, because you can just copy the structs by value (`GtkTreeIter new_iter = iter;`). You must free this iter with `gtk_tree_iter_free()`.

iter : A GtkTreeIter.

Returns : a newly-allocated copy of *iter*.

---

gtk_tree_iter_free ()

```
void        gtk_tree_iter_free              (GtkTreeIter *iter);
```

Frees an iterator that has been allocated on the heap. This function is mainly used for language bindings.

iter : A dynamically allocated tree iterator.

---

gtk_tree_model_get_flags ()

```
GtkTreeModelFlags gtk_tree_model_get_flags  (GtkTreeModel *tree_model);
```

Returns a set of flags supported by this interface. The flags are a bitwise combination of GtkTreeModelFlags. The flags supported should not change during the lifecycle of the tree_model.

tree_model : A GtkTreeModel.
Returns : The flags supported by this interface.

---

gtk_tree_model_get_n_columns ()

```
gint        gtk_tree_model_get_n_columns    (GtkTreeModel *tree_model);
```

Returns the number of columns supported by *tree_model*.

tree_model : A GtkTreeModel.
Returns : The number of columns.

---

gtk_tree_model_get_column_type ()

```
GType       gtk_tree_model_get_column_type  (GtkTreeModel *tree_model,
                                             gint index_);
```

Returns the type of the column.

tree_model : A GtkTreeModel.
*index_* : The column index.
Returns : The type of the column.

---

gtk_tree_model_get_iter ()

```
gboolean    gtk_tree_model_get_iter         (GtkTreeModel *tree_model,
                                             GtkTreeIter *iter,
                                             GtkTreePath *path);
```

Sets *iter* to a valid iterator pointing to *path*.

tree_model : A GtkTreeModel.
iter : The uninitialized GtkTreeIter.
path : The GtkTreePath.
Returns : TRUE, if *iter* was set.

---

gtk_tree_model_get_iter_from_string ()

```
gboolean    gtk_tree_model_get_iter_from_string
                                            (GtkTreeModel *tree_model,
                                             GtkTreeIter *iter,
                                             const gchar *path_string);
```

Sets *iter* to a valid iterator pointing to *path_string*, if it exists. Otherwise, *iter* is left invalid and FALSE is returned.

tree_model : A GtkTreeModel.
iter : An uninitialized GtkTreeIter.
path_string : A string representation of a GtkTreePath.
Returns : TRUE, if *iter* was set.

---

gtk_tree_model_get_iter_first ()

```
gboolean    gtk_tree_model_get_iter_first   (GtkTreeModel *tree_model,
                                             GtkTreeIter *iter);
```

Initializes *iter* with the first iterator in the tree (the one at the path "0") and returns TRUE. Returns FALSE if the tree is empty.

tree_model : A GtkTreeModel.
iter : The uninitialized GtkTreeIter.
Returns : TRUE, if *iter* was set.

---

gtk_tree_model_get_iter_root()

```
#define gtk_tree_model_get_iter_root(tree_model, iter)
gtk_tree_model_get_iter_first(tree_model, iter)
```

Warning

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

A alternate name for `gtk_tree_model_get_iter_first()` provided for compatibility reasons; this macro will be deprecated in future versions of GTK+.

tree_model : A GtkTreeModel.
iter : uninitialized GtkTreeIter.
Returns : TRUE, if *iter* was set.

---

gtk_tree_model_get_path ()

```
GtkTreePath* gtk_tree_model_get_path        (GtkTreeModel *tree_model,
                                             GtkTreeIter *iter);
```

Returns a newly-created GtkTreePath referenced by *iter*. This path should be freed with `gtk_tree_path_free()`.

tree_model : A GtkTreeModel.
iter : The GtkTreeIter.
Returns : a newly-created GtkTreePath.

---

gtk_tree_model_get_value ()

```
void        gtk_tree_model_get_value        (GtkTreeModel *tree_model,
                                             GtkTreeIter *iter,
                                             gint column,
                                             GValue *value);
```

Sets initializes and sets *value* to that at *column*. When done with *value*, g_value_unset() needs to be called to free any allocated memory.

tree_model : A GtkTreeModel.
iter : The GtkTreeIter.
column : The column to lookup the value at.
value : An empty GValue to set.

---

gtk_tree_model_iter_next ()

```
gboolean    gtk_tree_model_iter_next        (GtkTreeModel *tree_model,
                                             GtkTreeIter *iter);
```

Sets *iter* to point to the node following it at the current level. If there is no next *iter*, FALSE is returned and *iter* is set to be invalid.

tree_model : A GtkTreeModel.
iter : The GtkTreeIter.
Returns : TRUE if *iter* has been changed to the next node.

---

gtk_tree_model_iter_children ()

```
gboolean    gtk_tree_model_iter_children    (GtkTreeModel *tree_model,
                                             GtkTreeIter *iter,
                                             GtkTreeIter *parent);
```

Sets *iter* to point to the first child of *parent*. If *parent* has no children, FALSE is returned and *iter* is set to be invalid. *parent* will remain a valid node after this function has been called.

If *parent* is NULL returns the first node, equivalent to gtk_tree_model_get_iter_first (tree_model, iter);

tree_model : A GtkTreeModel.
iter : The new GtkTreeIter to be set to the child.
parent : The GtkTreeIter, or NULL
Returns : TRUE, if *child* has been set to the first child.

---

gtk_tree_model_iter_has_child ()

```
gboolean    gtk_tree_model_iter_has_child   (GtkTreeModel *tree_model,
                                             GtkTreeIter *iter);
```

Returns TRUE if *iter* has children, FALSE otherwise.

tree_model : A GtkTreeModel.
iter : The GtkTreeIter to test for children.
Returns : TRUE if *iter* has children.

---

gtk_tree_model_iter_n_children ()

```
gint        gtk_tree_model_iter_n_children  (GtkTreeModel *tree_model,
                                             GtkTreeIter *iter);
```

Returns the number of children that *iter* has. As a special case, if *iter* is NULL, then the number of toplevel nodes is returned.

tree_model : A GtkTreeModel.
iter : The GtkTreeIter, or NULL.
Returns : The number of children of *iter*.

---

gtk_tree_model_iter_nth_child ()

```
gboolean    gtk_tree_model_iter_nth_child   (GtkTreeModel *tree_model,
                                             GtkTreeIter *iter,
                                             GtkTreeIter *parent,
                                             gint n);
```

Sets *iter* to be the child of *parent*, using the given index. The first index is 0. If *n* is too big, or *parent* has no children, *iter* is set to an invalid iterator and FALSE is returned. *parent* will remain a valid node after this function has been called. As a special case, if *parent* is NULL, then the *n*th root node is set.

tree_model : A GtkTreeModel.
iter : The GtkTreeIter to set to the nth child.
parent : The GtkTreeIter to get the child from, or NULL.
n : Then index of the desired child.
Returns : TRUE, if *parent* has an *n*th child.

---

gtk_tree_model_iter_parent ()

```
gboolean    gtk_tree_model_iter_parent      (GtkTreeModel *tree_model,
                                             GtkTreeIter *iter,
                                             GtkTreeIter *child);
```

Sets *iter* to be the parent of *child*. If *child* is at the toplevel, and doesn't have a parent, then *iter* is set to an invalid iterator and FALSE is returned. *child* will remain a valid node after this function has been called.

tree_model : A GtkTreeModel
iter : The new GtkTreeIter to set to the parent.
child : The GtkTreeIter.
Returns : TRUE, if *iter* is set to the parent of *child*.

---

gtk_tree_model_get_string_from_iter ()

```
gchar*      gtk_tree_model_get_string_from_iter
                                            (GtkTreeModel *tree_model,
                                             GtkTreeIter *iter);
```

Generates a string representation of the iter. This string is a ':' separated list of numbers. For example, "4:10:0:3"

would be an acceptable return value for this string.

tree_model : A GtkTreeModel.
iter : An GtkTreeIter.
Returns : A newly-allocated string. Must be freed with g_free().

Since 2.2

---

gtk_tree_model_ref_node ()

```
void        gtk_tree_model_ref_node         (GtkTreeModel *tree_model,
                                             GtkTreeIter *iter);
```

Lets the tree ref the node. This is an optional method for models to implement. To be more specific, models may ignore this call as it exists primarily for performance reasons.

This function is primarily meant as a way for views to let caching model know when nodes are being displayed (and hence, whether or not to cache that node.) For example, a file-system based model would not want to keep the entire file-hierarchy in memory, just the sections that are currently being displayed by every current view.

A model should be expected to be able to get an iter independent of its reffed state.

tree_model : A GtkTreeModel.
iter : The GtkTreeIter.

---

gtk_tree_model_unref_node ()

```
void        gtk_tree_model_unref_node       (GtkTreeModel *tree_model,
                                             GtkTreeIter *iter);
```

Lets the tree unref the node. This is an optional method for models to implement. To be more specific, models may ignore this call as it exists primarily for performance reasons.

For more information on what this means, see gtk_tree_model_ref_node(). Please note that nodes that are deleted are not unreffed.

tree_model : A GtkTreeModel.
iter : The GtkTreeIter.

---

gtk_tree_model_get ()

```
void        gtk_tree_model_get              (GtkTreeModel *tree_model,
                                             GtkTreeIter *iter,
                                             ...);
```

Gets the value of one or more cells in the row referenced by *iter*. The variable argument list should contain integer column numbers, each column number followed by a place to store the value being retrieved. The list is terminated by a -1. For example, to get a value from column 0 with type `G_TYPE_STRING`, you would write: `gtk_tree_model_get (model, iter, 0, &place_string_here, -1)`, where `place_string_here` is a gchar* to be filled with the string. If appropriate, the returned values have to be freed or unreferenced.

tree_model : a GtkTreeModel
iter : a row in *tree_model*
... : pairs of column number and value return locations, terminated by -1

---

gtk_tree_model_get_valist ()

```
void        gtk_tree_model_get_valist       (GtkTreeModel *tree_model,
                                             GtkTreeIter *iter,
                                             va_list var_args);
```

See `gtk_tree_model_get()`, this version takes a va_list for language bindings to use.

tree_model : a GtkTreeModel
iter : a row in *tree_model*
var_args : va_list of column/return location pairs

---

gtk_tree_model_foreach ()

```
void        gtk_tree_model_foreach          (GtkTreeModel *model,
```

```
                                        GtkTreeModelForeachFunc func,
                                        gpointer user_data);
```

Calls func on each node in model in a depth-first fashion. If func returns TRUE, then the tree ceases to be walked, and `gtk_tree_model_foreach()` returns.

model : A GtkTreeModel
func : A function to be called on each row
user_data : User data to passed to func.

---

gtk_tree_model_row_changed ()

```
void        gtk_tree_model_row_changed      (GtkTreeModel *tree_model,
                                             GtkTreePath *path,
                                             GtkTreeIter *iter);
```

Emits the "row_changed" signal on *tree_model*.

tree_model : A GtkTreeModel
path : A GtkTreePath pointing to the changed row
iter : A valid GtkTreeIter pointing to the changed row

---

gtk_tree_model_row_inserted ()

```
void        gtk_tree_model_row_inserted     (GtkTreeModel *tree_model,
                                             GtkTreePath *path,
                                             GtkTreeIter *iter);
```

Emits the "row_inserted" signal on *tree_model*

tree_model : A GtkTreeModel
path : A GtkTreePath pointing to the inserted row
iter : A valid GtkTreeIter pointing to the inserted row

---

gtk_tree_model_row_has_child_toggled ()

```
void        gtk_tree_model_row_has_child_toggled
                                            (GtkTreeModel *tree_model,
                                             GtkTreePath *path,
                                             GtkTreeIter *iter);
```

Emits the "row_has_child_toggled" signal on *tree_model*. This should be called by models after the child state of a node changes.

tree_model : A GtkTreeModel
path : A GtkTreePath pointing to the changed row
iter : A valid GtkTreeIter pointing to the changed row

---

gtk_tree_model_row_deleted ()

```
void        gtk_tree_model_row_deleted      (GtkTreeModel *tree_model,
                                             GtkTreePath *path);
```

Emits the "row_deleted" signal on *tree_model*. This should be called by models after a row has been removed. The location pointed to by *path* should be the location that the row previously was at. It may not be a valid location anymore.

tree_model : A GtkTreeModel
path : A GtkTreePath pointing to the previous location of the deleted row.

---

gtk_tree_model_rows_reordered ()

```
void        gtk_tree_model_rows_reordered   (GtkTreeModel *tree_model,
                                             GtkTreePath *path,
                                             GtkTreeIter *iter,
                                             gint *new_order);
```

Emits the "rows_reordered" signal on *tree_model*. This should be called by models when their rows have been reordered.

tree_model : A GtkTreeModel

path : A GtkTreePath pointing to the tree node whose children have been reordered

iter : A valid GtkTreeIter pointing to the node whose children have been reordered

new_order : an array of integers mapping the current position of each child to its old position before the re-ordering, i.e. *new_order*[newpos] = oldpos.

Signals

The "row-changed" signal

```
void        user_function                   (GtkTreeModel *treemodel,
                                             GtkTreePath *arg1,
                                             GtkTreeIter *arg2,
                                             gpointer user_data);
```

treemodel : the object which received the signal.

arg1 :

arg2 :

user_data : user data set when the signal handler was connected.

The "row-deleted" signal

```
void        user_function                   (GtkTreeModel *treemodel,
                                             GtkTreePath *arg1,
                                             gpointer user_data);
```

treemodel : the object which received the signal.

arg1 :

user_data : user data set when the signal handler was connected.

The "row-has-child-toggled" signal

```
void        user_function                   (GtkTreeModel *treemodel,
                                             GtkTreePath *arg1,
```

```
                                                GtkTreeIter *arg2,
                                                gpointer user_data);
```

treemodel : the object which received the signal.
arg1 :
arg2 :
user_data : user data set when the signal handler was connected.

The "row-inserted" signal

```
void        user_function                   (GtkTreeModel *treemodel,
                                             GtkTreePath *arg1,
                                             GtkTreeIter *arg2,
                                             gpointer user_data);
```

treemodel : the object which received the signal.
arg1 :
arg2 :
user_data : user data set when the signal handler was connected.

The "rows-reordered" signal

```
void        user_function                   (GtkTreeModel *treemodel,
                                             GtkTreePath *arg1,
                                             GtkTreeIter *arg2,
                                             gpointer arg3,
                                             gpointer user_data);
```

treemodel : the object which received the signal.
arg1 :
arg2 :
arg3 :
user_data : user data set when the signal handler was connected.

See Also

GtkTreeView, GtkTreeStore, GtkListStore, GtkTreeDnd, GtkTreeSortable

---

[4] Here, iter is short for “iterator”

<< Tree and List Widget Overview **GtkTreeSelection >>**

GtkTreeSelection

GtkTreeSelection — The selection object for GtkTreeView

Synopsis

```
#include <gtk/gtk.h>

            GtkTreeSelection;
gboolean    (*GtkTreeSelectionFunc)         (GtkTreeSelection *selection,
                                             GtkTreeModel *model,
                                             GtkTreePath *path,
                                             gboolean path_currently_selected,
                                             gpointer data);
void        (*GtkTreeSelectionForeachFunc)  (GtkTreeModel *model,
                                             GtkTreePath *path,
                                             GtkTreeIter *iter,
                                             gpointer data);
void        gtk_tree_selection_set_mode     (GtkTreeSelection *selection,
                                             GtkSelectionMode type);
GtkSelectionMode gtk_tree_selection_get_mode
                                            (GtkTreeSelection *selection);
void        gtk_tree_selection_set_select_function
                                            (GtkTreeSelection *selection,
                                             GtkTreeSelectionFunc func,
                                             gpointer data,
                                             GtkDestroyNotify destroy);
gpointer    gtk_tree_selection_get_user_data
                                            (GtkTreeSelection *selection);
GtkTreeView* gtk_tree_selection_get_tree_view
                                            (GtkTreeSelection *selection);
gboolean    gtk_tree_selection_get_selected (GtkTreeSelection *selection,
                                             GtkTreeModel **model,
                                             GtkTreeIter *iter);
void        gtk_tree_selection_selected_foreach
                                            (GtkTreeSelection *selection,
```

```
                                                GtkTreeSelectionForeachFunc func,
                                                gpointer data);
GList*      gtk_tree_selection_get_selected_rows
                                               (GtkTreeSelection *selection,
                                                GtkTreeModel **model);
gint        gtk_tree_selection_count_selected_rows
                                               (GtkTreeSelection *selection);
void        gtk_tree_selection_select_path  (GtkTreeSelection *selection,
                                                GtkTreePath *path);
void        gtk_tree_selection_unselect_path
                                               (GtkTreeSelection *selection,
                                                GtkTreePath *path);
gboolean    gtk_tree_selection_path_is_selected
                                               (GtkTreeSelection *selection,
                                                GtkTreePath *path);
void        gtk_tree_selection_select_iter  (GtkTreeSelection *selection,
                                                GtkTreeIter *iter);
void        gtk_tree_selection_unselect_iter
                                               (GtkTreeSelection *selection,
                                                GtkTreeIter *iter);
gboolean    gtk_tree_selection_iter_is_selected
                                               (GtkTreeSelection *selection,
                                                GtkTreeIter *iter);
void        gtk_tree_selection_select_all   (GtkTreeSelection *selection);
void        gtk_tree_selection_unselect_all (GtkTreeSelection *selection);
void        gtk_tree_selection_select_range (GtkTreeSelection *selection,
                                                GtkTreePath *start_path,
                                                GtkTreePath *end_path);
void        gtk_tree_selection_unselect_range
                                               (GtkTreeSelection *selection,
                                                GtkTreePath *start_path,
                                                GtkTreePath *end_path);
```

Object Hierarchy

```
  GObject
   +----GtkTreeSelection
```

Signal Prototypes

```
"changed"   void        user_function      (GtkTreeSelection *treeselection,
                                            gpointer user_data);
```

Description

The GtkTreeSelection object is a helper object to manage the selection for a GtkTreeView widget. The GtkTreeSelection object is automatically created when a new GtkTreeView widget is created, and cannot exist independentally of this widget. The primary reason the GtkTreeSelection objects exists is for cleanliness of code and API. That is, there is no conceptual reason all these functions could not be methods on the GtkTreeView widget instead of a separate function.

The GtkTreeSelection object is gotten from a GtkTreeView by calling `gtk_tree_view_get_selection()`. It can be manipulated to check the selection status of the tree, as well as select and deselect individual rows. Selection is done completely view side. As a result, multiple views of the same model can have completely different selections. Additionally, you cannot change the selection of a row on the model that is not currently displayed by the view without expanding its parents first.

One of the important things to remember when monitoring the selection of a view is that the "changed" signal is mostly a hint. That is, it may only emit one signal when a range of rows is selected. Additionally, it may on occasion emit a "changed" signal when nothing has happened (mostly as a result of programmers calling select_row on an already selected row).

Details

GtkTreeSelection

```
typedef struct _GtkTreeSelection GtkTreeSelection;
```

GtkTreeSelectionFunc ()

```
gboolean    (*GtkTreeSelectionFunc)         (GtkTreeSelection *selection,
                                             GtkTreeModel *model,
                                             GtkTreePath *path,
                                             gboolean path_currently_selected,
                                             gpointer data);
```

A function used by `gtk_tree_selection_set_select_function()` to filter whether or not a row may be selected. It is called whenever a row's state might change. A return value of TRUE indicates to *`selection`* that it is okay to change the selection.

`selection` :	A GtkTreeSelection
`model` :	A GtkTreeModel being viewed
`path` :	The GtkTreePath of the row in question
`path_currently_selected` :	TRUE, if the path is currently selected
`data` :	user data
Returns :	TRUE, if the selection state of the row can be toggled

---

GtkTreeSelectionForeachFunc ()

```
void        (*GtkTreeSelectionForeachFunc)  (GtkTreeModel *model,
                                             GtkTreePath *path,
                                             GtkTreeIter *iter,
                                             gpointer data);
```

A function used by `gtk_tree_selection_selected_foreach()` to map all selected rows. It will be called on every selected row in the view.

`model` :	The GtkTreeModel being viewed
`path` :	The GtkTreePath of a selected row
`iter` :	A GtkTreeIter pointing to a selected row
`data` :	user data

---

gtk_tree_selection_set_mode ()

```
void        gtk_tree_selection_set_mode     (GtkTreeSelection *selection,
                                             GtkSelectionMode type);
```

Sets the selection mode of the *`selection`*. If the previous type was GTK_SELECTION_MULTIPLE, then the anchor is kept selected, if it was previously selected.

`selection` : A GtkTreeSelection.

type : The selection mode

---

gtk_tree_selection_get_mode ()

```
GtkSelectionMode gtk_tree_selection_get_mode
                                        (GtkTreeSelection *selection);
```

Gets the selection mode for *selection*. See gtk_tree_selection_set_mode().

selection : a GtkTreeSelection
Returns : the current selection mode

---

gtk_tree_selection_set_select_function ()

```
void        gtk_tree_selection_set_select_function
                                        (GtkTreeSelection *selection,
                                         GtkTreeSelectionFunc func,
                                         gpointer data,
                                         GtkDestroyNotify destroy);
```

Sets the selection function. If set, this function is called before any node is selected or unselected, giving some control over which nodes are selected. The select function should return TRUE if the state of the node may be toggled, and FALSE if the state of the node should be left unchanged.

selection : A GtkTreeSelection.
func : The selection function.
data : The selection function's data.
destroy : The destroy function for user data. May be NULL.

---

gtk_tree_selection_get_user_data ()

```
gpointer    gtk_tree_selection_get_user_data
                                        (GtkTreeSelection *selection);
```

Returns the user data for the selection function.

selection : A GtkTreeSelection.

Returns : The user data.

---

gtk_tree_selection_get_tree_view ()

```
GtkTreeView* gtk_tree_selection_get_tree_view
                                        (GtkTreeSelection *selection);
```

Returns the tree view associated with *selection*.

selection : A GtkTreeSelection

Returns : A GtkTreeView

---

gtk_tree_selection_get_selected ()

```
gboolean    gtk_tree_selection_get_selected (GtkTreeSelection *selection,
                                             GtkTreeModel **model,
                                             GtkTreeIter *iter);
```

Sets *iter* to the currently selected node if *selection* is set to GTK_SELECTION_SINGLE or GTK_SELECTION_BROWSE. *iter* may be NULL if you just want to test if *selection* has any selected nodes. *model* is filled with the current model as a convenience. This function will not work if you use *selection* is GTK_SELECTION_MULTIPLE.

selection : A GtkTreeSelection.

model : A pointer to set to the GtkTreeModel, or NULL.

iter : The GtkTreeIter, or NULL.

Returns : TRUE, if there is a selected node.

---

gtk_tree_selection_selected_foreach ()

```
void        gtk_tree_selection_selected_foreach
                                        (GtkTreeSelection *selection,
                                         GtkTreeSelectionForeachFunc func,
```

```
                                                gpointer data);
```

Calls a function for each selected node. Note that you cannot modify the tree or selection from within this function. As a result, `gtk_tree_selection_get_selected_rows()` might be more useful.

selection : A GtkTreeSelection.
func : The function to call for each selected node.
data : user data to pass to the function.

---

gtk_tree_selection_get_selected_rows ()

```
GList*      gtk_tree_selection_get_selected_rows
                                            (GtkTreeSelection *selection,
                                             GtkTreeModel **model);
```

Creates a list of path of all selected rows. Additionally, if you are planning on modifying the model after calling this function, you may want to convert the returned list into a list of GtkTreeRowReferences. To do this, you can use `gtk_tree_row_reference_new()`.

To free the return value, use:

```
g_list_foreach (list, gtk_tree_path_free, NULL);
g_list_free (list);
```

selection : A GtkTreeSelection.
model : A pointer to set to the GtkTreeModel, or NULL.
Returns : A GList containing a GtkTreePath for each selected row.

Since 2.2

---

gtk_tree_selection_count_selected_rows ()

```
gint        gtk_tree_selection_count_selected_rows
                                            (GtkTreeSelection *selection);
```

Returns the number of rows that have been selected in *tree*.

selection : A GtkTreeSelection.

Returns : The number of rows selected.

Since 2.2

---

gtk_tree_selection_select_path ()

```
void        gtk_tree_selection_select_path  (GtkTreeSelection *selection,
                                             GtkTreePath *path);
```

Select the row at *path*.

selection : A GtkTreeSelection.

path : The GtkTreePath to be selected.

---

gtk_tree_selection_unselect_path ()

```
void        gtk_tree_selection_unselect_path
                                            (GtkTreeSelection *selection,
                                             GtkTreePath *path);
```

Unselects the row at *path*.

selection : A GtkTreeSelection.

path : The GtkTreePath to be unselected.

---

gtk_tree_selection_path_is_selected ()

```
gboolean    gtk_tree_selection_path_is_selected
                                            (GtkTreeSelection *selection,
                                             GtkTreePath *path);
```

Returns TRUE if the row pointed to by *path* is currently selected. If *path* does not point to a valid location, FALSE is returned

selection : A GtkTreeSelection.
path : A GtkTreePath to check selection on.
Returns : TRUE if *path* is selected.

---

gtk_tree_selection_select_iter ()

```
void        gtk_tree_selection_select_iter  (GtkTreeSelection *selection,
                                             GtkTreeIter *iter);
```

Selects the specified iterator.

selection : A GtkTreeSelection.
iter : The GtkTreeIter to be selected.

---

gtk_tree_selection_unselect_iter ()

```
void        gtk_tree_selection_unselect_iter
                                            (GtkTreeSelection *selection,
                                             GtkTreeIter *iter);
```

Unselects the specified iterator.

selection : A GtkTreeSelection.
iter : The GtkTreeIter to be unselected.

---

gtk_tree_selection_iter_is_selected ()

```
gboolean    gtk_tree_selection_iter_is_selected
                                            (GtkTreeSelection *selection,
                                             GtkTreeIter *iter);
```

Returns TRUE if the row at *iter* is currently selected.

selection : A GtkTreeSelection
iter : A valid GtkTreeIter
Returns : TRUE, if *iter* is selected

---

gtk_tree_selection_select_all ()

```
void        gtk_tree_selection_select_all   (GtkTreeSelection *selection);
```

Selects all the nodes. *selection* must be set to GTK_SELECTION_MULTIPLE mode.

selection : A GtkTreeSelection.

---

gtk_tree_selection_unselect_all ()

```
void        gtk_tree_selection_unselect_all (GtkTreeSelection *selection);
```

Unselects all the nodes.

selection : A GtkTreeSelection.

---

gtk_tree_selection_select_range ()

```
void        gtk_tree_selection_select_range (GtkTreeSelection *selection,
                                             GtkTreePath *start_path,
                                             GtkTreePath *end_path);
```

Selects a range of nodes, determined by *start_path* and *end_path* inclusive. *selection* must be set to GTK_SELECTION_MULTIPLE mode.

selection : A GtkTreeSelection.
start_path : The initial node of the range.
end_path : The final node of the range.

gtk_tree_selection_unselect_range ()

```
void        gtk_tree_selection_unselect_range
                                            (GtkTreeSelection *selection,
                                             GtkTreePath *start_path,
                                             GtkTreePath *end_path);
```

Unselects a range of nodes, determined by *start_path* and *end_path* inclusive.

selection : A GtkTreeSelection.
start_path : The initial node of the range.
end_path : The initial node of the range.

Since 2.2

Signals

The "changed" signal

```
void        user_function                  (GtkTreeSelection *treeselection,
                                            gpointer user_data);
```

Emitted whenever the selection has (possibly) changed. Please note that this signal is mostly a hint. It may only be emitted once when a range of rows are selected, and it may occasionally be emitted when nothing has happened.

treeselection : the object which received the signal.
user_data : user data set when the signal handler was connected.

See Also

GtkTreeView, GtkTreeViewColumn, GtkTreeDnd, GtkTreeMode, GtkTreeSortable, GtkTreeModelSort, GtkListStore, GtkTreeStore, GtkCellRenderer, GtkCellEditable, GtkCellRendererPixbuf, GtkCellRendererText, GtkCellRendererToggle

<< GtkTreeModel **GtkTreeViewColumn >>**

GtkTreeViewColumn

GtkTreeViewColumn — A visible column in a GtkTreeView widget

Synopsis

```
#include <gtk/gtk.h>

enum        GtkTreeViewColumnSizing;
void        (*GtkTreeCellDataFunc)          (GtkTreeViewColumn *tree_column,
                                             GtkCellRenderer *cell,
                                             GtkTreeModel *tree_model,
                                             GtkTreeIter *iter,
                                             gpointer data);
            GtkTreeViewColumn;
GtkTreeViewColumn* gtk_tree_view_column_new (void);
GtkTreeViewColumn* gtk_tree_view_column_new_with_attributes
                                            (const gchar *title,
                                             GtkCellRenderer *cell,
                                             ...);
void        gtk_tree_view_column_pack_start (GtkTreeViewColumn *tree_column,
                                             GtkCellRenderer *cell,
                                             gboolean expand);
void        gtk_tree_view_column_pack_end   (GtkTreeViewColumn *tree_column,
                                             GtkCellRenderer *cell,
                                             gboolean expand);
void        gtk_tree_view_column_clear      (GtkTreeViewColumn *tree_column);
GList*      gtk_tree_view_column_get_cell_renderers
                                            (GtkTreeViewColumn *tree_column);
void        gtk_tree_view_column_add_attribute
                                            (GtkTreeViewColumn *tree_column,
                                             GtkCellRenderer *cell_renderer,
                                             const gchar *attribute,
                                             gint column);
void        gtk_tree_view_column_set_attributes
                                            (GtkTreeViewColumn *tree_column,
                                             GtkCellRenderer *cell_renderer,
                                             ...);
```

```
void        gtk_tree_view_column_set_cell_data_func
                                            (GtkTreeViewColumn *tree_column,
                                             GtkCellRenderer *cell_renderer,
                                             GtkTreeCellDataFunc func,
                                             gpointer func_data,
                                             GtkDestroyNotify destroy);
void        gtk_tree_view_column_clear_attributes
                                            (GtkTreeViewColumn *tree_column,
                                             GtkCellRenderer *cell_renderer);
void        gtk_tree_view_column_set_spacing
                                            (GtkTreeViewColumn *tree_column,
                                             gint spacing);
gint        gtk_tree_view_column_get_spacing
                                            (GtkTreeViewColumn *tree_column);
void        gtk_tree_view_column_set_visible
                                            (GtkTreeViewColumn *tree_column,
                                             gboolean visible);
gboolean    gtk_tree_view_column_get_visible
                                            (GtkTreeViewColumn *tree_column);
void        gtk_tree_view_column_set_resizable
                                            (GtkTreeViewColumn *tree_column,
                                             gboolean resizable);
gboolean    gtk_tree_view_column_get_resizable
                                            (GtkTreeViewColumn *tree_column);
void        gtk_tree_view_column_set_sizing (GtkTreeViewColumn *tree_column,
                                             GtkTreeViewColumnSizing type);
GtkTreeViewColumnSizing gtk_tree_view_column_get_sizing
                                            (GtkTreeViewColumn *tree_column);
gint        gtk_tree_view_column_get_width  (GtkTreeViewColumn *tree_column);
gint        gtk_tree_view_column_get_fixed_width
                                            (GtkTreeViewColumn *tree_column);
void        gtk_tree_view_column_set_fixed_width
                                            (GtkTreeViewColumn *tree_column,
                                             gint fixed_width);
void        gtk_tree_view_column_set_min_width
                                            (GtkTreeViewColumn *tree_column,
                                             gint min_width);
gint        gtk_tree_view_column_get_min_width
                                            (GtkTreeViewColumn *tree_column);
void        gtk_tree_view_column_set_max_width
                                            (GtkTreeViewColumn *tree_column,
                                             gint max_width);
gint        gtk_tree_view_column_get_max_width
                                            (GtkTreeViewColumn *tree_column);
void        gtk_tree_view_column_clicked    (GtkTreeViewColumn *tree_column);
```

```
void        gtk_tree_view_column_set_title  (GtkTreeViewColumn *tree_column,
                                             const gchar *title);
G_CONST_RETURN gchar* gtk_tree_view_column_get_title
                                            (GtkTreeViewColumn *tree_column);
void        gtk_tree_view_column_set_expand (GtkTreeViewColumn *tree_column,
                                             gboolean expand);
gboolean    gtk_tree_view_column_get_expand (GtkTreeViewColumn *tree_column);
void        gtk_tree_view_column_set_clickable
                                            (GtkTreeViewColumn *tree_column,
                                             gboolean clickable);
gboolean    gtk_tree_view_column_get_clickable
                                            (GtkTreeViewColumn *tree_column);
void        gtk_tree_view_column_set_widget (GtkTreeViewColumn *tree_column,
                                             GtkWidget *widget);
GtkWidget*  gtk_tree_view_column_get_widget (GtkTreeViewColumn *tree_column);
void        gtk_tree_view_column_set_alignment
                                            (GtkTreeViewColumn *tree_column,
                                             gfloat xalign);
gfloat      gtk_tree_view_column_get_alignment
                                            (GtkTreeViewColumn *tree_column);
void        gtk_tree_view_column_set_reorderable
                                            (GtkTreeViewColumn *tree_column,
                                             gboolean reorderable);
gboolean    gtk_tree_view_column_get_reorderable
                                            (GtkTreeViewColumn *tree_column);
void        gtk_tree_view_column_set_sort_column_id
                                            (GtkTreeViewColumn *tree_column,
                                             gint sort_column_id);
gint        gtk_tree_view_column_get_sort_column_id
                                            (GtkTreeViewColumn *tree_column);
void        gtk_tree_view_column_set_sort_indicator
                                            (GtkTreeViewColumn *tree_column,
                                             gboolean setting);
gboolean    gtk_tree_view_column_get_sort_indicator
                                            (GtkTreeViewColumn *tree_column);
void        gtk_tree_view_column_set_sort_order
                                            (GtkTreeViewColumn *tree_column,
                                             GtkSortType order);
GtkSortType gtk_tree_view_column_get_sort_order
                                            (GtkTreeViewColumn *tree_column);
void        gtk_tree_view_column_cell_set_cell_data
                                            (GtkTreeViewColumn *tree_column,
                                             GtkTreeModel *tree_model,
                                             GtkTreeIter *iter,
                                             gboolean is_expander,
```

```
                                                gboolean is_expanded);
void        gtk_tree_view_column_cell_get_size
                                               (GtkTreeViewColumn *tree_column,
                                                GdkRectangle *cell_area,
                                                gint *x_offset,
                                                gint *y_offset,
                                                gint *width,
                                                gint *height);
gboolean    gtk_tree_view_column_cell_get_position
                                               (GtkTreeViewColumn *tree_column,
                                                GtkCellRenderer *cell_renderer,
                                                gint *start_pos,
                                                gint *width);
gboolean    gtk_tree_view_column_cell_is_visible
                                               (GtkTreeViewColumn *tree_column);
void        gtk_tree_view_column_focus_cell (GtkTreeViewColumn *tree_column,
                                                GtkCellRenderer *cell);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkTreeViewColumn
```

Implemented Interfaces

GtkTreeViewColumn implements GtkCellLayout.

Properties

```
  "alignment"           gfloat              : Read / Write
  "clickable"           gboolean            : Read / Write
  "expand"              gboolean            : Read / Write
  "fixed-width"         gint                : Read / Write
  "max-width"           gint                : Read / Write
  "min-width"           gint                : Read / Write
  "reorderable"         gboolean            : Read / Write
  "resizable"           gboolean            : Read / Write
```

```
  "sizing"              GtkTreeViewColumnSizing  : Read / Write
  "sort-indicator"      gboolean                 : Read / Write
  "sort-order"          GtkSortType              : Read / Write
  "spacing"             gint                     : Read / Write
  "title"               gchararray               : Read / Write
  "visible"             gboolean                 : Read / Write
  "widget"              GtkWidget                : Read / Write
  "width"               gint                     : Read
```

Signal Prototypes

```
"clicked"   void        user_function      (GtkTreeViewColumn *treeviewcolumn,
                                            gpointer user_data);
```

Description

The GtkTreeViewColumn object is a visible column in a GtkTreeView widget.

Details

enum GtkTreeViewColumnSizing

```
typedef enum
{
  GTK_TREE_VIEW_COLUMN_GROW_ONLY,
  GTK_TREE_VIEW_COLUMN_AUTOSIZE,
  GTK_TREE_VIEW_COLUMN_FIXED
} GtkTreeViewColumnSizing;
```

The sizing method the column uses to determine its width. Please note that *GTK_TREE_VIEW_COLUMN_AUTOSIZE* are inefficient for large views, and can make columns appear choppy.

GTK_TREE_VIEW_COLUMN_GROW_ONLY	Columns only get bigger in reaction to changes in the model
GTK_TREE_VIEW_COLUMN_AUTOSIZE	Columns resize to be the optimal size everytime the model changes.
GTK_TREE_VIEW_COLUMN_FIXED	Columns are a fixed numbers of pixels wide.

GtkTreeCellDataFunc ()

```
void        (*GtkTreeCellDataFunc)          (GtkTreeViewColumn *tree_column,
                                             GtkCellRenderer *cell,
                                             GtkTreeModel *tree_model,
                                             GtkTreeIter *iter,
                                             gpointer data);
```

A function to set the properties of a cell instead of just using the straight mapping between the cell and the model. This is useful for customizing the cell renderer. For example, a function might get an integer from the *tree_model*, and render it to the "text" attribute of "cell" by converting it to its written equivilent. This is set by calling gtk_tree_view_column_set_cell_data_func()

tree_column : A GtkTreeColumn
cell : The GtkCellRenderer that is being rendered by *tree_column*
tree_model : The GtkTreeModel being rendered
iter : A GtkTreeIter of the current row rendered
data : user data

GtkTreeViewColumn

```
typedef struct _GtkTreeViewColumn GtkTreeViewColumn;
```

---

gtk_tree_view_column_new ()

```
GtkTreeViewColumn* gtk_tree_view_column_new (void);
```

Creates a new GtkTreeViewColumn.

Returns : A newly created GtkTreeViewColumn.

---

gtk_tree_view_column_new_with_attributes ()

```
GtkTreeViewColumn* gtk_tree_view_column_new_with_attributes
                                            (const gchar *title,
                                             GtkCellRenderer *cell,
                                             ...);
```

Creates a new GtkTreeViewColumn with a number of default values. This is equivalent to calling `gtk_tree_view_column_set_title()`, `gtk_tree_view_column_pack_start()`, and `gtk_tree_view_column_set_attributes()` on the newly created GtkTreeViewColumn.

Here's a simple example:

```
enum { TEXT_COLUMN, COLOR_COLUMN, N_COLUMNS };
...
{
  GtkTreeViewColumn *column;
  GtkCellRenderer   *renderer = gtk_cell_renderer_text_new ();

  column = gtk_tree_view_column_new_with_attributes ("Title",
                                                     renderer,
                                                     "text", TEXT_COLUMN,
                                                     "foreground", COLOR_COLUMN,
                                                     NULL);
}
```

title : The title to set the header to.
cell : The GtkCellRenderer.
... : A NULL-terminated list of attributes.
Returns : A newly created GtkTreeViewColumn.

---

gtk_tree_view_column_pack_start ()

```
void        gtk_tree_view_column_pack_start (GtkTreeViewColumn *tree_column,
                                             GtkCellRenderer *cell,
                                             gboolean expand);
```

Packs the *cell* into the beginning of the column. If *expand* is FALSE, then the *cell* is allocated no more space than it needs. Any unused space is divided evenly between cells for which *expand* is TRUE.

tree_column : A GtkTreeViewColumn.
cell : The GtkCellRenderer.
expand : TRUE if *cell* is to be given extra space allocated to *tree_column*.

---

gtk_tree_view_column_pack_end ()

```
void        gtk_tree_view_column_pack_end   (GtkTreeViewColumn *tree_column,
                                             GtkCellRenderer *cell,
                                             gboolean expand);
```

Adds the *cell* to end of the column. If *expand* is FALSE, then the *cell* is allocated no more space than it needs. Any unused space is divided evenly between cells for which *expand* is TRUE.

tree_column : A GtkTreeViewColumn.
cell : The GtkCellRenderer.
expand : TRUE if *cell* is to be given extra space allocated to *tree_column*.

---

gtk_tree_view_column_clear ()

```
void        gtk_tree_view_column_clear      (GtkTreeViewColumn *tree_column);
```

Unsets all the mappings on all renderers on the *tree_column*.

tree_column : A GtkTreeViewColumn

---

gtk_tree_view_column_get_cell_renderers ()

```
GList*      gtk_tree_view_column_get_cell_renderers
                                            (GtkTreeViewColumn *tree_column);
```

Returns a newly-allocated GList of all the cell renderers in the column, in no particular order. The list must be freed with g_list_free().

tree_column : A GtkTreeViewColumn
Returns : A list of GtkCellRenderers

---

gtk_tree_view_column_add_attribute ()

```
void        gtk_tree_view_column_add_attribute
                                            (GtkTreeViewColumn *tree_column,
                                             GtkCellRenderer *cell_renderer,
```

```
                                                const gchar *attribute,
                                                gint column);
```

Adds an attribute mapping to the list in *tree_column*. The *column* is the column of the model to get a value from, and the *attribute* is the parameter on *cell_renderer* to be set from the value. So for example if column 2 of the model contains strings, you could have the "text" attribute of a GtkCellRendererText get its values from column 2.

tree_column :	A GtkTreeViewColumn.
cell_renderer :	the GtkCellRenderer to set attributes on
attribute :	An attribute on the renderer
column :	The column position on the model to get the attribute from.

---

gtk_tree_view_column_set_attributes ()

```
void        gtk_tree_view_column_set_attributes
                                               (GtkTreeViewColumn *tree_column,
                                                GtkCellRenderer *cell_renderer,
                                                ...);
```

Sets the attributes in the list as the attributes of *tree_column*. The attributes should be in attribute/column order, as in gtk_tree_view_column_add_attribute(). All existing attributes are removed, and replaced with the new attributes.

tree_column :	A GtkTreeViewColumn.
cell_renderer :	the GtkCellRenderer we're setting the attributes of
... :	A NULL-terminated list of attributes.

---

gtk_tree_view_column_set_cell_data_func ()

```
void        gtk_tree_view_column_set_cell_data_func
                                               (GtkTreeViewColumn *tree_column,
                                                GtkCellRenderer *cell_renderer,
                                                GtkTreeCellDataFunc func,
                                                gpointer func_data,
                                                GtkDestroyNotify destroy);
```

Sets the GtkTreeViewColumnFunc to use for the column. This function is used instead of the standard attributes mapping for setting the column value, and should set the value of *tree_column*'s cell renderer as appropriate. *func* may be NULL to remove an older one.

tree_column : A GtkTreeViewColumn
cell_renderer : A GtkCellRenderer
func : The GtkTreeViewColumnFunc to use.
func_data : The user data for *func*.
destroy : The destroy notification for *func_data*

---

gtk_tree_view_column_clear_attributes ()

```
void        gtk_tree_view_column_clear_attributes
                                            (GtkTreeViewColumn *tree_column,
                                             GtkCellRenderer *cell_renderer);
```

Clears all existing attributes previously set with gtk_tree_view_column_set_attributes().

tree_column : a GtkTreeViewColumn
cell_renderer : a GtkCellRenderer to clear the attribute mapping on.

---

gtk_tree_view_column_set_spacing ()

```
void        gtk_tree_view_column_set_spacing
                                            (GtkTreeViewColumn *tree_column,
                                             gint spacing);
```

Sets the spacing field of *tree_column*, which is the number of pixels to place between cell renderers packed into it.

tree_column : A GtkTreeViewColumn.
spacing : distance between cell renderers in pixels.

---

gtk_tree_view_column_get_spacing ()

```
gint        gtk_tree_view_column_get_spacing
                                            (GtkTreeViewColumn *tree_column);
```

Returns the spacing of *tree_column*.

tree_column : A GtkTreeViewColumn.

Returns : the spacing of *tree_column*.

---

gtk_tree_view_column_set_visible ()

```
void        gtk_tree_view_column_set_visible
                                            (GtkTreeViewColumn *tree_column,
                                             gboolean visible);
```

Sets the visibility of *tree_column*.

tree_column : A GtkTreeViewColumn.

visible : TRUE if the *tree_column* is visible.

---

gtk_tree_view_column_get_visible ()

```
gboolean    gtk_tree_view_column_get_visible
                                            (GtkTreeViewColumn *tree_column);
```

Returns TRUE if *tree_column* is visible.

tree_column : A GtkTreeViewColumn.

Returns : whether the column is visible or not. If it is visible, then the tree will show the column.

---

gtk_tree_view_column_set_resizable ()

```
void        gtk_tree_view_column_set_resizable
                                            (GtkTreeViewColumn *tree_column,
                                             gboolean resizable);
```

If *resizable* is TRUE, then the user can explicitly resize the column by grabbing the outer edge of the column button. If resizable is TRUE and sizing mode of the column is GTK_TREE_VIEW_COLUMN_AUTOSIZE, then the sizing mode is changed to GTK_TREE_VIEW_COLUMN_GROW_ONLY.

tree_column : A GtkTreeViewColumn

resizable : TRUE, if the column can be resized

gtk_tree_view_column_get_resizable ()

```
gboolean    gtk_tree_view_column_get_resizable
                                            (GtkTreeViewColumn *tree_column);
```

Returns TRUE if the *tree_column* can be resized by the end user.

tree_column : A GtkTreeViewColumn
Returns : TRUE, if the *tree_column* can be resized.

gtk_tree_view_column_set_sizing ()

```
void        gtk_tree_view_column_set_sizing (GtkTreeViewColumn *tree_column,
                                             GtkTreeViewColumnSizing type);
```

Sets the growth behavior of *tree_column* to *type*.

tree_column : A GtkTreeViewColumn.
type : The GtkTreeViewColumnSizing.

gtk_tree_view_column_get_sizing ()

```
GtkTreeViewColumnSizing gtk_tree_view_column_get_sizing
                                            (GtkTreeViewColumn *tree_column);
```

Returns the current type of *tree_column*.

tree_column : A GtkTreeViewColumn.
Returns : The type of *tree_column*.

gtk_tree_view_column_get_width ()

```
gint        gtk_tree_view_column_get_width  (GtkTreeViewColumn *tree_column);
```

Returns the current size of *tree_column* in pixels.

tree_column : A GtkTreeViewColumn.

Returns : The current width of *tree_column*.

---

gtk_tree_view_column_get_fixed_width ()

```
gint        gtk_tree_view_column_get_fixed_width
                                            (GtkTreeViewColumn *tree_column);
```

Gets the fixed width of the column. This value is only meaning may not be the actual width of the column on the screen, just what is requested.

tree_column : a GtkTreeViewColumn

Returns : the fixed width of the column

---

gtk_tree_view_column_set_fixed_width ()

```
void        gtk_tree_view_column_set_fixed_width
                                            (GtkTreeViewColumn *tree_column,
                                             gint fixed_width);
```

Sets the size of the column in pixels. This is meaningful only if the sizing type is GTK_TREE_VIEW_COLUMN_FIXED. The size of the column is clamped to the min/max width for the column. Please note that the min/max width of the column doesn't actually affect the "fixed_width" property of the widget, just the actual size when displayed.

tree_column : A GtkTreeViewColumn.

fixed_width : The size to set *tree_column* to. Must be greater than 0.

---

gtk_tree_view_column_set_min_width ()

```
void        gtk_tree_view_column_set_min_width
                                            (GtkTreeViewColumn *tree_column,
                                             gint min_width);
```

Sets the minimum width of the *tree_column*. If *min_width* is -1, then the minimum width is unset.

tree_column : A GtkTreeViewColumn.
min_width : The minimum width of the column in pixels, or -1.

---

gtk_tree_view_column_get_min_width ()

```
gint        gtk_tree_view_column_get_min_width
                                            (GtkTreeViewColumn *tree_column);
```

Returns the minimum width in pixels of the *tree_column*, or -1 if no minimum width is set.

tree_column : A GtkTreeViewColumn.
Returns : The minimum width of the *tree_column*.

---

gtk_tree_view_column_set_max_width ()

```
void        gtk_tree_view_column_set_max_width
                                            (GtkTreeViewColumn *tree_column,
                                             gint max_width);
```

Sets the maximum width of the *tree_column*. If *max_width* is -1, then the maximum width is unset. Note, the column can actually be wider than max width if it's the last column in a view. In this case, the column expands to fill any extra space.

tree_column : A GtkTreeViewColumn.
max_width : The maximum width of the column in pixels, or -1.

---

gtk_tree_view_column_get_max_width ()

```
gint        gtk_tree_view_column_get_max_width
                                            (GtkTreeViewColumn *tree_column);
```

Returns the maximum width in pixels of the *tree_column*, or -1 if no maximum width is set.

tree_column : A GtkTreeViewColumn.

Returns : The maximum width of the *tree_column*.

---

gtk_tree_view_column_clicked ()

```
void        gtk_tree_view_column_clicked    (GtkTreeViewColumn *tree_column);
```

Emits the "clicked" signal on the column. This function will only work if *tree_column* is clickable.

tree_column : a GtkTreeViewColumn

---

gtk_tree_view_column_set_title ()

```
void        gtk_tree_view_column_set_title  (GtkTreeViewColumn *tree_column,
                                             const gchar *title);
```

Sets the title of the *tree_column*. If a custom widget has been set, then this value is ignored.

tree_column : A GtkTreeViewColumn.
title : The title of the *tree_column*.

---

gtk_tree_view_column_get_title ()

```
G_CONST_RETURN gchar* gtk_tree_view_column_get_title
                                            (GtkTreeViewColumn *tree_column);
```

Returns the title of the widget.

tree_column : A GtkTreeViewColumn.
Returns : the title of the column. This string should not be modified or freed.

---

gtk_tree_view_column_set_expand ()

```
void        gtk_tree_view_column_set_expand (GtkTreeViewColumn *tree_column,
                                             gboolean expand);
```

Sets the column to take available extra space. This space is shared equally amongst all columns that have the expand set to TRUE. If no column has this option set, then the last column gets all extra space. By default, every column is created with this FALSE.

tree_column : A GtkTreeViewColumn
expand :

Since 2.4

---

gtk_tree_view_column_get_expand ()

```
gboolean    gtk_tree_view_column_get_expand (GtkTreeViewColumn *tree_column);
```

Return TRUE if the column expands to take any available space.

tree_column :
Returns : TRUE, if the column expands

Since 2.4

---

gtk_tree_view_column_set_clickable ()

```
void        gtk_tree_view_column_set_clickable
                                            (GtkTreeViewColumn *tree_column,
                                             gboolean clickable);
```

Sets the header to be active if *active* is TRUE. When the header is active, then it can take keyboard focus, and can be clicked.

tree_column : A GtkTreeViewColumn.
clickable : TRUE if the header is active.

---

gtk_tree_view_column_get_clickable ()

```
gboolean    gtk_tree_view_column_get_clickable
                                            (GtkTreeViewColumn *tree_column);
```

Returns TRUE if the user can click on the header for the column.

tree_column : a GtkTreeViewColumn

Returns : TRUE if user can click the column header.

---

gtk_tree_view_column_set_widget ()

```
void        gtk_tree_view_column_set_widget (GtkTreeViewColumn *tree_column,
                                             GtkWidget *widget);
```

Sets the widget in the header to be *widget*. If widget is NULL, then the header button is set with a GtkLabel set to the title of *tree_column*.

tree_column : A GtkTreeViewColumn.

widget : A child GtkWidget, or NULL.

---

gtk_tree_view_column_get_widget ()

```
GtkWidget*  gtk_tree_view_column_get_widget (GtkTreeViewColumn *tree_column);
```

Returns the GtkWidget in the button on the column header. If a custom widget has not been set then NULL is returned.

tree_column : A GtkTreeViewColumn.

Returns : The GtkWidget in the column header, or NULL

---

gtk_tree_view_column_set_alignment ()

```
void        gtk_tree_view_column_set_alignment
                                            (GtkTreeViewColumn *tree_column,
                                             gfloat xalign);
```

Sets the alignment of the title or custom widget inside the column header. The alignment determines its location inside the

button -- 0.0 for left, 0.5 for center, 1.0 for right.

tree_column : A GtkTreeViewColumn.
xalign : The alignment, which is between [0.0 and 1.0] inclusive.

---

gtk_tree_view_column_get_alignment ()

```
gfloat      gtk_tree_view_column_get_alignment
                                        (GtkTreeViewColumn *tree_column);
```

Returns the current x alignment of *tree_column*. This value can range between 0.0 and 1.0.

tree_column : A GtkTreeViewColumn.
Returns : The current alignent of *tree_column*.

---

gtk_tree_view_column_set_reorderable ()

```
void        gtk_tree_view_column_set_reorderable
                                        (GtkTreeViewColumn *tree_column,
                                         gboolean reorderable);
```

If *reorderable* is TRUE, then the column can be reordered by the end user dragging the header.

tree_column : A GtkTreeViewColumn
reorderable : TRUE, if the column can be reordered.

---

gtk_tree_view_column_get_reorderable ()

```
gboolean    gtk_tree_view_column_get_reorderable
                                        (GtkTreeViewColumn *tree_column);
```

Returns TRUE if the *tree_column* can be reordered by the user.

tree_column : A GtkTreeViewColumn
Returns : TRUE if the *tree_column* can be reordered by the user.

---

gtk_tree_view_column_set_sort_column_id ()

```
void        gtk_tree_view_column_set_sort_column_id
                                            (GtkTreeViewColumn *tree_column,
                                             gint sort_column_id);
```

Sets the logical *sort_column_id* that this column sorts on when this column is selected for sorting. Doing so makes the column header clickable.

tree_column : a GtkTreeViewColumn

sort_column_id : The *sort_column_id* of the model to sort on.

---

gtk_tree_view_column_get_sort_column_id ()

```
gint        gtk_tree_view_column_get_sort_column_id
                                            (GtkTreeViewColumn *tree_column);
```

Gets the logical *sort_column_id* that the model sorts on when this column is selected for sorting. See gtk_tree_view_column_set_sort_column_id().

tree_column : a GtkTreeViewColumn

Returns : the current *sort_column_id* for this column, or -1 if this column can't be used for sorting.

---

gtk_tree_view_column_set_sort_indicator ()

```
void        gtk_tree_view_column_set_sort_indicator
                                            (GtkTreeViewColumn *tree_column,
                                             gboolean setting);
```

Call this function with a *setting* of TRUE to display an arrow in the header button indicating the column is sorted. Call gtk_tree_view_column_set_sort_order() to change the direction of the arrow.

tree_column : a GtkTreeViewColumn

setting : TRUE to display an indicator that the column is sorted

---

gtk_tree_view_column_get_sort_indicator ()

```
gboolean    gtk_tree_view_column_get_sort_indicator
                                            (GtkTreeViewColumn *tree_column);
```

Gets the value set by gtk_tree_view_column_set_sort_indicator().

tree_column : a GtkTreeViewColumn

Returns : whether the sort indicator arrow is displayed

---

gtk_tree_view_column_set_sort_order ()

```
void        gtk_tree_view_column_set_sort_order
                                            (GtkTreeViewColumn *tree_column,
                                             GtkSortType order);
```

Changes the appearance of the sort indicator.

This *does not* actually sort the model. Use gtk_tree_view_column_set_sort_column_id() if you want automatic sorting support. This function is primarily for custom sorting behavior, and should be used in conjunction with gtk_tree_sortable_set_sort_column() to do that. For custom models, the mechanism will vary.

The sort indicator changes direction to indicate normal sort or reverse sort. Note that you must have the sort indicator enabled to see anything when calling this function; see gtk_tree_view_column_set_sort_indicator().

tree_column : a GtkTreeViewColumn

order : sort order that the sort indicator should indicate

---

gtk_tree_view_column_get_sort_order ()

```
GtkSortType gtk_tree_view_column_get_sort_order
                                            (GtkTreeViewColumn *tree_column);
```

Gets the value set by gtk_tree_view_column_set_sort_order().

tree_column : a GtkTreeViewColumn

Returns : the sort order the sort indicator is indicating

gtk_tree_view_column_cell_set_cell_data ()

```
void        gtk_tree_view_column_cell_set_cell_data
                                            (GtkTreeViewColumn *tree_column,
                                             GtkTreeModel *tree_model,
                                             GtkTreeIter *iter,
                                             gboolean is_expander,
                                             gboolean is_expanded);
```

Sets the cell renderer based on the *tree_model* and *iter*. That is, for every attribute mapping in *tree_column*, it will get a value from the set column on the *iter*, and use that value to set the attribute on the cell renderer. This is used primarily by the GtkTreeView.

tree_column : A GtkTreeViewColumn.
tree_model : The GtkTreeModel to to get the cell renderers attributes from.
iter : The GtkTreeIter to to get the cell renderer's attributes from.
is_expander : TRUE, if the row has children
is_expanded : TRUE, if the row has visible children

gtk_tree_view_column_cell_get_size ()

```
void        gtk_tree_view_column_cell_get_size
                                            (GtkTreeViewColumn *tree_column,
                                             GdkRectangle *cell_area,
                                             gint *x_offset,
                                             gint *y_offset,
                                             gint *width,
                                             gint *height);
```

Obtains the width and height needed to render the column. This is used primarily by the GtkTreeView.

tree_column : A GtkTreeViewColumn.
cell_area : The area a cell in the column will be allocated, or NULL
x_offset : location to return x offset of a cell relative to *cell_area*, or NULL
y_offset : location to return y offset of a cell relative to *cell_area*, or NULL
width : location to return width needed to render a cell, or NULL
height : location to return height needed to render a cell, or NULL

gtk_tree_view_column_cell_get_position ()

```
gboolean    gtk_tree_view_column_cell_get_position
                                            (GtkTreeViewColumn *tree_column,
                                             GtkCellRenderer *cell_renderer,
                                             gint *start_pos,
                                             gint *width);
```

Obtains the horizontal position and size of a cell in a column. If the cell is not found in the column, *start_pos* and *width* are not changed and FALSE is returned.

tree_column :	a GtkTreeViewColumn
cell_renderer :	a GtkCellRenderer
start_pos :	return location for the horizontal position of *cell* within *tree_column*, may be NULL
width :	return location for the width of *cell*, may be NULL
Returns :	TRUE if *cell* belongs to *tree_column*.

---

gtk_tree_view_column_cell_is_visible ()

```
gboolean    gtk_tree_view_column_cell_is_visible
                                            (GtkTreeViewColumn *tree_column);
```

Returns TRUE if any of the cells packed into the *tree_column* are visible. For this to be meaningful, you must first initialize the cells with gtk_tree_view_column_cell_set_cell_data()

tree_column :	A GtkTreeViewColumn
Returns :	TRUE, if any of the cells packed into the *tree_column* are currently visible

---

gtk_tree_view_column_focus_cell ()

```
void        gtk_tree_view_column_focus_cell (GtkTreeViewColumn *tree_column,
                                             GtkCellRenderer *cell);
```

Sets the current keyboard focus to be at *cell*, if the column contains 2 or more editable and activatable cells.

tree_column : A GtkTreeViewColumn
cell : A GtkCellRenderer

Since 2.2

Properties

The "alignment" property

"alignment"	gfloat	: Read / Write

X Alignment of the column header text or widget.

Allowed values: [0,1]

Default value: 0

The "clickable" property

"clickable"	gboolean	: Read / Write

Whether the header can be clicked.

Default value: FALSE

The "expand" property

"expand"	gboolean	: Read / Write

Column gets share of extra width allocated to the widget.

Default value: FALSE

The "fixed-width" property

```
"fixed-width"          gint                  : Read / Write
```

Current fixed width of the column.

Allowed values: >= 1

Default value: 1

The "max-width" property

```
"max-width"            gint                  : Read / Write
```

Maximum allowed width of the column.

Allowed values: >= -1

Default value: -1

The "min-width" property

```
"min-width"            gint                  : Read / Write
```

Minimum allowed width of the column.

Allowed values: >= -1

Default value: -1

The "reorderable" property

```
"reorderable"          gboolean              : Read / Write
```

Whether the column can be reordered around the headers.

Default value: FALSE

The "resizable" property

```
"resizable"            gboolean              : Read / Write
```

Column is user-resizable.

Default value: FALSE

The "sizing" property

```
"sizing"               GtkTreeViewColumnSizing  : Read / Write
```

Resize mode of the column.

Default value: GTK_TREE_VIEW_COLUMN_GROW_ONLY

The "sort-indicator" property

```
"sort-indicator"       gboolean              : Read / Write
```

Whether to show a sort indicator.

Default value: FALSE

The "sort-order" property

```
"sort-order"           GtkSortType           : Read / Write
```

Sort direction the sort indicator should indicate.

Default value: GTK_SORT_ASCENDING

The "spacing" property

"spacing"	gint	: Read / Write

Space which is inserted between cells.

Allowed values: >= 0

Default value: 0

The "title" property

"title"	gchararray	: Read / Write

Title to appear in column header.

Default value: ""

The "visible" property

"visible"	gboolean	: Read / Write

Whether to display the column.

Default value: TRUE

The "widget" property

"widget"	GtkWidget	: Read / Write

Widget to put in column header button instead of column title.

The "width" property

```
  "width"               gint                  : Read
```

Current width of the column.

Allowed values: >= 0

Default value: 0

Signals

The "clicked" signal

```
void        user_function                  (GtkTreeViewColumn *treeviewcolumn,
                                            gpointer user_data);
```

treeviewcolumn : the object which received the signal.
user_data : user data set when the signal handler was connected.

See Also

GtkTreeView, GtkTreeSelection, GtkTreeDnd, GtkTreeMode, GtkTreeSortable, GtkTreeModelSort, GtkListStore, GtkTreeStore, GtkCellRenderer, GtkCellEditable, GtkCellRendererPixbuf, GtkCellRendererText, GtkCellRendererToggle

<< GtkTreeSelection **GtkTreeView >>**

GtkTreeView

GtkTreeView — A widget for displaying both trees and lists

Synopsis

```
#include <gtk/gtk.h>

            GtkTreeView;
enum        GtkTreeViewDropPosition;
            GtkTreeViewPrivate;
gboolean    (*GtkTreeViewColumnDropFunc)    (GtkTreeView *tree_view,
                                             GtkTreeViewColumn *column,
                                             GtkTreeViewColumn *prev_column,
                                             GtkTreeViewColumn *next_column,
                                             gpointer data);
void        (*GtkTreeViewMappingFunc)       (GtkTreeView *tree_view,
                                             GtkTreePath *path,
                                             gpointer user_data);
gboolean    (*GtkTreeViewSearchEqualFunc)   (GtkTreeModel *model,
                                             gint column,
                                             const gchar *key,
                                             GtkTreeIter *iter,
                                             gpointer search_data);
GtkWidget*  gtk_tree_view_new               (void);
GtkWidget*  gtk_tree_view_new_with_model    (GtkTreeModel *model);
GtkTreeModel* gtk_tree_view_get_model       (GtkTreeView *tree_view);
void        gtk_tree_view_set_model         (GtkTreeView *tree_view,
                                             GtkTreeModel *model);
GtkTreeSelection* gtk_tree_view_get_selection
                                            (GtkTreeView *tree_view);
GtkAdjustment* gtk_tree_view_get_hadjustment
                                            (GtkTreeView *tree_view);
void        gtk_tree_view_set_hadjustment   (GtkTreeView *tree_view,
                                             GtkAdjustment *adjustment);
GtkAdjustment* gtk_tree_view_get_vadjustment
                                            (GtkTreeView *tree_view);
```

```
void        gtk_tree_view_set_vadjustment   (GtkTreeView *tree_view,
                                             GtkAdjustment *adjustment);
gboolean    gtk_tree_view_get_headers_visible
                                            (GtkTreeView *tree_view);
void        gtk_tree_view_set_headers_visible
                                            (GtkTreeView *tree_view,
                                             gboolean headers_visible);
void        gtk_tree_view_columns_autosize  (GtkTreeView *tree_view);
void        gtk_tree_view_set_headers_clickable
                                            (GtkTreeView *tree_view,
                                             gboolean setting);
void        gtk_tree_view_set_rules_hint    (GtkTreeView *tree_view,
                                             gboolean setting);
gboolean    gtk_tree_view_get_rules_hint    (GtkTreeView *tree_view);
gint        gtk_tree_view_append_column     (GtkTreeView *tree_view,
                                             GtkTreeViewColumn *column);
gint        gtk_tree_view_remove_column     (GtkTreeView *tree_view,
                                             GtkTreeViewColumn *column);
gint        gtk_tree_view_insert_column     (GtkTreeView *tree_view,
                                             GtkTreeViewColumn *column,
                                             gint position);
gint        gtk_tree_view_insert_column_with_attributes
                                            (GtkTreeView *tree_view,
                                             gint position,
                                             const gchar *title,
                                             GtkCellRenderer *cell,
                                             ...);
gint        gtk_tree_view_insert_column_with_data_func
                                            (GtkTreeView *tree_view,
                                             gint position,
                                             const gchar *title,
                                             GtkCellRenderer *cell,
                                             GtkTreeCellDataFunc func,
                                             gpointer data,
                                             GDestroyNotify dnotify);
GtkTreeViewColumn* gtk_tree_view_get_column (GtkTreeView *tree_view,
                                             gint n);
GList*      gtk_tree_view_get_columns       (GtkTreeView *tree_view);
void        gtk_tree_view_move_column_after (GtkTreeView *tree_view,
                                             GtkTreeViewColumn *column,
                                             GtkTreeViewColumn *base_column);
void        gtk_tree_view_set_expander_column
                                            (GtkTreeView *tree_view,
                                             GtkTreeViewColumn *column);
```

```
GtkTreeViewColumn* gtk_tree_view_get_expander_column
                                        (GtkTreeView *tree_view);
void        gtk_tree_view_set_column_drag_function
                                        (GtkTreeView *tree_view,
                                         GtkTreeViewColumnDropFunc func,
                                         gpointer user_data,
                                         GtkDestroyNotify destroy);
void        gtk_tree_view_scroll_to_point   (GtkTreeView *tree_view,
                                         gint tree_x,
                                         gint tree_y);
void        gtk_tree_view_scroll_to_cell    (GtkTreeView *tree_view,
                                         GtkTreePath *path,
                                         GtkTreeViewColumn *column,
                                         gboolean use_align,
                                         gfloat row_align,
                                         gfloat col_align);
void        gtk_tree_view_set_cursor        (GtkTreeView *tree_view,
                                         GtkTreePath *path,
                                         GtkTreeViewColumn *focus_column,
                                         gboolean start_editing);
void        gtk_tree_view_set_cursor_on_cell
                                        (GtkTreeView *tree_view,
                                         GtkTreePath *path,
                                         GtkTreeViewColumn *focus_column,
                                         GtkCellRenderer *focus_cell,
                                         gboolean start_editing);
void        gtk_tree_view_get_cursor        (GtkTreeView *tree_view,
                                         GtkTreePath **path,
                                         GtkTreeViewColumn **focus_column);
void        gtk_tree_view_row_activated     (GtkTreeView *tree_view,
                                         GtkTreePath *path,
                                         GtkTreeViewColumn *column);
void        gtk_tree_view_expand_all        (GtkTreeView *tree_view);
void        gtk_tree_view_collapse_all      (GtkTreeView *tree_view);
void        gtk_tree_view_expand_to_path    (GtkTreeView *tree_view,
                                         GtkTreePath *path);
gboolean    gtk_tree_view_expand_row        (GtkTreeView *tree_view,
                                         GtkTreePath *path,
                                         gboolean open_all);
gboolean    gtk_tree_view_collapse_row      (GtkTreeView *tree_view,
                                         GtkTreePath *path);
void        gtk_tree_view_map_expanded_rows (GtkTreeView *tree_view,
                                         GtkTreeViewMappingFunc func,
                                         gpointer data);
```

```
gboolean    gtk_tree_view_row_expanded      (GtkTreeView *tree_view,
                                             GtkTreePath *path);
void        gtk_tree_view_set_reorderable   (GtkTreeView *tree_view,
                                             gboolean reorderable);
gboolean    gtk_tree_view_get_reorderable   (GtkTreeView *tree_view);
gboolean    gtk_tree_view_get_path_at_pos   (GtkTreeView *tree_view,
                                             gint x,
                                             gint y,
                                             GtkTreePath **path,
                                             GtkTreeViewColumn **column,
                                             gint *cell_x,
                                             gint *cell_y);
void        gtk_tree_view_get_cell_area     (GtkTreeView *tree_view,
                                             GtkTreePath *path,
                                             GtkTreeViewColumn *column,
                                             GdkRectangle *rect);
void        gtk_tree_view_get_background_area
                                            (GtkTreeView *tree_view,
                                             GtkTreePath *path,
                                             GtkTreeViewColumn *column,
                                             GdkRectangle *rect);
void        gtk_tree_view_get_visible_rect  (GtkTreeView *tree_view,
                                             GdkRectangle *visible_rect);
GdkWindow*  gtk_tree_view_get_bin_window    (GtkTreeView *tree_view);
void        gtk_tree_view_widget_to_tree_coords
                                            (GtkTreeView *tree_view,
                                             gint wx,
                                             gint wy,
                                             gint *tx,
                                             gint *ty);
void        gtk_tree_view_tree_to_widget_coords
                                            (GtkTreeView *tree_view,
                                             gint tx,
                                             gint ty,
                                             gint *wx,
                                             gint *wy);
void        gtk_tree_view_enable_model_drag_dest
                                            (GtkTreeView *tree_view,
                                             const GtkTargetEntry *targets,
                                             gint n_targets,
                                             GdkDragAction actions);
void        gtk_tree_view_enable_model_drag_source
                                            (GtkTreeView *tree_view,
                                             GdkModifierType start_button_mask,
```

```
                                                const GtkTargetEntry *targets,
                                                gint n_targets,
                                                GdkDragAction actions);
void        gtk_tree_view_unset_rows_drag_source
                                               (GtkTreeView *tree_view);
void        gtk_tree_view_unset_rows_drag_dest
                                               (GtkTreeView *tree_view);
void        gtk_tree_view_set_drag_dest_row (GtkTreeView *tree_view,
                                                GtkTreePath *path,
                                                GtkTreeViewDropPosition pos);
void        gtk_tree_view_get_drag_dest_row (GtkTreeView *tree_view,
                                                GtkTreePath **path,
                                                GtkTreeViewDropPosition *pos);
gboolean    gtk_tree_view_get_dest_row_at_pos
                                               (GtkTreeView *tree_view,
                                                gint drag_x,
                                                gint drag_y,
                                                GtkTreePath **path,
                                                GtkTreeViewDropPosition *pos);
GdkPixmap*  gtk_tree_view_create_row_drag_icon
                                               (GtkTreeView *tree_view,
                                                GtkTreePath *path);
void        gtk_tree_view_set_enable_search (GtkTreeView *tree_view,
                                                gboolean enable_search);
gboolean    gtk_tree_view_get_enable_search (GtkTreeView *tree_view);
gint        gtk_tree_view_get_search_column (GtkTreeView *tree_view);
void        gtk_tree_view_set_search_column (GtkTreeView *tree_view,
                                                gint column);
GtkTreeViewSearchEqualFunc gtk_tree_view_get_search_equal_func
                                               (GtkTreeView *tree_view);
void        gtk_tree_view_set_search_equal_func
                                               (GtkTreeView *tree_view,
                                                GtkTreeViewSearchEqualFunc
search_equal_func,
                                                gpointer search_user_data,
                                                GtkDestroyNotify search_destroy);
gboolean    gtk_tree_view_get_fixed_height_mode
                                               (GtkTreeView *tree_view);
void        gtk_tree_view_set_fixed_height_mode
                                               (GtkTreeView *tree_view,
                                                gboolean enable);
gboolean    gtk_tree_view_get_hover_selection
                                               (GtkTreeView *tree_view);
void        gtk_tree_view_set_hover_selection
```

```
                                                    (GtkTreeView *tree_view,
                                                     gboolean hover);
gboolean    gtk_tree_view_get_hover_expand  (GtkTreeView *tree_view);
void        gtk_tree_view_set_hover_expand  (GtkTreeView *tree_view,
                                             gboolean expand);
void        (*GtkTreeDestroyCountFunc)      (GtkTreeView *tree_view,
                                             GtkTreePath *path,
                                             gint children,
                                             gpointer user_data);
void        gtk_tree_view_set_destroy_count_func
                                            (GtkTreeView *tree_view,
                                             GtkTreeDestroyCountFunc func,
                                             gpointer data,
                                             GtkDestroyNotify destroy);
gboolean    (*GtkTreeViewRowSeparatorFunc)  (GtkTreeModel *model,
                                             GtkTreeIter *iter,
                                             gpointer data);
GtkTreeViewRowSeparatorFunc gtk_tree_view_get_row_separator_func
                                            (GtkTreeView *tree_view);
void        gtk_tree_view_set_row_separator_func
                                            (GtkTreeView *tree_view,
                                             GtkTreeViewRowSeparatorFunc func,
                                             gpointer data,
                                             GtkDestroyNotify destroy);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkWidget
               +----GtkContainer
                     +----GtkTreeView
```

Implemented Interfaces

GtkTreeView implements AtkImplementorIface.

Properties

```
  "enable-search"         gboolean              : Read / Write
  "expander-column"       GtkTreeViewColumn     : Read / Write
  "fixed-height-mode"     gboolean              : Read / Write
  "hadjustment"           GtkAdjustment         : Read / Write
  "headers-clickable"     gboolean              : Write
  "headers-visible"       gboolean              : Read / Write
  "hover-expand"          gboolean              : Read / Write
  "hover-selection"       gboolean              : Read / Write
  "model"                 GtkTreeModel          : Read / Write
  "reorderable"           gboolean              : Read / Write
  "rules-hint"            gboolean              : Read / Write
  "search-column"         gint                  : Read / Write
  "vadjustment"           GtkAdjustment         : Read / Write
```

Style Properties

```
  "allow-rules"           gboolean              : Read
  "even-row-color"        GdkColor              : Read
  "expander-size"         gint                  : Read
  "horizontal-separator"  gint                  : Read
  "indent-expanders"      gboolean              : Read
  "odd-row-color"         GdkColor              : Read
  "vertical-separator"    gint                  : Read
```

Signal Prototypes

```
"columns-changed"
            void        user_function      (GtkTreeView *treeview,
                                            gpointer user_data);
"cursor-changed"
            void        user_function      (GtkTreeView *treeview,
                                            gpointer user_data);
"expand-collapse-cursor-row"
            gboolean    user_function      (GtkTreeView *treeview,
                                            gboolean arg1,
                                            gboolean arg2,
                                            gboolean arg3,
```

```
                                        gpointer user_data);
"move-cursor"
            gboolean    user_function   (GtkTreeView *treeview,
                                         GtkMovementStep arg1,
                                         gint arg2,
                                         gpointer user_data);
"row-activated"
            void        user_function   (GtkTreeView *treeview,
                                         GtkTreePath *arg1,
                                         GtkTreeViewColumn *arg2,
                                         gpointer user_data);
"row-collapsed"
            void        user_function   (GtkTreeView *treeview,
                                         GtkTreeIter *arg1,
                                         GtkTreePath *arg2,
                                         gpointer user_data);
"row-expanded"
            void        user_function   (GtkTreeView *treeview,
                                         GtkTreeIter *arg1,
                                         GtkTreePath *arg2,
                                         gpointer user_data);
"select-all"
            gboolean    user_function   (GtkTreeView *treeview,
                                         gpointer user_data);
"select-cursor-parent"
            gboolean    user_function   (GtkTreeView *treeview,
                                         gpointer user_data);
"select-cursor-row"
            gboolean    user_function   (GtkTreeView *treeview,
                                         gboolean arg1,
                                         gpointer user_data);
"set-scroll-adjustments"
            void        user_function   (GtkTreeView *treeview,
                                         GtkAdjustment *arg1,
                                         GtkAdjustment *arg2,
                                         gpointer user_data);
"start-interactive-search"
            gboolean    user_function   (GtkTreeView *treeview,
                                         gpointer user_data);
"test-collapse-row"
            gboolean    user_function   (GtkTreeView *treeview,
                                         GtkTreeIter *arg1,
                                         GtkTreePath *arg2,
                                         gpointer user_data);
```

```
"test-expand-row"
            gboolean    user_function      (GtkTreeView *treeview,
                                            GtkTreeIter *arg1,
                                            GtkTreePath *arg2,
                                            gpointer user_data);
"toggle-cursor-row"
            gboolean    user_function      (GtkTreeView *treeview,
                                            gpointer user_data);
"unselect-all"
            gboolean    user_function      (GtkTreeView *treeview,
                                            gpointer user_data);
```

Description

Widget that displays any object that implements the GtkTreeModel interface.

Details

GtkTreeView

```
typedef struct _GtkTreeView GtkTreeView;
```

---

enum GtkTreeViewDropPosition

```
typedef enum
{
  /* drop before/after this row */
  GTK_TREE_VIEW_DROP_BEFORE,
  GTK_TREE_VIEW_DROP_AFTER,
  /* drop as a child of this row (with fallback to before or after
   * if into is not possible)
   */
  GTK_TREE_VIEW_DROP_INTO_OR_BEFORE,
  GTK_TREE_VIEW_DROP_INTO_OR_AFTER
} GtkTreeViewDropPosition;
```

An enum for determining where a dropped row goes.

GtkTreeViewPrivate

```
typedef struct _GtkTreeViewPrivate GtkTreeViewPrivate;
```

A private struct for internal use only. The definition of this structure is not publically available.

GtkTreeViewColumnDropFunc ()

```
gboolean    (*GtkTreeViewColumnDropFunc)    (GtkTreeView *tree_view,
                                             GtkTreeViewColumn *column,
                                             GtkTreeViewColumn *prev_column,
                                             GtkTreeViewColumn *next_column,
                                             gpointer data);
```

Function type for determining whether *column* can be dropped in a particular spot (as determined by *prev_column* and *next_column*). In left to right locales, *prev_column* is on the left of the potential drop spot, and *next_column* is on the right. In right to left mode, this is reversed. This function should return TRUE if the spot is a valid drop spot. Please note that returning TRUE does not actually indicate that the column drop was made, but is meant only to indicate a possible drop spot to the user.

tree_view :	A GtkTreeView
column :	The GtkTreeViewColumn being dragged
prev_column :	A GtkTreeViewColumn on one side of *column*
next_column :	A GtkTreeViewColumn on the other side of *column*
data :	user data
Returns :	TRUE, if column can be dropped in this spot

GtkTreeViewMappingFunc ()

```
void        (*GtkTreeViewMappingFunc)       (GtkTreeView *tree_view,
                                             GtkTreePath *path,
                                             gpointer user_data);
```

Function used for gtk_tree_view_map_expanded_rows.

tree_view : A GtkTreeView
path : The path that's expanded
user_data : user data

GtkTreeViewSearchEqualFunc ()

```
gboolean    (*GtkTreeViewSearchEqualFunc)   (GtkTreeModel *model,
                                             gint column,
                                             const gchar *key,
                                             GtkTreeIter *iter,
                                             gpointer search_data);
```

A function used for checking whether a row in *model* matches a search key string entered by the user. Note the return value is reversed from what you would normally expect, though it has some similarity to `strcmp()` returning 0 for equal strings.

model : the GtkTreeModel being searched
column : the search column set by `gtk_tree_view_set_search_column()`
key : the key string to compare with
iter : a GtkTreeIter pointing the row of *model* that should be compared with *key*.
search_data : user data from `gtk_tree_view_set_search_equal_func()`
Returns : FALSE if the row matches, TRUE otherwise.

---

gtk_tree_view_new ()

```
GtkWidget*  gtk_tree_view_new               (void);
```

Creates a new GtkTreeView widget.

Returns : A newly created GtkTreeView widget.

---

gtk_tree_view_new_with_model ()

```
GtkWidget*  gtk_tree_view_new_with_model    (GtkTreeModel *model);
```

Creates a new GtkTreeView widget with the model initialized to *model*.

model : the model.
Returns : A newly created GtkTreeView widget.

---

gtk_tree_view_get_model ()

```
GtkTreeModel* gtk_tree_view_get_model       (GtkTreeView *tree_view);
```

Returns the model the the GtkTreeView is based on. Returns NULL if the model is unset.

tree_view : a GtkTreeView
Returns : A GtkTreeModel, or NULL if none is currently being used.

---

gtk_tree_view_set_model ()

```
void        gtk_tree_view_set_model         (GtkTreeView *tree_view,
                                             GtkTreeModel *model);
```

Sets the model for a GtkTreeView. If the *tree_view* already has a model set, it will remove it before setting the new model. If *model* is NULL, then it will unset the old model.

tree_view : A GtkTreeNode.
model : The model.

---

gtk_tree_view_get_selection ()

```
GtkTreeSelection* gtk_tree_view_get_selection
                                            (GtkTreeView *tree_view);
```

Gets the GtkTreeSelection associated with *tree_view*.

tree_view : A GtkTreeView.
Returns : A GtkTreeSelection object.

gtk_tree_view_get_hadjustment ()

```
GtkAdjustment* gtk_tree_view_get_hadjustment
                                                (GtkTreeView *tree_view);
```

Gets the GtkAdjustment currently being used for the horizontal aspect.

tree_view : A GtkTreeView

Returns : A GtkAdjustment object, or NULL if none is currently being used.

gtk_tree_view_set_hadjustment ()

```
void        gtk_tree_view_set_hadjustment   (GtkTreeView *tree_view,
                                             GtkAdjustment *adjustment);
```

Sets the GtkAdjustment for the current horizontal aspect.

tree_view : A GtkTreeView

adjustment : The GtkAdjustment to set, or NULL

gtk_tree_view_get_vadjustment ()

```
GtkAdjustment* gtk_tree_view_get_vadjustment
                                                (GtkTreeView *tree_view);
```

Gets the GtkAdjustment currently being used for the vertical aspect.

tree_view : A GtkTreeView

Returns : A GtkAdjustment object, or NULL if none is currently being used.

gtk_tree_view_set_vadjustment ()

```
void        gtk_tree_view_set_vadjustment   (GtkTreeView *tree_view,
                                             GtkAdjustment *adjustment);
```

Sets the GtkAdjustment for the current vertical aspect.

tree_view: A GtkTreeView
adjustment: The GtkAdjustment to set, or NULL

---

gtk_tree_view_get_headers_visible ()

```
gboolean    gtk_tree_view_get_headers_visible
                                            (GtkTreeView *tree_view);
```

Returns TRUE if the headers on the *tree_view* are visible.

tree_view: A GtkTreeView.
Returns: Whether the headers are visible or not.

---

gtk_tree_view_set_headers_visible ()

```
void        gtk_tree_view_set_headers_visible
                                            (GtkTreeView *tree_view,
                                             gboolean headers_visible);
```

Sets the the visibility state of the headers.

tree_view: A GtkTreeView.
headers_visible: TRUE if the headers are visible

---

gtk_tree_view_columns_autosize ()

```
void        gtk_tree_view_columns_autosize  (GtkTreeView *tree_view);
```

Resizes all columns to their optimal width. Only works after the treeview has been realized.

tree_view : A GtkTreeView.

---

gtk_tree_view_set_headers_clickable ()

```
void        gtk_tree_view_set_headers_clickable
                                            (GtkTreeView *tree_view,
                                             gboolean setting);
```

Allow the column title buttons to be clicked.

tree_view : A GtkTreeView.

setting : TRUE if the columns are clickable.

---

gtk_tree_view_set_rules_hint ()

```
void        gtk_tree_view_set_rules_hint    (GtkTreeView *tree_view,
                                             gboolean setting);
```

This function tells GTK+ that the user interface for your application requires users to read across tree rows and associate cells with one another. By default, GTK+ will then render the tree with alternating row colors. Do *not* use it just because you prefer the appearance of the ruled tree; that's a question for the theme. Some themes will draw tree rows in alternating colors even when rules are turned off, and users who prefer that appearance all the time can choose those themes. You should call this function only as a *semantic* hint to the theme engine that your tree makes alternating colors useful from a functional standpoint (since it has lots of columns, generally).

tree_view : a GtkTreeView

setting : TRUE if the tree requires reading across rows

---

gtk_tree_view_get_rules_hint ()

```
gboolean    gtk_tree_view_get_rules_hint    (GtkTreeView *tree_view);
```

Gets the setting set by gtk_tree_view_set_rules_hint().

tree_view : a GtkTreeView

Returns : TRUE if rules are useful for the user of this tree

---

gtk_tree_view_append_column ()

```
gint        gtk_tree_view_append_column     (GtkTreeView *tree_view,
                                             GtkTreeViewColumn *column);
```

Appends *column* to the list of columns. If *tree_view* has "fixed_height" mode enabled, then *column* must have its "sizing" property set to be GTK_TREE_VIEW_COLUMN_FIXED.

tree_view : A GtkTreeView.

column : The GtkTreeViewColumn to add.

Returns : The number of columns in *tree_view* after appending.

---

gtk_tree_view_remove_column ()

```
gint        gtk_tree_view_remove_column     (GtkTreeView *tree_view,
                                             GtkTreeViewColumn *column);
```

Removes *column* from *tree_view*.

tree_view : A GtkTreeView.

column : The GtkTreeViewColumn to remove.

Returns : The number of columns in *tree_view* after removing.

---

gtk_tree_view_insert_column ()

```
gint        gtk_tree_view_insert_column     (GtkTreeView *tree_view,
                                             GtkTreeViewColumn *column,
                                             gint position);
```

This inserts the *column* into the *tree_view* at *position*. If *position* is -1, then the column is inserted at the end. If *tree_view* has "fixed_height" mode enabled, then *column* must have its "sizing" property set to be GTK_TREE_VIEW_COLUMN_FIXED.

tree_view : A GtkTreeView.
column : The GtkTreeViewColumn to be inserted.
position : The position to insert *column* in.
Returns : The number of columns in *tree_view* after insertion.

---

gtk_tree_view_insert_column_with_attributes ()

```
gint        gtk_tree_view_insert_column_with_attributes
                                            (GtkTreeView *tree_view,
                                             gint position,
                                             const gchar *title,
                                             GtkCellRenderer *cell,
                                             ...);
```

Creates a new GtkTreeViewColumn and inserts it into the *tree_view* at *position*. If *position* is -1, then the newly created column is inserted at the end. The column is initialized with the attributes given. If *tree_view* has "fixed_height" mode enabled, then *column* must have its sizing property set to be GTK_TREE_VIEW_COLUMN_FIXED.

tree_view : A GtkTreeView
position : The position to insert the new column in.
title : The title to set the header to.
cell : The GtkCellRenderer.
... : A NULL-terminated list of attributes.
Returns : The number of columns in *tree_view* after insertion.

---

gtk_tree_view_insert_column_with_data_func ()

```
gint        gtk_tree_view_insert_column_with_data_func
                                            (GtkTreeView *tree_view,
                                             gint position,
                                             const gchar *title,
                                             GtkCellRenderer *cell,
                                             GtkTreeCellDataFunc func,
                                             gpointer data,
                                             GDestroyNotify dnotify);
```

Convenience function that inserts a new column into the GtkTreeView with the given cell renderer and a GtkCellDataFunc to set cell renderer attributes (normally using data from the model). See also `gtk_tree_view_column_set_cell_data_func()`, `gtk_tree_view_column_pack_start()`. If *`tree_view`* has "fixed_height" mode enabled, then *`column`* must have its "sizing" property set to be GTK_TREE_VIEW_COLUMN_FIXED.

`tree_view`: a GtkTreeView
`position`: Position to insert, -1 for append
`title`: column title
`cell`: cell renderer for column
`func`: function to set attributes of cell renderer
`data`: data for *`func`*
`dnotify`: destroy notifier for *`data`*
Returns: number of columns in the tree view post-insert

---

gtk_tree_view_get_column ()

```
GtkTreeViewColumn* gtk_tree_view_get_column (GtkTreeView *tree_view,
                                             gint n);
```

Gets the GtkTreeViewColumn at the given position in the tree_view.

`tree_view`: A GtkTreeView.
`n`: The position of the column, counting from 0.
Returns: The GtkTreeViewColumn, or `NULL` if the position is outside the range of columns.

---

gtk_tree_view_get_columns ()

```
GList*      gtk_tree_view_get_columns       (GtkTreeView *tree_view);
```

Returns a GList of all the GtkTreeViewColumn s currently in *`tree_view`*. The returned list must be freed with `g_list_free()`.

`tree_view`: A GtkTreeView
Returns: A list of GtkTreeViewColumn s

---

gtk_tree_view_move_column_after ()

```
void        gtk_tree_view_move_column_after (GtkTreeView *tree_view,
                                             GtkTreeViewColumn *column,
                                             GtkTreeViewColumn *base_column);
```

Moves *column* to be after to *base_column*. If *base_column* is NULL, then *column* is placed in the first position.

tree_view : A GtkTreeView

column : The GtkTreeViewColumn to be moved.

base_column : The GtkTreeViewColumn to be moved relative to, or NULL.

---

gtk_tree_view_set_expander_column ()

```
void        gtk_tree_view_set_expander_column
                                            (GtkTreeView *tree_view,
                                             GtkTreeViewColumn *column);
```

Sets the column to draw the expander arrow at. It must be in *tree_view*. If *column* is NULL, then the expander arrow is always at the first visible column.

tree_view : A GtkTreeView

column : NULL, or the column to draw the expander arrow at.

---

gtk_tree_view_get_expander_column ()

```
GtkTreeViewColumn* gtk_tree_view_get_expander_column
                                            (GtkTreeView *tree_view);
```

Returns the column that is the current expander column. This column has the expander arrow drawn next to it.

tree_view : A GtkTreeView

Returns : The expander column.

---

gtk_tree_view_set_column_drag_function ()

```
void        gtk_tree_view_set_column_drag_function
                                            (GtkTreeView *tree_view,
                                             GtkTreeViewColumnDropFunc func,
                                             gpointer user_data,
                                             GtkDestroyNotify destroy);
```

Sets a user function for determining where a column may be dropped when dragged. This function is called on every column pair in turn at the beginning of a column drag to determine where a drop can take place. The arguments passed to *func* are: the *tree_view*, the GtkTreeViewColumn being dragged, the two GtkTreeViewColumn s determining the drop spot, and *user_data*. If either of the GtkTreeViewColumn arguments for the drop spot are NULL, then they indicate an edge. If *func* is set to be NULL, then *tree_view* reverts to the default behavior of allowing all columns to be dropped everywhere.

tree_view : A GtkTreeView.
func : A function to determine which columns are reorderable, or NULL.
user_data : User data to be passed to *func*, or NULL
destroy : Destroy notifier for *user_data*, or NULL

---

gtk_tree_view_scroll_to_point ()

```
void        gtk_tree_view_scroll_to_point   (GtkTreeView *tree_view,
                                             gint tree_x,
                                             gint tree_y);
```

Scrolls the tree view such that the top-left corner of the visible area is *tree_x*, *tree_y*, where *tree_x* and *tree_y* are specified in tree window coordinates. The *tree_view* must be realized before this function is called. If it isn't, you probably want to be using gtk_tree_view_scroll_to_cell().

If either *tree_x* or *tree_y* are -1, then that direction isn't scrolled.

tree_view : a GtkTreeView
tree_x : X coordinate of new top-left pixel of visible area, or -1
tree_y : Y coordinate of new top-left pixel of visible area, or -1

---

gtk_tree_view_scroll_to_cell ()

```
void        gtk_tree_view_scroll_to_cell    (GtkTreeView *tree_view,
                                             GtkTreePath *path,
                                             GtkTreeViewColumn *column,
                                             gboolean use_align,
                                             gfloat row_align,
                                             gfloat col_align);
```

Moves the alignments of *tree_view* to the position specified by *column* and *path*. If *column* is NULL, then no horizontal scrolling occurs. Likewise, if *path* is NULL no vertical scrolling occurs. At a minimum, one of *column* or *path* need to be non-NULL. *row_align* determines where the row is placed, and *col_align* determines where *column* is placed. Both are expected to be between 0.0 and 1.0. 0.0 means left/top alignment, 1.0 means right/bottom alignment, 0.5 means center.

If *use_align* is FALSE, then the alignment arguments are ignored, and the tree does the minimum amount of work to scroll the cell onto the screen. This means that the cell will be scrolled to the edge closest to its current position. If the cell is currently visible on the screen, nothing is done.

This function only works if the model is set, and *path* is a valid row on the model. If the model changes before the *tree_view* is realized, the centered path will be modified to reflect this change.

tree_view : A GtkTreeView.
path : The path of the row to move to, or NULL.
column : The GtkTreeViewColumn to move horizontally to, or NULL.
use_align : whether to use alignment arguments, or FALSE.
row_align : The vertical alignment of the row specified by *path*.
col_align : The horizontal alignment of the column specified by *column*.

---

gtk_tree_view_set_cursor ()

```
void        gtk_tree_view_set_cursor        (GtkTreeView *tree_view,
                                             GtkTreePath *path,
                                             GtkTreeViewColumn *focus_column,
                                             gboolean start_editing);
```

Sets the current keyboard focus to be at *path*, and selects it. This is useful when you want to focus the user's attention on a particular row. If *focus_column* is not NULL, then focus is given to the column specified by it. Additionally, if *focus_column* is specified, and *start_editing* is TRUE, then editing should be started in the specified cell. This function is often followed by *gtk_widget_grab_focus* (*tree_view*) in order to give keyboard focus to the widget. Please note that editing can only happen when the widget is realized.

tree_view: A GtkTreeView
path: A GtkTreePath
focus_column: A GtkTreeViewColumn, or NULL
start_editing: TRUE if the specified cell should start being edited.

---

gtk_tree_view_set_cursor_on_cell ()

```
void        gtk_tree_view_set_cursor_on_cell
                                            (GtkTreeView *tree_view,
                                             GtkTreePath *path,
                                             GtkTreeViewColumn *focus_column,
                                             GtkCellRenderer *focus_cell,
                                             gboolean start_editing);
```

Sets the current keyboard focus to be at *path*, and selects it. This is useful when you want to focus the user's attention on a particular row. If *focus_column* is not NULL, then focus is given to the column specified by it. If *focus_column* and *focus_cell* are not NULL, and *focus_column* contains 2 or more editable or activatable cells, then focus is given to the cell specified by *focus_cell*. Additionally, if *focus_column* is specified, and *start_editing* is TRUE, then editing should be started in the specified cell. This function is often followed by *gtk_widget_grab_focus* (*tree_view*) in order to give keyboard focus to the widget. Please note that editing can only happen when the widget is realized.

tree_view: A GtkTreeView
path: A GtkTreePath
focus_column: A GtkTreeViewColumn, or NULL
focus_cell: A GtkCellRenderer, or NULL
start_editing: TRUE if the specified cell should start being edited.

Since 2.2

---

gtk_tree_view_get_cursor ()

```
void        gtk_tree_view_get_cursor        (GtkTreeView *tree_view,
                                             GtkTreePath **path,
                                             GtkTreeViewColumn **focus_column);
```

Fills in *path* and *focus_column* with the current path and focus column. If the cursor isn't currently set, then

*`path` will be `NULL`. If no column currently has focus, then *`focus_column` will be `NULL`.

The returned GtkTreePath must be freed with `gtk_tree_path_free()` when you are done with it.

`tree_view` :	A GtkTreeView
`path` :	A pointer to be filled with the current cursor path, or `NULL`
`focus_column` :	A pointer to be filled with the current focus column, or `NULL`

---

gtk_tree_view_row_activated ()

```
void        gtk_tree_view_row_activated     (GtkTreeView *tree_view,
                                             GtkTreePath *path,
                                             GtkTreeViewColumn *column);
```

Activates the cell determined by `path` and `column`.

`tree_view` :	A GtkTreeView
`path` :	The GtkTreePath to be activated.
`column` :	The GtkTreeViewColumn to be activated.

---

gtk_tree_view_expand_all ()

```
void        gtk_tree_view_expand_all        (GtkTreeView *tree_view);
```

Recursively expands all nodes in the `tree_view`.

`tree_view` :	A GtkTreeView.

---

gtk_tree_view_collapse_all ()

```
void        gtk_tree_view_collapse_all      (GtkTreeView *tree_view);
```

Recursively collapses all visible, expanded nodes in `tree_view`.

tree_view : A GtkTreeView.

---

gtk_tree_view_expand_to_path ()

```
void        gtk_tree_view_expand_to_path    (GtkTreeView *tree_view,
                                             GtkTreePath *path);
```

Expands the row at *path*. This will also expand all parent rows of *path* as necessary.

tree_view : A GtkTreeView.
path : path to a row.

Since 2.2

---

gtk_tree_view_expand_row ()

```
gboolean    gtk_tree_view_expand_row        (GtkTreeView *tree_view,
                                             GtkTreePath *path,
                                             gboolean open_all);
```

Opens the row so its children are visible.

tree_view : a GtkTreeView
path : path to a row
open_all : whether to recursively expand, or just expand immediate children
Returns : TRUE if the row existed and had children

---

gtk_tree_view_collapse_row ()

```
gboolean    gtk_tree_view_collapse_row      (GtkTreeView *tree_view,
                                             GtkTreePath *path);
```

Collapses a row (hides its child rows, if they exist).

tree_view : a GtkTreeView
path : path to a row in the *tree_view*
Returns : TRUE if the row was collapsed.

---

gtk_tree_view_map_expanded_rows ()

```
void        gtk_tree_view_map_expanded_rows (GtkTreeView *tree_view,
                                             GtkTreeViewMappingFunc func,
                                             gpointer data);
```

Calls *func* on all expanded rows.

tree_view : A GtkTreeView
func : A function to be called
data : User data to be passed to the function.

---

gtk_tree_view_row_expanded ()

```
gboolean    gtk_tree_view_row_expanded      (GtkTreeView *tree_view,
                                             GtkTreePath *path);
```

Returns TRUE if the node pointed to by *path* is expanded in *tree_view*.

tree_view : A GtkTreeView.
path : A GtkTreePath to test expansion state.
Returns : TRUE if path is expanded.

---

gtk_tree_view_set_reorderable ()

```
void        gtk_tree_view_set_reorderable   (GtkTreeView *tree_view,
                                             gboolean reorderable);
```

This function is a convenience function to allow you to reorder models that support the GtkDragSourceIface and the GtkDragDestIface. Both GtkTreeStore and GtkListStore support these. If *reorderable* is TRUE, then the user can reorder the model by dragging and dropping rows. The developer can listen to these changes by connecting to the

model's row_inserted and row_deleted signals.

This function does not give you any degree of control over the order -- any reordering is allowed. If more control is needed, you should probably handle drag and drop manually.

tree_view : A GtkTreeView.
reorderable : TRUE, if the tree can be reordered.

---

gtk_tree_view_get_reorderable ()

```
gboolean    gtk_tree_view_get_reorderable   (GtkTreeView *tree_view);
```

Retrieves whether the user can reorder the tree via drag-and-drop. See gtk_tree_view_set_reorderable().

tree_view : a GtkTreeView
Returns : TRUE if the tree can be reordered.

---

gtk_tree_view_get_path_at_pos ()

```
gboolean    gtk_tree_view_get_path_at_pos   (GtkTreeView *tree_view,
                                             gint x,
                                             gint y,
                                             GtkTreePath **path,
                                             GtkTreeViewColumn **column,
                                             gint *cell_x,
                                             gint *cell_y);
```

Finds the path at the point (*x*, *y*), relative to widget coordinates. That is, *x* and *y* are relative to an events coordinates. *x* and *y* must come from an event on the *tree_view* only where event->window == gtk_tree_view_get_bin (). It is primarily for things like popup menus. If *path* is non-NULL, then it will be filled with the GtkTreePath at that point. This path should be freed with gtk_tree_path_free(). If *column* is non-NULL, then it will be filled with the column at that point. *cell_x* and *cell_y* return the coordinates relative to the cell background (i.e. the *background_area* passed to gtk_cell_renderer_render()). This function is only meaningful if *tree_view* is realized.

tree_view : A GtkTreeView.
x : The x position to be identified.
y : The y position to be identified.

path :	A pointer to a GtkTreePath pointer to be filled in, or NULL
column :	A pointer to a GtkTreeViewColumn pointer to be filled in, or NULL
cell_x :	A pointer where the X coordinate relative to the cell can be placed, or NULL
cell_y :	A pointer where the Y coordinate relative to the cell can be placed, or NULL
Returns :	TRUE if a row exists at that coordinate.

---

gtk_tree_view_get_cell_area ()

```
void        gtk_tree_view_get_cell_area     (GtkTreeView *tree_view,
                                             GtkTreePath *path,
                                             GtkTreeViewColumn *column,
                                             GdkRectangle *rect);
```

Fills the bounding rectangle in tree window coordinates for the cell at the row specified by *path* and the column specified by *column*. If *path* is NULL, or points to a path not currently displayed, the *y* and *height* fields of the rectangle will be filled with 0. If *column* is NULL, the *x* and *width* fields will be filled with 0. The sum of all cell rects does not cover the entire tree; there are extra pixels in between rows, for example. The returned rectangle is equivalent to the *cell_area* passed to gtk_cell_renderer_render(). This function is only valid if *tree_view* is realized.

tree_view :	a GtkTreeView
path :	a GtkTreePath for the row, or NULL to get only horizontal coordinates
column :	a GtkTreeViewColumn for the column, or NULL to get only vertical coordinates
rect :	rectangle to fill with cell rect

---

gtk_tree_view_get_background_area ()

```
void        gtk_tree_view_get_background_area
                                            (GtkTreeView *tree_view,
                                             GtkTreePath *path,
                                             GtkTreeViewColumn *column,
                                             GdkRectangle *rect);
```

Fills the bounding rectangle in tree window coordinates for the cell at the row specified by *path* and the column specified by *column*. If *path* is NULL, or points to a node not found in the tree, the *y* and *height* fields of the rectangle will be filled with 0. If *column* is NULL, the *x* and *width* fields will be filled with 0. The returned rectangle is equivalent to the *background_area* passed to gtk_cell_renderer_render(). These background areas tile to cover the entire tree window (except for the area used for header buttons). Contrast with the *cell_area*,

returned by gtk_tree_view_get_cell_area(), which returns only the cell itself, excluding surrounding borders and the tree expander area.

tree_view : a GtkTreeView
path : a GtkTreePath for the row, or NULL to get only horizontal coordinates
column : a GtkTreeViewColumn for the column, or NULL to get only vertical coordiantes
rect : rectangle to fill with cell background rect

---

gtk_tree_view_get_visible_rect ()

```
void        gtk_tree_view_get_visible_rect  (GtkTreeView *tree_view,
                                             GdkRectangle *visible_rect);
```

Fills *visible_rect* with the currently-visible region of the buffer, in tree coordinates. Convert to widget coordinates with gtk_tree_view_tree_to_widget_coords(). Tree coordinates start at 0,0 for row 0 of the tree, and cover the entire scrollable area of the tree.

tree_view : a GtkTreeView
visible_rect : rectangle to fill

---

gtk_tree_view_get_bin_window ()

```
GdkWindow*  gtk_tree_view_get_bin_window    (GtkTreeView *tree_view);
```

Returns the window that *tree_view* renders to. This is used primarily to compare to event->window to confirm that the event on *tree_view* is on the right window.

tree_view : A GtkTreeView
Returns : A GdkWindow, or NULL when *tree_view* hasn't been realized yet

---

gtk_tree_view_widget_to_tree_coords ()

```
void        gtk_tree_view_widget_to_tree_coords
                                            (GtkTreeView *tree_view,
                                             gint wx,
```

```
                                                gint wy,
                                                gint *tx,
                                                gint *ty);
```

Converts widget coordinates to coordinates for the tree window (the full scrollable area of the tree).

tree_view : a GtkTreeView
wx : widget X coordinate
wy : widget Y coordinate
tx : return location for tree X coordinate
ty : return location for tree Y coordinate

---

gtk_tree_view_tree_to_widget_coords ()

```
void        gtk_tree_view_tree_to_widget_coords
                                               (GtkTreeView *tree_view,
                                                gint tx,
                                                gint ty,
                                                gint *wx,
                                                gint *wy);
```

Converts tree coordinates (coordinates in full scrollable area of the tree) to widget coordinates.

tree_view : a GtkTreeView
tx : tree X coordinate
ty : tree Y coordinate
wx : return location for widget X coordinate
wy : return location for widget Y coordinate

---

gtk_tree_view_enable_model_drag_dest ()

```
void        gtk_tree_view_enable_model_drag_dest
                                               (GtkTreeView *tree_view,
                                                const GtkTargetEntry *targets,
                                                gint n_targets,
                                                GdkDragAction actions);
```

Turns *tree_view* into a drop destination for automatic DND.

tree_view : a GtkTreeView
targets : the table of targets that the drag will support
n_targets : the number of items in *targets*
actions : the bitmask of possible actions for a drag from this widget

---

gtk_tree_view_enable_model_drag_source ()

```
void        gtk_tree_view_enable_model_drag_source
                                            (GtkTreeView *tree_view,
                                             GdkModifierType start_button_mask,
                                             const GtkTargetEntry *targets,
                                             gint n_targets,
                                             GdkDragAction actions);
```

Turns *tree_view* into a drag source for automatic DND.

tree_view : a GtkTreeView
start_button_mask : Mask of allowed buttons to start drag
targets : the table of targets that the drag will support
n_targets : the number of items in *targets*
actions : the bitmask of possible actions for a drag from this widget

---

gtk_tree_view_unset_rows_drag_source ()

```
void        gtk_tree_view_unset_rows_drag_source
                                            (GtkTreeView *tree_view);
```

Undoes the effect of gtk_tree_view_enable_model_drag_source().

tree_view : a GtkTreeView

---

gtk_tree_view_unset_rows_drag_dest ()

```
void        gtk_tree_view_unset_rows_drag_dest
                                            (GtkTreeView *tree_view);
```

Undoes the effect of gtk_tree_view_enable_model_drag_dest().

tree_view : a GtkTreeView

---

gtk_tree_view_set_drag_dest_row ()

```
void        gtk_tree_view_set_drag_dest_row (GtkTreeView *tree_view,
                                             GtkTreePath *path,
                                             GtkTreeViewDropPosition pos);
```

Sets the row that is highlighted for feedback.

tree_view : a GtkTreeView
path : The path of the row to highlight, or NULL.
pos : Specifies whether to drop before, after or into the row

---

gtk_tree_view_get_drag_dest_row ()

```
void        gtk_tree_view_get_drag_dest_row (GtkTreeView *tree_view,
                                             GtkTreePath **path,
                                             GtkTreeViewDropPosition *pos);
```

Gets information about the row that is highlighted for feedback.

tree_view : a GtkTreeView
path : Return location for the path of the highlighted row, or NULL.
pos : Return location for the drop position, or NULL

---

gtk_tree_view_get_dest_row_at_pos ()

```
gboolean    gtk_tree_view_get_dest_row_at_pos
                                            (GtkTreeView *tree_view,
```

```
                                                gint drag_x,
                                                gint drag_y,
                                                GtkTreePath **path,
                                                GtkTreeViewDropPosition *pos);
```

Determines the destination row for a given position.

tree_view : a GtkTreeView
drag_x : the position to determine the destination row for
drag_y : the position to determine the destination row for
path : Return location for the path of the highlighted row, or NULL.
pos : Return location for the drop position, or NULL
Returns : whether there is a row at the given postiion,

---

gtk_tree_view_create_row_drag_icon ()

```
GdkPixmap*  gtk_tree_view_create_row_drag_icon
                                               (GtkTreeView *tree_view,
                                                GtkTreePath *path);
```

Creates a GdkPixmap representation of the row at *path*. This image is used for a drag icon.

tree_view : a GtkTreeView
path : a GtkTreePath in *tree_view*
Returns : a newly-allocated pixmap of the drag icon.

---

gtk_tree_view_set_enable_search ()

```
void        gtk_tree_view_set_enable_search (GtkTreeView *tree_view,
                                                gboolean enable_search);
```

If *enable_search* is set, then the user can type in text to search through the tree interactively.

tree_view : A GtkTreeView
enable_search : TRUE, if the user can search interactively

---

gtk_tree_view_get_enable_search ()

```
gboolean    gtk_tree_view_get_enable_search (GtkTreeView *tree_view);
```

Returns whether or not the tree allows interactive searching.

tree_view : A GtkTreeView

Returns : whether or not to let the user search interactively

---

gtk_tree_view_get_search_column ()

```
gint        gtk_tree_view_get_search_column (GtkTreeView *tree_view);
```

Gets the column searched on by the interactive search code.

tree_view : A GtkTreeView

Returns : the column the interactive search code searches in.

---

gtk_tree_view_set_search_column ()

```
void        gtk_tree_view_set_search_column (GtkTreeView *tree_view,
                                             gint column);
```

Sets *column* as the column where the interactive search code should search in. Additionally, turns on interactive searching. Note that *column* refers to a column of the model.

tree_view : A GtkTreeView

column : the column of the model to search in

---

gtk_tree_view_get_search_equal_func ()

```
GtkTreeViewSearchEqualFunc gtk_tree_view_get_search_equal_func
                                             (GtkTreeView *tree_view);
```

Returns the compare function currently in use.

tree_view : A GtkTreeView
Returns : the currently used compare function for the search code.

---

gtk_tree_view_set_search_equal_func ()

```
void        gtk_tree_view_set_search_equal_func
                                            (GtkTreeView *tree_view,
                                             GtkTreeViewSearchEqualFunc
search_equal_func,
                                             gpointer search_user_data,
                                             GtkDestroyNotify search_destroy);
```

Sets the compare function for the interactive search capabilities; note that somewhat like `strcmp()` returning 0 for equality GtkTreeViewSearchEqualFunc returns FALSE on matches.

tree_view : A GtkTreeView
search_equal_func : the compare function to use during the search
search_user_data : user data to pass to *search_equal_func*, or NULL
search_destroy : Destroy notifier for *search_user_data*, or NULL

---

gtk_tree_view_get_fixed_height_mode ()

```
gboolean    gtk_tree_view_get_fixed_height_mode
                                            (GtkTreeView *tree_view);
```

Returns whether fixed height mode is turned on for *tree_view*.

tree_view : a GtkTreeView
Returns : TRUE if *tree_view* is in fixed height mode

Since 2.6

---

gtk_tree_view_set_fixed_height_mode ()

```
void        gtk_tree_view_set_fixed_height_mode
                                            (GtkTreeView *tree_view,
                                             gboolean enable);
```

Enables or disables the fixed height mode of *tree_view*. Fixed height mode speeds up GtkTreeView by assuming that all rows have the same height. Only enable this option if all rows are the same height and all columns are of type GTK_TREE_VIEW_COLUMN_FIXED.

tree_view : a GtkTreeView

enable : TRUE to enable fixed height mode

Since 2.6

---

gtk_tree_view_get_hover_selection ()

```
gboolean    gtk_tree_view_get_hover_selection
                                            (GtkTreeView *tree_view);
```

Returns whether hover selection mode is turned on for *tree_view*.

tree_view : a GtkTreeView

Returns : TRUE if *tree_view* is in hover selection mode

Since 2.6

---

gtk_tree_view_set_hover_selection ()

```
void        gtk_tree_view_set_hover_selection
                                            (GtkTreeView *tree_view,
                                             gboolean hover);
```

Enables of disables the hover selection mode of *tree_view*. Hover selection makes the selected row follow the pointer. Currently, this works only for the selection modes GTK_SELECTION_SINGLE and

GTK_SELECTION_BROWSE.

tree_view : a GtkTreeView
hover : TRUE to enable hover selection mode

Since 2.6

---

gtk_tree_view_get_hover_expand ()

```
gboolean    gtk_tree_view_get_hover_expand  (GtkTreeView *tree_view);
```

Returns whether hover expansion mode is turned on for *tree_view*.

tree_view : a GtkTreeView
Returns : TRUE if *tree_view* is in hover expansion mode

Since 2.6

---

gtk_tree_view_set_hover_expand ()

```
void        gtk_tree_view_set_hover_expand  (GtkTreeView *tree_view,
                                             gboolean expand);
```

Enables of disables the hover expansion mode of *tree_view*. Hover expansion makes rows expand or collaps if the pointer moves over them.

tree_view : a GtkTreeView
expand : TRUE to enable hover selection mode

Since 2.6

GtkTreeDestroyCountFunc ()

```
void        (*GtkTreeDestroyCountFunc)      (GtkTreeView *tree_view,
                                             GtkTreePath *path,
                                             gint children,
                                             gpointer user_data);
```

tree_view:
path:
children:
user_data:

---

gtk_tree_view_set_destroy_count_func ()

```
void        gtk_tree_view_set_destroy_count_func
                                            (GtkTreeView *tree_view,
                                             GtkTreeDestroyCountFunc func,
                                             gpointer data,
                                             GtkDestroyNotify destroy);
```

This function should almost never be used. It is meant for private use by ATK for determining the number of visible children that are removed when the user collapses a row, or a row is deleted.

tree_view: A GtkTreeView
func: Function to be called when a view row is destroyed, or NULL
data: User data to be passed to *func*, or NULL
destroy: Destroy notifier for *data*, or NULL

GtkTreeViewRowSeparatorFunc ()

```
gboolean    (*GtkTreeViewRowSeparatorFunc)  (GtkTreeModel *model,
                                             GtkTreeIter *iter,
                                             gpointer data);
```

Function type for determining whether the row pointed to by *iter* should be rendered as a separator. A common way to implement this is to have a boolean column in the model, whose values the GtkTreeViewRowSeparatorFunc returns.

model: the GtkTreeModel

iter : a GtkTreeIter pointing at a row in *model*

data : user data

Returns : TRUE if the row is a separator

---

gtk_tree_view_get_row_separator_func ()

```
GtkTreeViewRowSeparatorFunc gtk_tree_view_get_row_separator_func
                                                (GtkTreeView *tree_view);
```

Returns the current row separator function.

tree_view : a GtkTreeView

Returns : the current row separator function.

Since 2.6

---

gtk_tree_view_set_row_separator_func ()

```
void        gtk_tree_view_set_row_separator_func
                                               (GtkTreeView *tree_view,
                                                GtkTreeViewRowSeparatorFunc func,
                                                gpointer data,
                                                GtkDestroyNotify destroy);
```

Sets the row separator function, which is used to determine whether a row should be drawn as a separator. If the row separator function is NULL, no separators are drawn. This is the default value.

tree_view : a GtkTreeView

func : a GtkTreeViewRowSeparatorFunc

data : user data to pass to *func*, or NULL

destroy : destroy notifier for *data*, or NULL

Since 2.6

Properties

The "enable-search" property

```
"enable-search"        gboolean              : Read / Write
```

View allows user to search through columns interactively.

Default value: TRUE

The "expander-column" property

```
"expander-column"      GtkTreeViewColumn     : Read / Write
```

Set the column for the expander column.

The "fixed-height-mode" property

```
"fixed-height-mode"    gboolean              : Read / Write
```

Setting the ::fixed-height-mode property to TRUE speeds up GtkTreeView by assuming that all rows have the same height. Only enable this option if all rows are the same height. Please see `gtk_tree_view_set_fixed_height_mode()` for more information on this option.

Default value: FALSE

Since 2.4

The "hadjustment" property

```
"hadjustment"          GtkAdjustment         : Read / Write
```

Horizontal Adjustment for the widget.

The "headers-clickable" property

"headers-clickable"	gboolean	: Write

Column headers respond to click events.

Default value: FALSE

The "headers-visible" property

"headers-visible"	gboolean	: Read / Write

Show the column header buttons.

Default value: TRUE

The "hover-expand" property

"hover-expand"	gboolean	: Read / Write

Enables of disables the hover expansion mode of *tree_view*. Hover expansion makes rows expand or collaps if the pointer moves over them.

This mode is primarily indended for treeviews in popups, e.g. in GtkComboBox or GtkEntryCompletion.

Default value: FALSE

Since 2.6

The "hover-selection" property

"hover-selection"	gboolean	: Read / Write

Enables of disables the hover selection mode of *tree_view*. Hover selection makes the selected row follow the pointer. Currently, this works only for the selection modes GTK_SELECTION_SINGLE and GTK_SELECTION_BROWSE.

This mode is primarily indended for treeviews in popups, e.g. in GtkComboBox or GtkEntryCompletion.

Default value: FALSE

Since 2.6

The "model" property

"model"	GtkTreeModel	: Read / Write

The model for the tree view.

The "reorderable" property

"reorderable"	gboolean	: Read / Write

View is reorderable.

Default value: FALSE

The "rules-hint" property

"rules-hint"	gboolean	: Read / Write

Set a hint to the theme engine to draw rows in alternating colors.

Default value: FALSE

The "search-column" property

```
"search-column"        gint                : Read / Write
```

Model column to search through when searching through code.

Allowed values: >= -1

Default value: -1

The "vadjustment" property

```
"vadjustment"          GtkAdjustment       : Read / Write
```

Vertical Adjustment for the widget.

Style Properties

The "allow-rules" style property

```
"allow-rules"          gboolean            : Read
```

Allow drawing of alternating color rows.

Default value: TRUE

The "even-row-color" style property

```
"even-row-color"       GdkColor            : Read
```

Color to use for even rows.

The "expander-size" style property

```
"expander-size"        gint                : Read
```

Size of the expander arrow.

Allowed values: >= 0

Default value: 12

The "horizontal-separator" style property

```
"horizontal-separator" gint                : Read
```

Horizontal space between cells. Must be an even number.

Allowed values: >= 0

Default value: 2

The "indent-expanders" style property

```
"indent-expanders"     gboolean            : Read
```

Make the expanders indented.

Default value: TRUE

The "odd-row-color" style property

```
"odd-row-color"        GdkColor            : Read
```

Color to use for odd rows.

The "vertical-separator" style property

```
  "vertical-separator"   gint                : Read
```

Vertical space between cells. Must be an even number.

Allowed values: >= 0

Default value: 2

Signals

The "columns-changed" signal

```
void        user_function                  (GtkTreeView *treeview,
                                            gpointer user_data);
```

treeview : the object which received the signal.
user_data : user data set when the signal handler was connected.

The "cursor-changed" signal

```
void        user_function                  (GtkTreeView *treeview,
                                            gpointer user_data);
```

treeview : the object which received the signal.
user_data : user data set when the signal handler was connected.

The "expand-collapse-cursor-row" signal

```
gboolean    user_function                  (GtkTreeView *treeview,
                                            gboolean arg1,
                                            gboolean arg2,
                                            gboolean arg3,
                                            gpointer user_data);
```

treeview : the object which received the signal.
arg1 :
arg2 :
arg3 :
user_data : user data set when the signal handler was connected.
Returns :

The "move-cursor" signal

```
gboolean    user_function                   (GtkTreeView *treeview,
                                             GtkMovementStep arg1,
                                             gint arg2,
                                             gpointer user_data);
```

treeview : the object which received the signal.
arg1 :
arg2 :
user_data : user data set when the signal handler was connected.
Returns :

The "row-activated" signal

```
void        user_function                   (GtkTreeView *treeview,
                                             GtkTreePath *arg1,
                                             GtkTreeViewColumn *arg2,
                                             gpointer user_data);
```

treeview : the object which received the signal.
arg1 :
arg2 :
user_data : user data set when the signal handler was connected.

The "row-collapsed" signal

```
void        user_function                   (GtkTreeView *treeview,
                                             GtkTreeIter *arg1,
                                             GtkTreePath *arg2,
                                             gpointer user_data);
```

treeview : the object which received the signal.
arg1 :
arg2 :
user_data : user data set when the signal handler was connected.

The "row-expanded" signal

```
void        user_function                   (GtkTreeView *treeview,
                                             GtkTreeIter *arg1,
                                             GtkTreePath *arg2,
                                             gpointer user_data);
```

treeview : the object which received the signal.
arg1 :
arg2 :
user_data : user data set when the signal handler was connected.

The "select-all" signal

```
gboolean    user_function                   (GtkTreeView *treeview,
                                             gpointer user_data);
```

treeview : the object which received the signal.
user_data : user data set when the signal handler was connected.
Returns :

The "select-cursor-parent" signal

```
gboolean    user_function                   (GtkTreeView *treeview,
```

```
                                                gpointer user_data);
```

treeview : the object which received the signal.
user_data : user data set when the signal handler was connected.
Returns :

The "select-cursor-row" signal

```
gboolean    user_function                   (GtkTreeView *treeview,
                                             gboolean arg1,
                                             gpointer user_data);
```

treeview : the object which received the signal.
arg1 :
user_data : user data set when the signal handler was connected.
Returns :

The "set-scroll-adjustments" signal

```
void        user_function                   (GtkTreeView *treeview,
                                             GtkAdjustment *arg1,
                                             GtkAdjustment *arg2,
                                             gpointer user_data);
```

treeview : the object which received the signal.
arg1 :
arg2 :
user_data : user data set when the signal handler was connected.

The "start-interactive-search" signal

```
gboolean    user_function                   (GtkTreeView *treeview,
                                             gpointer user_data);
```

treeview : the object which received the signal.
user_data : user data set when the signal handler was connected.
Returns :

The "test-collapse-row" signal

```
gboolean    user_function                   (GtkTreeView *treeview,
                                             GtkTreeIter *arg1,
                                             GtkTreePath *arg2,
                                             gpointer user_data);
```

treeview : the object which received the signal.
arg1 :
arg2 :
user_data : user data set when the signal handler was connected.
Returns :

The "test-expand-row" signal

```
gboolean    user_function                   (GtkTreeView *treeview,
                                             GtkTreeIter *arg1,
                                             GtkTreePath *arg2,
                                             gpointer user_data);
```

treeview : the object which received the signal.
arg1 :
arg2 :
user_data : user data set when the signal handler was connected.
Returns :

The "toggle-cursor-row" signal

```
gboolean    user_function                   (GtkTreeView *treeview,
                                             gpointer user_data);
```

treeview : the object which received the signal.
user_data : user data set when the signal handler was connected.
Returns :

The "unselect-all" signal

```
gboolean    user_function                   (GtkTreeView *treeview,
                                             gpointer user_data);
```

treeview : the object which received the signal.
user_data : user data set when the signal handler was connected.
Returns :

See Also

GtkTreeViewColumn, GtkTreeSelection, GtkTreeDnd, GtkTreeMode, GtkTreeSortable, GtkTreeModelSort, GtkListStore, GtkTreeStore, GtkCellRenderer, GtkCellEditable, GtkCellRendererPixbuf, GtkCellRendererText, GtkCellRendererToggle

<< GtkTreeViewColumn **GtkTreeView drag-and-drop >>**

GtkTreeView drag-and-drop

GtkTreeView drag-and-drop — Interfaces for drag-and-drop support in GtkTreeView

Synopsis

```
#include <gtk/gtk.h>

            GtkTreeDragSource;
            GtkTreeDragSourceIface;
gboolean    gtk_tree_drag_source_drag_data_delete
                                        (GtkTreeDragSource *drag_source,
                                         GtkTreePath *path);
gboolean    gtk_tree_drag_source_drag_data_get
                                        (GtkTreeDragSource *drag_source,
                                         GtkTreePath *path,
                                         GtkSelectionData *selection_data);
gboolean    gtk_tree_drag_source_row_draggable
                                        (GtkTreeDragSource *drag_source,
                                         GtkTreePath *path);
            GtkTreeDragDest;
            GtkTreeDragDestIface;
gboolean    gtk_tree_drag_dest_drag_data_received
                                        (GtkTreeDragDest *drag_dest,
                                         GtkTreePath *dest,
                                         GtkSelectionData *selection_data);
gboolean    gtk_tree_drag_dest_row_drop_possible
                                        (GtkTreeDragDest *drag_dest,
                                         GtkTreePath *dest_path,
                                         GtkSelectionData *selection_data);
gboolean    gtk_tree_set_row_drag_data      (GtkSelectionData *selection_data,
                                         GtkTreeModel *tree_model,
                                         GtkTreePath *path);
gboolean    gtk_tree_get_row_drag_data      (GtkSelectionData *selection_data,
                                         GtkTreeModel **tree_model,
                                         GtkTreePath **path);
```

Object Hierarchy

```
GInterface
 +----GtkTreeDragSource
```

```
GInterface
 +----GtkTreeDragDest
```

Known Implementations

GtkTreeDragSource is implemented by GtkTreeModelSort, GtkTreeStore, GtkListStore and GtkTreeModelFilter.

GtkTreeDragDest is implemented by GtkTreeStore and GtkListStore.

Description

GTK+ supports Drag-and-Drop in tree views with a high-level and a low-level API.

The low-level API consists of the GTK+ DND API, augmented by some treeview utility functions: `gtk_tree_view_set_drag_dest_row()`, `gtk_tree_view_get_drag_dest_row()`, `gtk_tree_view_get_dest_row_at_pos()`, `gtk_tree_view_create_row_drag_icon()`, `gtk_tree_set_row_drag_data()` and `gtk_tree_get_row_drag_data()`. This API leaves a lot of flexibility, but nothing is done automatically, and implementing advanced features like hover-to-open-rows or autoscrolling on top of this API is a lot of work.

On the other hand, if you write to the high-level API, then all the bookkeeping of rows is done for you, as well as things like hover-to-open and auto-scroll, but your models have to implement the GtkTreeDragSource and GtkTreeDragDest interfaces.

Details

GtkTreeDragSource

```
typedef struct _GtkTreeDragSource GtkTreeDragSource;
```

GtkTreeDragSourceIface

```
typedef struct {
  GTypeInterface g_iface;

  /* VTable - not signals */

  gboolean     (* row_draggable)        (GtkTreeDragSource   *drag_source,
                                         GtkTreePath         *path);

  gboolean     (* drag_data_get)        (GtkTreeDragSource   *drag_source,
                                         GtkTreePath         *path,
                                         GtkSelectionData    *selection_data);

  gboolean     (* drag_data_delete)     (GtkTreeDragSource *drag_source,
                                         GtkTreePath       *path);
} GtkTreeDragSourceIface;
```

---

gtk_tree_drag_source_drag_data_delete ()

```
gboolean    gtk_tree_drag_source_drag_data_delete
                                        (GtkTreeDragSource *drag_source,
                                         GtkTreePath *path);
```

Asks the GtkTreeDragSource to delete the row at *path*, because it was moved somewhere else via drag-and-drop. Returns FALSE if the deletion fails because *path* no longer exists, or for some model-specific reason. Should robustly handle a *path* no longer found in the model!

drag_source :	a GtkTreeDragSource
path :	row that was being dragged
Returns :	TRUE if the row was successfully deleted

---

gtk_tree_drag_source_drag_data_get ()

```
gboolean    gtk_tree_drag_source_drag_data_get
                                        (GtkTreeDragSource *drag_source,
                                         GtkTreePath *path,
                                         GtkSelectionData *selection_data);
```

Asks the GtkTreeDragSource to fill in *selection_data* with a representation of the row at *path*. *selection_data->target* gives the required type of the data. Should robustly handle a *path* no longer found in the model!

drag_source :	a GtkTreeDragSource
path :	row that was dragged
selection_data :	a GtkSelectionData to fill with data from the dragged row
Returns :	TRUE if data of the required type was provided

---

gtk_tree_drag_source_row_draggable ()

```
gboolean    gtk_tree_drag_source_row_draggable
                                            (GtkTreeDragSource *drag_source,
                                             GtkTreePath *path);
```

Asks the GtkTreeDragSource whether a particular row can be used as the source of a DND operation. If the source doesn't implement this interface, the row is assumed draggable.

drag_source :	a GtkTreeDragSource
path :	row on which user is initiating a drag
Returns :	TRUE if the row can be dragged

GtkTreeDragDest

```
typedef struct _GtkTreeDragDest GtkTreeDragDest;
```

GtkTreeDragDestIface

```
typedef struct {
  GTypeInterface g_iface;

  /* VTable - not signals */

  gboolean     (* drag_data_received) (GtkTreeDragDest   *drag_dest,
                                       GtkTreePath       *dest,
```

```
                                        GtkSelectionData  *selection_data);

  gboolean     (* row_drop_possible)  (GtkTreeDragDest   *drag_dest,
                                        GtkTreePath       *dest_path,
                                        GtkSelectionData  *selection_data);
} GtkTreeDragDestIface;
```

---

gtk_tree_drag_dest_drag_data_received ()

```
gboolean    gtk_tree_drag_dest_drag_data_received
                                        (GtkTreeDragDest *drag_dest,
                                         GtkTreePath *dest,
                                         GtkSelectionData *selection_data);
```

Asks the GtkTreeDragDest to insert a row before the path *dest*, deriving the contents of the row from *selection_data*. If *dest* is outside the tree so that inserting before it is impossible, FALSE will be returned. Also, FALSE may be returned if the new row is not created for some model-specific reason. Should robustly handle a *dest* no longer found in the model!

drag_dest :	a GtkTreeDragDest
dest :	row to drop in front of
selection_data :	data to drop
Returns :	whether a new row was created before position *dest*

---

gtk_tree_drag_dest_row_drop_possible ()

```
gboolean    gtk_tree_drag_dest_row_drop_possible
                                        (GtkTreeDragDest *drag_dest,
                                         GtkTreePath *dest_path,
                                         GtkSelectionData *selection_data);
```

Determines whether a drop is possible before the given *dest_path*, at the same depth as *dest_path*. i.e., can we drop the data in *selection_data* at that location. *dest_path* does not have to exist; the return value will almost certainly be FALSE if the parent of *dest_path* doesn't exist, though.

drag_dest :	a GtkTreeDragDest
dest_path :	destination row

selection_data : the data being dragged

Returns : TRUE if a drop is possible before *dest_path*

---

gtk_tree_set_row_drag_data ()

```
gboolean    gtk_tree_set_row_drag_data      (GtkSelectionData *selection_data,
                                             GtkTreeModel *tree_model,
                                             GtkTreePath *path);
```

Sets selection data of target type GTK_TREE_MODEL_ROW. Normally used in a drag_data_get handler.

selection_data : some GtkSelectionData

tree_model : a GtkTreeModel

path : a row in *tree_model*

Returns : TRUE if the GtkSelectionData had the proper target type to allow us to set a tree row

---

gtk_tree_get_row_drag_data ()

```
gboolean    gtk_tree_get_row_drag_data      (GtkSelectionData *selection_data,
                                             GtkTreeModel **tree_model,
                                             GtkTreePath **path);
```

Obtains a *tree_model* and *path* from selection data of target type GTK_TREE_MODEL_ROW. Normally called from a drag_data_received handler. This function can only be used if *selection_data* originates from the same process that's calling this function, because a pointer to the tree model is being passed around. If you aren't in the same process, then you'll get memory corruption. In the GtkTreeDragDest drag_data_received handler, you can assume that selection data of type GTK_TREE_MODEL_ROW is in from the current process. The returned path must be freed with gtk_tree_path_free().

selection_data : a GtkSelectionData

tree_model : a GtkTreeModel

path : row in *tree_model*

Returns : TRUE if *selection_data* had target type GTK_TREE_MODEL_ROW and is otherwise valid

<< GtkTreeView **GtkCellView >>**

GtkCellView

GtkCellView — A widget displaying a single row of a GtkTreeModel

Synopsis

```
#include <gtk/gtk.h>

            GtkCellView;
GtkWidget*  gtk_cell_view_new               (void);
GtkWidget*  gtk_cell_view_new_with_text     (const gchar *text);
GtkWidget*  gtk_cell_view_new_with_markup   (const gchar *markup);
GtkWidget*  gtk_cell_view_new_with_pixbuf   (GdkPixbuf *pixbuf);
void        gtk_cell_view_set_value         (GtkCellView *cell_view,
                                             GtkCellRenderer *renderer,
                                             gchar *property,
                                             GValue *value);
void        gtk_cell_view_set_values        (GtkCellView *cell_view,
                                             GtkCellRenderer *renderer,
                                             ...);
void        gtk_cell_view_set_model         (GtkCellView *cell_view,
                                             GtkTreeModel *model);
void        gtk_cell_view_set_displayed_row (GtkCellView *cell_view,
                                             GtkTreePath *path);
GtkTreePath* gtk_cell_view_get_displayed_row
                                            (GtkCellView *cell_view);
gboolean    gtk_cell_view_get_size_of_row   (GtkCellView *cell_view,
                                             GtkTreePath *path,
                                             GtkRequisition *requisition);
void        gtk_cell_view_set_background_color
                                            (GtkCellView *cell_view,
                                             const GdkColor *color);
void        gtk_cell_view_set_cell_data     (GtkCellView *cell_view);
GList*      gtk_cell_view_get_cell_renderers
```

```
                                                    (GtkCellView *cell_view);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkWidget
               +----GtkCellView
```

Implemented Interfaces

GtkCellView implements AtkImplementorIface and GtkCellLayout.

Properties

```
  "background"           gchararray            : Write
  "background-gdk"       GdkColor              : Read / Write
  "background-set"       gboolean              : Read / Write
```

Description

Details

GtkCellView

```
typedef struct _GtkCellView GtkCellView;
```

---

gtk_cell_view_new ()

```
GtkWidget*  gtk_cell_view_new                (void);
```

Creates a new GtkCellView widget.

Returns : A newly created GtkCellView widget.

Since 2.6

---

gtk_cell_view_new_with_text ()

```
GtkWidget*  gtk_cell_view_new_with_text      (const gchar *text);
```

Creates a new GtkCellView widget, adds a GtkCellRendererText to it, and makes its show *text*.

text : the text to display in the cell view
Returns : A newly created GtkCellView widget.

Since 2.6

---

gtk_cell_view_new_with_markup ()

```
GtkWidget*  gtk_cell_view_new_with_markup    (const gchar *markup);
```

Creates a new GtkCellView widget, adds a GtkCellRendererText to it, and makes its show *markup*. The text can text can be marked up with the Pango text markup language.

markup : the text to display in the cell view
Returns : A newly created GtkCellView widget.

Since 2.6

---

gtk_cell_view_new_with_pixbuf ()

```
GtkWidget*  gtk_cell_view_new_with_pixbuf   (GdkPixbuf *pixbuf);
```

Creates a new GtkCellView widget, adds a GtkCellRendererPixbuf to it, and makes its show *pixbuf*.

pixbuf : the image to display in the cell view
Returns : A newly created GtkCellView widget.

Since 2.6

---

gtk_cell_view_set_value ()

```
void        gtk_cell_view_set_value         (GtkCellView *cell_view,
                                             GtkCellRenderer *renderer,
                                             gchar *property,
                                             GValue *value);
```

Sets a property of a cell renderer of *cell_view*, and makes sure the display of *cell_view* is updated.

cell_view : a GtkCellView widget
renderer : one of the renderers of *cell_view*
property : the name of the property of *renderer* to set
value : the new value to set the property to

Since 2.6

---

gtk_cell_view_set_values ()

```
void        gtk_cell_view_set_values        (GtkCellView *cell_view,
                                             GtkCellRenderer *renderer,
                                             ...);
```

Sets multiple properties of a cell renderer of *cell_view*, and makes sure the display of *cell_view* is updated.

cell_view: a GtkCellView widget
renderer: one of the renderers of *cell_view*
...: a list of pairs of property names and GValues, finished by NULL

Since 2.6

---

gtk_cell_view_set_model ()

```
void        gtk_cell_view_set_model         (GtkCellView *cell_view,
                                             GtkTreeModel *model);
```

Sets the model for *cell_view*. If *cell_view* already has a model set, it will remove it before setting the new model. If *model* is NULL, then it will unset the old model.

cell_view: a GtkCellView
model: a GtkTreeModel

Since 2.6

---

gtk_cell_view_set_displayed_row ()

```
void        gtk_cell_view_set_displayed_row (GtkCellView *cell_view,
                                             GtkTreePath *path);
```

Sets the row of the model that is currently displayed by the GtkCellView. If the path is unset, then the contents of the cellview "stick" at their last value; this is not normally a desired result, but may be a needed intermediate state if say, the model for the GtkCellView becomes temporarily empty.

cell_view : a GtkCellView

path : a GtkTreePath or NULL to unset.

Since 2.6

---

gtk_cell_view_get_displayed_row ()

```
GtkTreePath* gtk_cell_view_get_displayed_row
                                                (GtkCellView *cell_view);
```

cell_view :

Returns :

---

gtk_cell_view_get_size_of_row ()

```
gboolean    gtk_cell_view_get_size_of_row   (GtkCellView *cell_view,
                                             GtkTreePath *path,
                                             GtkRequisition *requisition);
```

Sets *requisition* to the size needed by *cell_view* to display the model row pointed to by *path*.

cell_view : a GtkCellView

path : a GtkTreePath

requisition : return location for the size

Returns : TRUE

Since 2.6

---

gtk_cell_view_set_background_color ()

```
void        gtk_cell_view_set_background_color
```

```
                                                    (GtkCellView *cell_view,
                                                     const GdkColor *color);
```

Sets the background color of *`view`*.

`cell_view`: a GtkCellView

`color`: the new background color

Since 2.6

---

gtk_cell_view_set_cell_data ()

```
void        gtk_cell_view_set_cell_data     (GtkCellView *cell_view);
```

`cell_view`:

---

gtk_cell_view_get_cell_renderers ()

```
GList*      gtk_cell_view_get_cell_renderers
                                            (GtkCellView *cell_view);
```

Returns the cell renderers which have been added to *`cell_view`*.

`cell_view`: a GtkCellView

Returns: a list of cell renderers. The list, but not the renderers has been newly allocated and should be freed with `g_list_free()` when no longer needed.

Since 2.6

Properties

The "background" property

```
"background"           gchararray           : Write
```

Background color as a string.

Default value: NULL

The "background-gdk" property

```
"background-gdk"       GdkColor             : Read / Write
```

Background color as a GdkColor.

The "background-set" property

```
"background-set"       gboolean             : Read / Write
```

Whether this tag affects the background color.

Default value: FALSE

<< GtkTreeView drag-and-drop **GtkIconView >>**

GtkIconView

GtkIconView — A widget which displays a list of icons in a grid

Synopsis

```
#include <gtk/gtk.h>

            GtkIconView;
            GtkIconViewPrivate;
void        (*GtkIconViewForeachFunc)       (GtkIconView *icon_view,
                                             GtkTreePath *path,
                                             gpointer data);
GtkWidget*  gtk_icon_view_new               (void);
GtkWidget*  gtk_icon_view_new_with_model    (GtkTreeModel *model);
void        gtk_icon_view_set_model         (GtkIconView *icon_view,
                                             GtkTreeModel *model);
GtkTreeModel* gtk_icon_view_get_model       (GtkIconView *icon_view);
void        gtk_icon_view_set_text_column   (GtkIconView *icon_view,
                                             gint column);
gint        gtk_icon_view_get_text_column   (GtkIconView *icon_view);
void        gtk_icon_view_set_markup_column (GtkIconView *icon_view,
                                             gint column);
gint        gtk_icon_view_get_markup_column (GtkIconView *icon_view);
void        gtk_icon_view_set_pixbuf_column (GtkIconView *icon_view,
                                             gint column);
gint        gtk_icon_view_get_pixbuf_column (GtkIconView *icon_view);
GtkTreePath* gtk_icon_view_get_path_at_pos  (GtkIconView *icon_view,
                                             gint x,
                                             gint y);
void        gtk_icon_view_selected_foreach  (GtkIconView *icon_view,
                                             GtkIconViewForeachFunc func,
                                             gpointer data);
void        gtk_icon_view_set_selection_mode
```

```
                                                    (GtkIconView *icon_view,
                                                     GtkSelectionMode mode);
GtkSelectionMode gtk_icon_view_get_selection_mode
                                                    (GtkIconView *icon_view);
void        gtk_icon_view_set_orientation   (GtkIconView *icon_view,
                                                     GtkOrientation orientation);
GtkOrientation gtk_icon_view_get_orientation
                                                    (GtkIconView *icon_view);
void        gtk_icon_view_select_path       (GtkIconView *icon_view,
                                                     GtkTreePath *path);
void        gtk_icon_view_unselect_path     (GtkIconView *icon_view,
                                                     GtkTreePath *path);
gboolean    gtk_icon_view_path_is_selected  (GtkIconView *icon_view,
                                                     GtkTreePath *path);
GList*      gtk_icon_view_get_selected_items
                                                    (GtkIconView *icon_view);
void        gtk_icon_view_select_all        (GtkIconView *icon_view);
void        gtk_icon_view_unselect_all      (GtkIconView *icon_view);
void        gtk_icon_view_item_activated    (GtkIconView *icon_view,
                                                     GtkTreePath *path);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkWidget
               +----GtkContainer
                     +----GtkIconView
```

Implemented Interfaces

GtkIconView implements AtkImplementorIface.

Properties

```
  "markup-column"         gint                : Read / Write
  "model"                 GtkTreeModel        : Read / Write
  "orientation"           GtkOrientation      : Read / Write
  "pixbuf-column"         gint                : Read / Write
  "selection-mode"        GtkSelectionMode    : Read / Write
  "text-column"           gint                : Read / Write
```

Style Properties

```
  "selection-box-alpha"   guchar              : Read
  "selection-box-color"   GdkColor            : Read
```

Signal Prototypes

```
"activate-cursor-item"
            gboolean    user_function      (GtkIconView *iconview,
                                            gpointer user_data);
"item-activated"
            void        user_function      (GtkIconView *iconview,
                                            GtkTreePath *arg1,
                                            gpointer user_data);
"move-cursor"
            gboolean    user_function      (GtkIconView *iconview,
                                            GtkMovementStep arg1,
                                            gint arg2,
                                            gpointer user_data);
"select-all"
            void        user_function      (GtkIconView *iconview,
                                            gpointer user_data);
"select-cursor-item"
            void        user_function      (GtkIconView *iconview,
                                            gpointer user_data);
"selection-changed"
            void        user_function      (GtkIconView *iconview,
                                            gpointer user_data);
```

```
"set-scroll-adjustments"
            void        user_function      (GtkIconView *iconview,
                                            GtkAdjustment *arg1,
                                            GtkAdjustment *arg2,
                                            gpointer user_data);
"toggle-cursor-item"
            void        user_function      (GtkIconView *iconview,
                                            gpointer user_data);
"unselect-all"
            void        user_function      (GtkIconView *iconview,
                                            gpointer user_data);
```

Description

GtkIconView provides an alternative view on a list model. It displays the model as a grid of icons with labels. Like GtkTreeView, it allows to select one or multiple items (depending on the selection mode, see `gtk_icon_view_set_selection_mode()`). In addition to selection with the arrow keys, GtkIconView supports rubberband selection, which is controlled by dragging the pointer.

Details

GtkIconView

```
typedef struct _GtkIconView GtkIconView;
```

The GtkIconView struct contains only private fields and should not be directly accessed.

GtkIconViewPrivate

```
typedef struct _GtkIconViewPrivate GtkIconViewPrivate;
```

GtkIconViewForeachFunc ()

```
void        (*GtkIconViewForeachFunc)       (GtkIconView *icon_view,
                                             GtkTreePath *path,
                                             gpointer data);
```

A function used by `gtk_icon_view_selected_foreach()` to map all selected rows. It will be called on every selected row in the view.

icon_view :
path : The GtkTreePath of a selected row
data : user data

---

gtk_icon_view_new ()

```
GtkWidget*  gtk_icon_view_new               (void);
```

Creates a new GtkIconView widget

Returns : A newly created GtkIconView widget

Since 2.6

---

gtk_icon_view_new_with_model ()

```
GtkWidget*  gtk_icon_view_new_with_model    (GtkTreeModel *model);
```

Creates a new GtkIconView widget with the model *model*.

model : The model.
Returns : A newly created GtkIconView widget.

Since 2.6

---

gtk_icon_view_set_model ()

```
void        gtk_icon_view_set_model         (GtkIconView *icon_view,
                                             GtkTreeModel *model);
```

Sets the model for a GtkIconView. If the *icon_view* already has a model set, it will remove it before setting the new model. If *model* is NULL, then it will unset the old model.

icon_view : A GtkIconView.
model : The model.

Since 2.6

---

gtk_icon_view_get_model ()

```
GtkTreeModel* gtk_icon_view_get_model       (GtkIconView *icon_view);
```

Returns the model the GtkIconView is based on. Returns NULL if the model is unset.

icon_view : a GtkIconView
Returns : A GtkTreeModel, or NULL if none is currently being used.

Since 2.6

---

gtk_icon_view_set_text_column ()

```
void        gtk_icon_view_set_text_column   (GtkIconView *icon_view,
                                             gint column);
```

Sets the column with text for *icon_view* to be *column*. The text column must be of type G_TYPE_STRING.

icon_view : A GtkIconView.
column : A column in the currently used model.

Since 2.6

---

gtk_icon_view_get_text_column ()

```
gint        gtk_icon_view_get_text_column   (GtkIconView *icon_view);
```

Returns the column with text for *icon_view*.

icon_view : A GtkIconView.
Returns : the text column, or -1 if it's unset.

Since 2.6

---

gtk_icon_view_set_markup_column ()

```
void        gtk_icon_view_set_markup_column (GtkIconView *icon_view,
                                             gint column);
```

Sets the column with markup information for *icon_view* to be *column*. The markup column must be of type G_TYPE_STRING. If the markup column is set to something, it overrides the text column set by gtk_icon_view_set_text_column().

icon_view : A GtkIconView.
column : A column in the currently used model.

Since 2.6

---

gtk_icon_view_get_markup_column ()

```
gint        gtk_icon_view_get_markup_column (GtkIconView *icon_view);
```

Returns the column with markup text for *icon_view*.

icon_view : A GtkIconView.

Returns : the markup column, or -1 if it's unset.

Since 2.6

---

gtk_icon_view_set_pixbuf_column ()

```
void        gtk_icon_view_set_pixbuf_column (GtkIconView *icon_view,
                                             gint column);
```

Sets the column with pixbufs for *icon_view* to be *column*. The pixbuf column must be of type GDK_TYPE_PIXBUF

icon_view : A GtkIconView.

column : A column in the currently used model.

Since 2.6

---

gtk_icon_view_get_pixbuf_column ()

```
gint        gtk_icon_view_get_pixbuf_column (GtkIconView *icon_view);
```

Returns the column with pixbufs for *icon_view*.

icon_view : A GtkIconView.

Returns : the pixbuf column, or -1 if it's unset.

Since 2.6

---

gtk_icon_view_get_path_at_pos ()

```
GtkTreePath* gtk_icon_view_get_path_at_pos  (GtkIconView *icon_view,
                                             gint x,
                                             gint y);
```

Finds the path at the point (*x*, *y*), relative to widget coordinates.

icon_view : A GtkIconView.

x : The x position to be identified

y : The y position to be identified

Returns : The GtkTreePath corresponding to the icon or NULL if no icon exists at that position.

Since 2.6

---

gtk_icon_view_selected_foreach ()

```
void        gtk_icon_view_selected_foreach  (GtkIconView *icon_view,
                                             GtkIconViewForeachFunc func,
                                             gpointer data);
```

Calls a function for each selected icon. Note that the model or selection cannot be modified from within this function.

icon_view : A GtkIconView.

func : The funcion to call for each selected icon.
data : User data to pass to the function.

Since 2.6

---

gtk_icon_view_set_selection_mode ()

```
void        gtk_icon_view_set_selection_mode
                                            (GtkIconView *icon_view,
                                             GtkSelectionMode mode);
```

Sets the selection mode of the *icon_view*.

icon_view : A GtkIconView.
mode : The selection mode

Since 2.6

---

gtk_icon_view_get_selection_mode ()

```
GtkSelectionMode gtk_icon_view_get_selection_mode
                                            (GtkIconView *icon_view);
```

Gets the selection mode of the *icon_view*.

icon_view : A GtkIconView.
Returns : the current selection mode

Since 2.6

---

gtk_icon_view_set_orientation ()

```
void        gtk_icon_view_set_orientation   (GtkIconView *icon_view,
                                             GtkOrientation orientation);
```

Sets the ::orientation property which determines whether the labels are drawn beside the icons instead of below.

icon_view : a GtkIconView

orientation : the relative position of texts and icons

Since 2.6

---

gtk_icon_view_get_orientation ()

```
GtkOrientation gtk_icon_view_get_orientation
                                            (GtkIconView *icon_view);
```

Returns the value of the ::orientation property which determines whether the labels are drawn beside the icons instead of below.

icon_view : a GtkIconView

Returns : the relative position of texts and icons

Since 2.6

---

gtk_icon_view_select_path ()

```
void        gtk_icon_view_select_path       (GtkIconView *icon_view,
                                             GtkTreePath *path);
```

Selects the row at *path*.

icon_view : A GtkIconView.

path : The GtkTreePath to be selected.

Since 2.6

---

gtk_icon_view_unselect_path ()

```
void        gtk_icon_view_unselect_path     (GtkIconView *icon_view,
                                             GtkTreePath *path);
```

Unselects the row at *path*.

icon_view : A GtkIconView.

path : The GtkTreePath to be unselected.

Since 2.6

---

gtk_icon_view_path_is_selected ()

```
gboolean    gtk_icon_view_path_is_selected  (GtkIconView *icon_view,
                                             GtkTreePath *path);
```

Returns TRUE if the icon pointed to by *path* is currently selected. If *icon* does not point to a valid location, FALSE is returned.

icon_view : A GtkIconView.

path : A GtkTreePath to check selection on.

Returns : TRUE if *path* is selected.

Since 2.6

gtk_icon_view_get_selected_items ()

```
GList*      gtk_icon_view_get_selected_items
                                            (GtkIconView *icon_view);
```

Creates a list of path of all selected items. Additionally, if you are planning on modifying the model after calling this function, you may want to convert the returned list into a list of GtkTreeRowReferences. To do this, you can use `gtk_tree_row_reference_new()`.

To free the return value, use:

```
g_list_foreach (list, gtk_tree_path_free, NULL);
g_list_free (list);
```

`icon_view` : A GtkIconView.

Returns : A GList containing a GtkTreePath for each selected row.

Since 2.6

---

gtk_icon_view_select_all ()

```
void        gtk_icon_view_select_all        (GtkIconView *icon_view);
```

Selects all the icons. *`icon_view`* must has its selection mode set to GTK_SELECTION_MULTIPLE.

`icon_view` : A GtkIconView.

Since 2.6

---

gtk_icon_view_unselect_all ()

```
void        gtk_icon_view_unselect_all      (GtkIconView *icon_view);
```

Unselects all the icons.

icon_view : A GtkIconView.

Since 2.6

---

gtk_icon_view_item_activated ()

```
void        gtk_icon_view_item_activated    (GtkIconView *icon_view,
                                             GtkTreePath *path);
```

Activates the item determined by *path*.

icon_view : A GtkIconView
path : The GtkTreePath to be activated

Since 2.6

Properties

The "markup-column" property

```
  "markup-column"        gint                  : Read / Write
```

The ::markup-column property contains the number of the model column containing markup information to be displayed. The markup column must be of type G_TYPE_STRING. If this property and the :text-column property are both set to column numbers, it overrides the text column. If both are set to -1, no texts are displayed.

Allowed values: >= -1

Default value: -1

Since 2.6

The "model" property

"model"	GtkTreeModel	: Read / Write

The model for the icon view.

The "orientation" property

"orientation"	GtkOrientation	: Read / Write

How the text and icon of each item are positioned relative to each other.

Default value: GTK_ORIENTATION_VERTICAL

The "pixbuf-column" property

"pixbuf-column"	gint	: Read / Write

The ::pixbuf-column property contains the number of the model column containing the pixbufs which are displayed. The pixbuf column must be of type GDK_TYPE_PIXBUF. Setting this property to -1 turns off the display of pixbufs.

Allowed values: >= -1

Default value: -1

Since 2.6

The "selection-mode" property

```
"selection-mode"       GtkSelectionMode      : Read / Write
```

The ::selection-mode property specifies the selection mode of icon view. If the mode is GTK_SELECTION_MULTIPLE, rubberband selection is enabled, for the other modes, only keyboard selection is possible.

Default value: GTK_SELECTION_SINGLE

Since 2.6

The "text-column" property

```
"text-column"          gint                  : Read / Write
```

The ::text-column property contains the number of the model column containing the texts which are displayed. The text column must be of type G_TYPE_STRING. If this property and the :markup-column property are both set to -1, no texts are displayed.

Allowed values: >= -1

Default value: -1

Since 2.6

Style Properties

The "selection-box-alpha" style property

```
"selection-box-alpha"  guchar                : Read
```

Opacity of the selection box.

Default value: 64

The "selection-box-color" style property

```
  "selection-box-color"  GdkColor              : Read
```

Color of the selection box.

Signals

The "activate-cursor-item" signal

```
gboolean    user_function                (GtkIconView *iconview,
                                          gpointer user_data);
```

iconview : the object which received the signal.
user_data : user data set when the signal handler was connected.
Returns :

The "item-activated" signal

```
void        user_function                (GtkIconView *iconview,
                                          GtkTreePath *arg1,
                                          gpointer user_data);
```

iconview : the object which received the signal.
arg1 :
user_data : user data set when the signal handler was connected.

The "move-cursor" signal

```
gboolean    user_function                   (GtkIconView *iconview,
                                             GtkMovementStep arg1,
                                             gint arg2,
                                             gpointer user_data);
```

iconview : the object which received the signal.
arg1 :
arg2 :
user_data : user data set when the signal handler was connected.
Returns :

The "select-all" signal

```
void        user_function                   (GtkIconView *iconview,
                                             gpointer user_data);
```

iconview : the object which received the signal.
user_data : user data set when the signal handler was connected.

The "select-cursor-item" signal

```
void        user_function                   (GtkIconView *iconview,
                                             gpointer user_data);
```

iconview : the object which received the signal.
user_data : user data set when the signal handler was connected.

The "selection-changed" signal

```
void        user_function                   (GtkIconView *iconview,
```

```
                                        gpointer user_data);
```

iconview : the object which received the signal.
user_data : user data set when the signal handler was connected.

The "set-scroll-adjustments" signal

```
void        user_function                   (GtkIconView *iconview,
                                             GtkAdjustment *arg1,
                                             GtkAdjustment *arg2,
                                             gpointer user_data);
```

iconview : the object which received the signal.
arg1 :
arg2 :
user_data : user data set when the signal handler was connected.

The "toggle-cursor-item" signal

```
void        user_function                   (GtkIconView *iconview,
                                             gpointer user_data);
```

iconview : the object which received the signal.
user_data : user data set when the signal handler was connected.

The "unselect-all" signal

```
void        user_function                   (GtkIconView *iconview,
                                             gpointer user_data);
```

iconview: the object which received the signal.

user_data: user data set when the signal handler was connected.

<< GtkCellView **GtkTreeSortable >>**

GtkTreeSortable

GtkTreeSortable — The interface for sortable models used by GtkTreeView

Synopsis

```
#include <gtk/gtk.h>

            GtkTreeSortable;
            GtkTreeSortableIface;
gint        (*GtkTreeIterCompareFunc)       (GtkTreeModel *model,
                                             GtkTreeIter *a,
                                             GtkTreeIter *b,
                                             gpointer user_data);
void        gtk_tree_sortable_sort_column_changed
                                            (GtkTreeSortable *sortable);
gboolean    gtk_tree_sortable_get_sort_column_id
                                            (GtkTreeSortable *sortable,
                                             gint *sort_column_id,
                                             GtkSortType *order);
void        gtk_tree_sortable_set_sort_column_id
                                            (GtkTreeSortable *sortable,
                                             gint sort_column_id,
                                             GtkSortType order);
void        gtk_tree_sortable_set_sort_func (GtkTreeSortable *sortable,
                                             gint sort_column_id,
                                             GtkTreeIterCompareFunc sort_func,
                                             gpointer user_data,
                                             GtkDestroyNotify destroy);
void        gtk_tree_sortable_set_default_sort_func
                                            (GtkTreeSortable *sortable,
                                             GtkTreeIterCompareFunc sort_func,
                                             gpointer user_data,
                                             GtkDestroyNotify destroy);
gboolean    gtk_tree_sortable_has_default_sort_func
                                            (GtkTreeSortable *sortable);
```

Object Hierarchy

```
  GInterface
   +----GtkTreeSortable
```

Prerequisites

GtkTreeSortable requires GtkTreeModel and GObject.

Known Implementations

GtkTreeSortable is implemented by GtkTreeModelSort, GtkTreeStore and GtkListStore.

Signal Prototypes

```
"sort-column-changed"
            void        user_function      (GtkTreeSortable *treesortable,
                                            gpointer user_data);
```

Description

Details

GtkTreeSortable

```
typedef struct _GtkTreeSortable GtkTreeSortable;
```

GtkTreeSortableIface

```
```

```
typedef struct {
  GTypeInterface g_iface;

  /* signals */
  void     (* sort_column_changed)   (GtkTreeSortable        *sortable);

  /* virtual table */
  gboolean (* get_sort_column_id)    (GtkTreeSortable        *sortable,
                                      gint                   *sort_column_id,
                                      GtkSortType            *order);
  void     (* set_sort_column_id)    (GtkTreeSortable        *sortable,
                                      gint                    sort_column_id,
                                      GtkSortType             order);
  void     (* set_sort_func)         (GtkTreeSortable        *sortable,
                                      gint                    sort_column_id,
                                      GtkTreeIterCompareFunc  func,
                                      gpointer                data,
                                      GtkDestroyNotify        destroy);
  void     (* set_default_sort_func) (GtkTreeSortable        *sortable,
                                      GtkTreeIterCompareFunc  func,
                                      gpointer                data,
                                      GtkDestroyNotify        destroy);
  gboolean (* has_default_sort_func) (GtkTreeSortable        *sortable);
} GtkTreeSortableIface;
```

GtkTreeIterCompareFunc ()

```
gint        (*GtkTreeIterCompareFunc)       (GtkTreeModel *model,
                                             GtkTreeIter *a,
                                             GtkTreeIter *b,
                                             gpointer user_data);
```

A GtkTreeIterCompareFunc should return a negative integer, zero, or a positive integer if *a* sorts before *b*, *a* sorts with *b*, or *a* sorts after *b* respectively. If two iters compare as equal, their order in the sorted model is undefined. In order to ensure that the GtkTreeSortable behaves as expected, the GtkTreeIterCompareFunc must define a partial order on the model, i.e. it must be reflexive, antisymmetric and transitive.

For example, if *model* is a product catalogue, then a compare function for the "price" column could be one which returns `price_of(a) - price_of(b)`.

model :	The GtkTreeModel the comparison is within
a :	A GtkTreeIter in *model*

b : Another GtkTreeIter in *model*

user_data : Data passed when the compare func is assigned e.g. by gtk_tree_sortable_set_sort_func()

Returns :

---

gtk_tree_sortable_sort_column_changed ()

```
void        gtk_tree_sortable_sort_column_changed
                                            (GtkTreeSortable *sortable);
```

Emits a GtkTreeSortable::sort_column_changed signal on

sortable : A GtkTreeSortable

---

gtk_tree_sortable_get_sort_column_id ()

```
gboolean    gtk_tree_sortable_get_sort_column_id
                                            (GtkTreeSortable *sortable,
                                             gint *sort_column_id,
                                             GtkSortType *order);
```

Fills in *sort_column_id* and *order* with the current sort column and the order, if applicable. If the sort column is not set, then FALSE is returned, and the values in *sort_column_id* and *order* are unchanged.

sortable : A GtkTreeSortable

sort_column_id : The sort column id to be filled in

order : The GtkSortType to be filled in

Returns : TRUE, if the sort column has been set

---

gtk_tree_sortable_set_sort_column_id ()

```
void        gtk_tree_sortable_set_sort_column_id
                                            (GtkTreeSortable *sortable,
                                             gint sort_column_id,
                                             GtkSortType order);
```

Sets the current sort column to be *sort_column_id*. The *sortable* will resort itself to reflect this change, after emitting a GtkTreeSortable::sort_column_changed signal. If *sort_column_id* is GTK_TREE_SORTABLE_DEFAULT_SORT_COLUMN_ID, then the default sort function will be used, if it is set.

sortable :	A GtkTreeSortable
sort_column_id :	the sort column id to set
order :	The sort order of the column

---

gtk_tree_sortable_set_sort_func ()

```
void        gtk_tree_sortable_set_sort_func (GtkTreeSortable *sortable,
                                             gint sort_column_id,
                                             GtkTreeIterCompareFunc sort_func,
                                             gpointer user_data,
                                             GtkDestroyNotify destroy);
```

Sets the comparison function used when sorting to be *sort_func*. If the current sort column id of *sortable* is the same as *sort_column_id*, then the model will sort using this function.

sortable :	A GtkTreeSortable
sort_column_id :	the sort column id to set the function for
sort_func :	The sorting function
user_data :	User data to pass to the sort func, or NULL
destroy :	Destroy notifier of *user_data*, or NULL

---

gtk_tree_sortable_set_default_sort_func ()

```
void        gtk_tree_sortable_set_default_sort_func
                                            (GtkTreeSortable *sortable,
                                             GtkTreeIterCompareFunc sort_func,
                                             gpointer user_data,
                                             GtkDestroyNotify destroy);
```

Sets the default comparison function used when sorting to be *sort_func*. If the current sort column id of *sortable* is GTK_TREE_SORTABLE_DEFAULT_SORT_COLUMN_ID, then the model will sort using this function.

If *sort_func* is NULL, then there will be no default comparison function. This means that once the model has been sorted, it can't go back to the default state. In this case, when the current sort column id of *sortable* is GTK_TREE_SORTABLE_DEFAULT_SORT_COLUMN_ID, the model will be unsorted.

sortable : A GtkTreeSortable
sort_func : The sorting function
user_data : User data to pass to the sort func, or NULL
destroy : Destroy notifier of *user_data*, or NULL

---

gtk_tree_sortable_has_default_sort_func ()

```
gboolean    gtk_tree_sortable_has_default_sort_func
                                            (GtkTreeSortable *sortable);
```

Returns TRUE if the model has a default sort function. This is used primarily by GtkTreeViewColumns in order to determine if a model can go back to the default state, or not.

sortable : A GtkTreeSortable
Returns : TRUE, if the model has a default sort function

Signals

The "sort-column-changed" signal

```
void        user_function                  (GtkTreeSortable *treesortable,
                                            gpointer user_data);
```

treesortable : the object which received the signal.
user_data : user data set when the signal handler was connected.

GtkTreeModelSort

GtkTreeModelSort — A GtkTreeModel which makes an underlying tree model sortable

Synopsis

```
#include <gtk/gtk.h>

            GtkTreeModelSort;
GtkTreeModel* gtk_tree_model_sort_new_with_model
                                            (GtkTreeModel *child_model);
GtkTreeModel* gtk_tree_model_sort_get_model (GtkTreeModelSort *tree_model);
GtkTreePath* gtk_tree_model_sort_convert_child_path_to_path
                                            (GtkTreeModelSort *tree_model_sort,
                                             GtkTreePath *child_path);
void        gtk_tree_model_sort_convert_child_iter_to_iter
                                            (GtkTreeModelSort *tree_model_sort,
                                             GtkTreeIter *sort_iter,
                                             GtkTreeIter *child_iter);
GtkTreePath* gtk_tree_model_sort_convert_path_to_child_path
                                            (GtkTreeModelSort *tree_model_sort,
                                             GtkTreePath *sorted_path);
void        gtk_tree_model_sort_convert_iter_to_child_iter
                                            (GtkTreeModelSort *tree_model_sort,
                                             GtkTreeIter *child_iter,
                                             GtkTreeIter *sorted_iter);
void        gtk_tree_model_sort_reset_default_sort_func
                                            (GtkTreeModelSort *tree_model_sort);
void        gtk_tree_model_sort_clear_cache (GtkTreeModelSort *tree_model_sort);
gboolean    gtk_tree_model_sort_iter_is_valid
                                            (GtkTreeModelSort *tree_model_sort,
                                             GtkTreeIter *iter);
```

Object Hierarchy

```
GObject
 +----GtkTreeModelSort
```

Implemented Interfaces

GtkTreeModelSort implements GtkTreeModel, GtkTreeDragSource and GtkTreeSortable.

Properties

```
"model"                GtkTreeModel          : Read / Write / Construct Only
```

Description

The GtkTreeModelSort is a model which implements the GtkTreeSortable interface. It does not hold any data itself, but rather is created with a child model and proxies its data. It has identical column types to this child model, and the changes in the child are propagated. The primary purpose of this model is to provide a way to sort a different model without modifying it.

The use of this is best demonstrated through an example. In the following sample code we create two GtkTreeView widgets each with a view of the same data. As the model is wrapped here by a GtkTreeModelSort, the two GtkTreeViews can each sort their view of the data without affecting the other. By contrast, if we simply put the same model in each widget, then sorting the first would sort the second.

Example 3. Using a GtkTreeModelSort

```
{
  GtkTreeView *tree_view1;
  GtkTreeView *tree_view2;
  GtkTreeModel *sort_model1;
  GtkTreeModel *sort_model2;
  GtkTreeModel *child_model;

  /* get the child model */
  child_model = get_my_model();

  /* Create the first tree */
  sort_model1 = gtk_tree_model_sort_new_with_model (child_model);
  tree_view1 = gtk_tree_view_new_with_model (sort_model1);

  /* Create the second tree */
  sort_model2 = gtk_tree_model_sort_new_with_model (child_model);
  tree_view2 = gtk_tree_view_new_with_model (sort_model2);

  /* Now we can sort the two models independently */
  gtk_tree_sortable_set_sort_column_id (GTK_TREE_SORTABLE (sort_model1),
                                        COLUMN_1, GTK_SORT_ASCENDING);
```

```
  gtk_tree_sortable_set_sort_column_id (GTK_TREE_SORTABLE (sort_model2),
                                        COLUMN_1, GTK_SORT_DESCENDING);
}
```

To demonstrate how to access the underlying child model from the sort model, the next example will be a callback for the GtkTreeSelection "changed" signal. In this callback, we get a string from COLUMN_1 of the model. We then modify the string, find the same selected row on the child model, and change the row there.

Example 4. Accessing the child model of in a selection changed callback

```
void
selection_changed (GtkTreeSelection *selection, gpointer data)
{
  GtkTreeModel *sort_model = NULL;
  GtkTreeModel *child_model;
  GtkTreeIter sort_iter;
  GtkTreeIter child_iter;
  char *some_data = NULL;
  char *modified_data;

  /* Get the current selected row and the model. */
  if (! gtk_tree_selection_get_selected (selection,
                                         &sort_model,
                                         &sort_iter))
    return;


  /* Look up the current value on the selected row and get a new value
   * to change it to.
   */
  gtk_tree_model_get (GTK_TREE_MODEL (sort_model), &sort_iter,
                      COLUMN_1, &some_data,
                      -1);

  modified_data = change_the_data (some_data);
  g_free (some_data);

  /* Get an iterator on the child model, instead of the sort model. */
  gtk_tree_model_sort_convert_iter_to_child_iter (GTK_TREE_MODEL_SORT (sort_model),
                                                  &child_iter,
                                                  &sort_iter);

  /* Get the child model and change the value of the row.  In this
   * example, the child model is a GtkListStore.  It could be any other
   * type of model, though.
   */
  child_model = gtk_tree_model_sort_get_model (GTK_TREE_MODEL_SORT (sort_model));
  gtk_list_store_set (GTK_LIST_STORE (child_model), &child_iter,
                      COLUMN_1, &modified_data,
                      -1);
  g_free (modified_data);
```

```
}
```

Details

GtkTreeModelSort

```
typedef struct _GtkTreeModelSort GtkTreeModelSort;
```

This should not be accessed directly. Use the accessor functions below.

---

gtk_tree_model_sort_new_with_model ()

```
GtkTreeModel* gtk_tree_model_sort_new_with_model
                                            (GtkTreeModel *child_model);
```

Creates a new GtkTreeModel, with *child_model* as the child_model.

child_model : A GtkTreeModel

Returns : A new GtkTreeModel.

---

gtk_tree_model_sort_get_model ()

```
GtkTreeModel* gtk_tree_model_sort_get_model (GtkTreeModelSort *tree_model);
```

Returns the model the GtkTreeModelSort is sorting.

tree_model : a GtkTreeModelSort

Returns : the "child model" being sorted

---

gtk_tree_model_sort_convert_child_path_to_path ()

```
GtkTreePath* gtk_tree_model_sort_convert_child_path_to_path
                                            (GtkTreeModelSort *tree_model_sort,
                                             GtkTreePath *child_path);
```

Converts *child_path* to a path relative to *tree_model_sort*. That is, *child_path* points to a path in the child model. The returned path will point to the same row in the sorted model. If *child_path* isn't a valid path on the child model, then NULL is returned.

tree_model_sort :	A GtkTreeModelSort
child_path :	A GtkTreePath to convert
Returns :	A newly allocated GtkTreePath, or NULL

---

gtk_tree_model_sort_convert_child_iter_to_iter ()

```
void        gtk_tree_model_sort_convert_child_iter_to_iter
                                            (GtkTreeModelSort *tree_model_sort,
                                             GtkTreeIter *sort_iter,
                                             GtkTreeIter *child_iter);
```

Sets *sort_iter* to point to the row in *tree_model_sort* that corresponds to the row pointed at by *child_iter*.

tree_model_sort :	A GtkTreeModelSort
sort_iter :	An uninitialized GtkTreeIter.
child_iter :	A valid GtkTreeIter pointing to a row on the child model

---

gtk_tree_model_sort_convert_path_to_child_path ()

```
GtkTreePath* gtk_tree_model_sort_convert_path_to_child_path
                                            (GtkTreeModelSort *tree_model_sort,
                                             GtkTreePath *sorted_path);
```

Converts *sorted_path* to a path on the child model of *tree_model_sort*. That is, *sorted_path* points to a location in *tree_model_sort*. The returned path will point to the same location in the model not being sorted. If *sorted_path* does not point to a location in the child model, NULL is returned.

tree_model_sort :	A GtkTreeModelSort
sorted_path :	A GtkTreePath to convert
Returns :	A newly allocated GtkTreePath, or NULL

---

gtk_tree_model_sort_convert_iter_to_child_iter ()

```
void        gtk_tree_model_sort_convert_iter_to_child_iter
                                            (GtkTreeModelSort *tree_model_sort,
                                             GtkTreeIter *child_iter,
                                             GtkTreeIter *sorted_iter);
```

Sets *child_iter* to point to the row pointed to by *sorted_iter*.

tree_model_sort : A GtkTreeModelSort
child_iter : An uninitialized GtkTreeIter
sorted_iter : A valid GtkTreeIter pointing to a row on *tree_model_sort*.

---

gtk_tree_model_sort_reset_default_sort_func ()

```
void        gtk_tree_model_sort_reset_default_sort_func
                                            (GtkTreeModelSort *tree_model_sort);
```

This resets the default sort function to be in the 'unsorted' state. That is, it is in the same order as the child model. It will re-sort the model to be in the same order as the child model only if the GtkTreeModelSort is in 'unsorted' state.

tree_model_sort : A GtkTreeModelSort

---

gtk_tree_model_sort_clear_cache ()

```
void        gtk_tree_model_sort_clear_cache (GtkTreeModelSort *tree_model_sort);
```

This function should almost never be called. It clears the *tree_model_sort* of any cached iterators that haven't been reffed with gtk_tree_model_ref_node(). This might be useful if the child model being sorted is static (and doesn't change often) and there has been a lot of unreffed access to nodes. As a side effect of this function, all unreffed iters will be invalid.

tree_model_sort : A GtkTreeModelSort

---

gtk_tree_model_sort_iter_is_valid ()

```
gboolean    gtk_tree_model_sort_iter_is_valid
                                            (GtkTreeModelSort *tree_model_sort,
                                             GtkTreeIter *iter);
```

WARNING: This function is slow. Only use it for debugging and/or testing purposes.

Checks if the given iter is a valid iter for this GtkTreeModelSort.

tree_model_sort :	A GtkTreeModelSort.
iter :	A GtkTreeIter.
Returns :	TRUE if the iter is valid, FALSE if the iter is invalid.

Since 2.2

Properties

The "model" property

"model"	GtkTreeModel	: Read / Write / Construct Only

The model for the TreeModelSort to sort.

See Also

GtkTreeModel, GtkListStore, GtkTreeStore, GtkTreeSortable, GtkTreeModelFilter

<< GtkTreeSortable **GtkTreeModelFilter >>**

GtkTreeModelFilter

GtkTreeModelFilter — A GtkTreeModel which hides parts of an underlying tree model

Synopsis

```
#include <gtk/gtk.h>

            GtkTreeModelFilter;
gboolean    (*GtkTreeModelFilterVisibleFunc)
                                            (GtkTreeModel *model,
                                             GtkTreeIter *iter,
                                             gpointer data);
void        (*GtkTreeModelFilterModifyFunc) (GtkTreeModel *model,
                                             GtkTreeIter *iter,
                                             GValue *value,
                                             gint column,
                                             gpointer data);
GtkTreeModel* gtk_tree_model_filter_new     (GtkTreeModel *child_model,
                                             GtkTreePath *root);
void        gtk_tree_model_filter_set_visible_func
                                            (GtkTreeModelFilter *filter,
                                             GtkTreeModelFilterVisibleFunc func,
                                             gpointer data,
                                             GtkDestroyNotify destroy);
void        gtk_tree_model_filter_set_modify_func
                                            (GtkTreeModelFilter *filter,
                                             gint n_columns,
                                             GType *types,
                                             GtkTreeModelFilterModifyFunc func,
                                             gpointer data,
                                             GtkDestroyNotify destroy);
void        gtk_tree_model_filter_set_visible_column
                                            (GtkTreeModelFilter *filter,
                                             gint column);
GtkTreeModel* gtk_tree_model_filter_get_model
                                            (GtkTreeModelFilter *filter);
```

```
void        gtk_tree_model_filter_convert_child_iter_to_iter
                                            (GtkTreeModelFilter *filter,
                                             GtkTreeIter *filter_iter,
                                             GtkTreeIter *child_iter);
void        gtk_tree_model_filter_convert_iter_to_child_iter
                                            (GtkTreeModelFilter *filter,
                                             GtkTreeIter *child_iter,
                                             GtkTreeIter *filter_iter);
GtkTreePath* gtk_tree_model_filter_convert_child_path_to_path
                                            (GtkTreeModelFilter *filter,
                                             GtkTreePath *child_path);
GtkTreePath* gtk_tree_model_filter_convert_path_to_child_path
                                            (GtkTreeModelFilter *filter,
                                             GtkTreePath *filter_path);
void        gtk_tree_model_filter_refilter  (GtkTreeModelFilter *filter);
void        gtk_tree_model_filter_clear_cache
                                            (GtkTreeModelFilter *filter);
```

Object Hierarchy

```
  GObject
   +----GtkTreeModelFilter
```

Implemented Interfaces

GtkTreeModelFilter implements GtkTreeModel and GtkTreeDragSource.

Properties

```
  "child-model"          GtkTreeModel          : Read / Write / Construct Only
  "virtual-root"         GtkTreePath           : Read / Write / Construct Only
```

Description

A GtkTreeModelFilter is a tree model which wraps another tree model, and can do the following things:

- Filter specific rows, based on data from a "visible column", a column storing booleans indicating whether the row should be filtered or not, or based on the return value of a "visible function", which gets a model, iter and user_data and returns a boolean indicating whether the row should be filtered or not.
- Modify the "appearance" of the model, using a modify function. This is extremely powerful and allows for just changing some values and also for creating a completely different model based on the given child model.
- Set a different root node, also known as a "virtual root". You can pass in a GtkTreePath indicating the root node for the filter at construction time.

Details

GtkTreeModelFilter

```
typedef struct _GtkTreeModelFilter GtkTreeModelFilter;
```

The GtkTreeModelFilter struct contains only private fields.

GtkTreeModelFilterVisibleFunc ()

```
gboolean    (*GtkTreeModelFilterVisibleFunc)
                                        (GtkTreeModel *model,
                                         GtkTreeIter *iter,
                                         gpointer data);
```

A function which decides whether the row indicated by *iter* is visible.

model : the child model of the GtkTreeModelFilter

iter : a GtkTreeIter pointing to the row in *model* whose visibility is determined

data : user data given to gtk_tree_model_filter_set_visible_func()

Returns : Whether the row indicated by *iter* is visible.

GtkTreeModelFilterModifyFunc ()

```
void        (*GtkTreeModelFilterModifyFunc) (GtkTreeModel *model,
                                         GtkTreeIter *iter,
                                         GValue *value,
                                         gint column,
                                         gpointer data);
```

A function which calculates display values from raw values in the model. It must fill *value* with the display value for the column *column* in the row indicated by *iter*.

Since this function is called for each data access, it's not a particularly efficient operation.

model : the GtkTreeModelFilter
iter : a GtkTreeIter pointing to the row whose display values are determined
value : A GValue which is already initialized for with the correct type for the column *column*.
column : the column whose display value is determined
data : user data given to `gtk_tree_model_filter_set_modify_func()`

---

gtk_tree_model_filter_new ()

```
GtkTreeModel* gtk_tree_model_filter_new     (GtkTreeModel *child_model,
                                             GtkTreePath *root);
```

Creates a new GtkTreeModel, with *child_model* as the child_model and *root* as the virtual root.

child_model : A GtkTreeModel.
root : A GtkTreePath or NULL.
Returns : A new GtkTreeModel.

Since 2.4

---

gtk_tree_model_filter_set_visible_func ()

```
void        gtk_tree_model_filter_set_visible_func
                                            (GtkTreeModelFilter *filter,
                                             GtkTreeModelFilterVisibleFunc func,
                                             gpointer data,
                                             GtkDestroyNotify destroy);
```

Sets the visible function used when filtering the *filter* to be *func*. The function should return TRUE if the given row should be visible and FALSE otherwise.

filter : A GtkTreeModelFilter.

func : A GtkTreeModelFilterVisibleFunc, the visible function.

data : User data to pass to the visible function, or NULL.

destroy : Destroy notifier of *data*, or NULL.

Since 2.4

---

gtk_tree_model_filter_set_modify_func ()

```
void        gtk_tree_model_filter_set_modify_func
                                            (GtkTreeModelFilter *filter,
                                             gint n_columns,
                                             GType *types,
                                             GtkTreeModelFilterModifyFunc func,
                                             gpointer data,
                                             GtkDestroyNotify destroy);
```

With the *n_columns* and *types* parameters, you give an array of column types for this model (which will be exposed to the parent model/view). The *func*, *data* and *destroy* parameters are for specifying the modify function. The modify function will get called for *each* data access, the goal of the modify function is to return the data which should be displayed at the location specified using the parameters of the modify function.

filter : A GtkTreeModelFilter.

n_columns : The number of columns in the filter model.

types : The GTypes of the columns.

func : A GtkTreeModelFilterModifyFunc

data : User data to pass to the modify function, or NULL.

destroy : Destroy notifier of *data*, or NULL.

Since 2.4

---

gtk_tree_model_filter_set_visible_column ()

```
void        gtk_tree_model_filter_set_visible_column
                                            (GtkTreeModelFilter *filter,
                                             gint column);
```

Sets *column* of the child_model to be the column where *filter* should look for visibility information. *columns* should be a column of type G_TYPE_BOOLEAN, where TRUE means that a row is visible, and FALSE if not.

filter : A GtkTreeModelFilter.

column : A gint which is the column containing the visible information.

Since 2.4

---

gtk_tree_model_filter_get_model ()

```
GtkTreeModel* gtk_tree_model_filter_get_model
                                        (GtkTreeModelFilter *filter);
```

Returns a pointer to the child model of *filter*.

filter : A GtkTreeModelFilter.

Returns : A pointer to a GtkTreeModel.

Since 2.4

---

gtk_tree_model_filter_convert_child_iter_to_iter ()

```
void        gtk_tree_model_filter_convert_child_iter_to_iter
                                        (GtkTreeModelFilter *filter,
                                         GtkTreeIter *filter_iter,
                                         GtkTreeIter *child_iter);
```

Sets *filter_iter* to point to the row in *filter* that corresponds to the row pointed at by *child_iter*.

filter : A GtkTreeModelFilter.

filter_iter : An uninitialized GtkTreeIter.

child_iter : A valid GtkTreeIter pointing to a row on the child model.

Since 2.4

---

gtk_tree_model_filter_convert_iter_to_child_iter ()

```
void        gtk_tree_model_filter_convert_iter_to_child_iter
                                            (GtkTreeModelFilter *filter,
                                             GtkTreeIter *child_iter,
                                             GtkTreeIter *filter_iter);
```

Sets *child_iter* to point to the row pointed to by *filter_iter*.

filter : A GtkTreeModelFilter.
child_iter : An uninitialized GtkTreeIter.
filter_iter : A valid GtkTreeIter pointing to a row on *filter*.

Since 2.4

---

gtk_tree_model_filter_convert_child_path_to_path ()

```
GtkTreePath* gtk_tree_model_filter_convert_child_path_to_path
                                            (GtkTreeModelFilter *filter,
                                             GtkTreePath *child_path);
```

Converts *child_path* to a path relative to *filter*. That is, *child_path* points to a path in the child model. The rerturned path will point to the same row in the filtered model. If *child_path* isn't a valid path on the child model, then NULL is returned.

filter : A GtkTreeModelFilter.
child_path : A GtkTreePath to convert.
Returns : A newly allocated GtkTreePath, or NULL.

Since 2.4

---

gtk_tree_model_filter_convert_path_to_child_path ()

```
GtkTreePath* gtk_tree_model_filter_convert_path_to_child_path
```

```
                                                (GtkTreeModelFilter *filter,
                                                 GtkTreePath *filter_path);
```

Converts *filter_path* to a path on the child model of *filter*. That is, *filter_path* points to a location in *filter*. The returned path will point to the same location in the model not being filtered. If *filter_path* does not point to a location in the child model, NULL is returned.

filter : A GtkTreeModelFilter.
filter_path : A GtkTreePath to convert.
Returns : A newly allocated GtkTreePath, or NULL.

Since 2.4

---

gtk_tree_model_filter_refilter ()

```
void        gtk_tree_model_filter_refilter  (GtkTreeModelFilter *filter);
```

Emits ::row_changed for each row in the child model, which causes the filter to re-evaluate whether a row is visible or not.

filter : A GtkTreeModelFilter.

Since 2.4

---

gtk_tree_model_filter_clear_cache ()

```
void        gtk_tree_model_filter_clear_cache
                                                (GtkTreeModelFilter *filter);
```

This function should almost never be called. It clears the *filter* of any cached iterators that haven't been reffed with gtk_tree_model_ref_node(). This might be useful if the child model being filtered is static (and doesn't change often) and there has been a lot of unreffed access to nodes. As a side effect of this function, all unreffed itters will be invalid.

filter : A GtkTreeModelFilter.

Since 2.4

Properties

The "child-model" property

```
  "child-model"          GtkTreeModel          : Read / Write / Construct Only
```

The model for the filtermodel to filter.

The "virtual-root" property

```
  "virtual-root"         GtkTreePath           : Read / Write / Construct Only
```

The virtual root (relative to the child model) for this filtermodel.

See Also

GtkTreeModelSort

<< GtkTreeModelSort **GtkCellLayout >>**

GtkCellLayout

GtkCellLayout — An interface for packing cells

Synopsis

```
#include <gtk/gtk.h>

            GtkCellLayout;
            GtkCellLayoutIface;
void        (*GtkCellLayoutDataFunc)        (GtkCellLayout *cell_layout,
                                             GtkCellRenderer *cell,
                                             GtkTreeModel *tree_model,
                                             GtkTreeIter *iter,
                                             gpointer data);
void        gtk_cell_layout_pack_start      (GtkCellLayout *cell_layout,
                                             GtkCellRenderer *cell,
                                             gboolean expand);
void        gtk_cell_layout_pack_end        (GtkCellLayout *cell_layout,
                                             GtkCellRenderer *cell,
                                             gboolean expand);
void        gtk_cell_layout_reorder         (GtkCellLayout *cell_layout,
                                             GtkCellRenderer *cell,
                                             gint position);
void        gtk_cell_layout_clear           (GtkCellLayout *cell_layout);
void        gtk_cell_layout_set_attributes  (GtkCellLayout *cell_layout,
                                             GtkCellRenderer *cell,
                                             ...);
void        gtk_cell_layout_add_attribute   (GtkCellLayout *cell_layout,
                                             GtkCellRenderer *cell,
                                             const gchar *attribute,
                                             gint column);
void        gtk_cell_layout_set_cell_data_func
                                            (GtkCellLayout *cell_layout,
```

```
                                                GtkCellRenderer *cell,
                                                GtkCellLayoutDataFunc func,
                                                gpointer func_data,
                                                GDestroyNotify destroy);
void        gtk_cell_layout_clear_attributes
                                               (GtkCellLayout *cell_layout,
                                                GtkCellRenderer *cell);
```

Object Hierarchy

```
  GInterface
   +----GtkCellLayout
```

Prerequisites

GtkCellLayout requires GObject.

Known Implementations

GtkCellLayout is implemented by GtkCellView, GtkEntryCompletion, GtkTreeViewColumn, GtkComboBox and GtkComboBoxEntry.

Description

GtkCellLayout is an interface to be implemented by all objects which want to provide a GtkTreeViewColumn-like API for packing cells, setting attributes and data funcs.

Details

GtkCellLayout

```
typedef struct _GtkCellLayout GtkCellLayout;
```

GtkCellLayoutIface

```
typedef struct {
  GTypeInterface g_iface;

  /* Virtual Table */
  void (* pack_start)         (GtkCellLayout         *cell_layout,
                               GtkCellRenderer       *cell,
                               gboolean               expand);
  void (* pack_end)           (GtkCellLayout         *cell_layout,
                               GtkCellRenderer       *cell,
                               gboolean               expand);
  void (* clear)              (GtkCellLayout         *cell_layout);
  void (* add_attribute)      (GtkCellLayout         *cell_layout,
                               GtkCellRenderer       *cell,
                               const gchar           *attribute,
                               gint                   column);
  void (* set_cell_data_func) (GtkCellLayout         *cell_layout,
                               GtkCellRenderer       *cell,
                               GtkCellLayoutDataFunc  func,
                               gpointer               func_data,
                               GDestroyNotify         destroy);
  void (* clear_attributes)   (GtkCellLayout         *cell_layout,
                               GtkCellRenderer       *cell);
  void (* reorder)            (GtkCellLayout         *cell_layout,
                               GtkCellRenderer       *cell,
                               gint                   position);
} GtkCellLayoutIface;
```

GtkCellLayoutDataFunc ()

```
void        (*GtkCellLayoutDataFunc)        (GtkCellLayout *cell_layout,
                                             GtkCellRenderer *cell,
                                             GtkTreeModel *tree_model,
                                             GtkTreeIter *iter,
                                             gpointer data);
```

A function which should set the value of *cell_layout*'s cell renderer(s) as appropriate.

cell_layout :	a GtkCellLayout
cell :	the cell renderer whose value is to be set
tree_model :	the model
iter :	a GtkTreeIter indicating the row to set the value for
data :	user data passed to gtk_cell_layout_set_cell_data_func()

---

gtk_cell_layout_pack_start ()

```
void        gtk_cell_layout_pack_start      (GtkCellLayout *cell_layout,
                                             GtkCellRenderer *cell,
                                             gboolean expand);
```

Packs the *cell* into the beginning of *cell_layout*. If *expand* is FALSE, then the *cell* is allocated no more space than it needs. Any unused space is divided evenly between cells for which *expand* is TRUE.

cell_layout :	A GtkCellLayout.
cell :	A GtkCellRenderer.
expand :	TRUE if *cell* is to be given extra space allocated to *cell_layout*.

Since 2.4

---

gtk_cell_layout_pack_end ()

```
void        gtk_cell_layout_pack_end        (GtkCellLayout *cell_layout,
                                             GtkCellRenderer *cell,
                                             gboolean expand);
```

Adds the *cell* to the end of *cell_layout*. If *expand* is FALSE, then the *cell* is allocated no more space than it needs. Any unused space is divided evenly between cells for which *expand* is TRUE.

cell_layout :	A GtkCellLayout.

cell : A GtkCellRenderer.

expand : TRUE if *cell* is to be given extra space allocated to *cell_layout*.

Since 2.4

---

gtk_cell_layout_reorder ()

```
void        gtk_cell_layout_reorder         (GtkCellLayout *cell_layout,
                                             GtkCellRenderer *cell,
                                             gint position);
```

Re-inserts *cell* at *position*. Note that *cell* has already to be packed into *cell_layout* for this to function properly.

cell_layout : A GtkCellLayout.

cell : A GtkCellRenderer to reorder.

position : New position to insert *cell* at.

Since 2.4

---

gtk_cell_layout_clear ()

```
void        gtk_cell_layout_clear           (GtkCellLayout *cell_layout);
```

Unsets all the mappings on all renderers on *cell_layout* and removes all renderers from *cell_layout*.

cell_layout : A GtkCellLayout.

Since 2.4

---

gtk_cell_layout_set_attributes ()

```
void        gtk_cell_layout_set_attributes  (GtkCellLayout *cell_layout,
                                             GtkCellRenderer *cell,
                                             ...);
```

Sets the attributes in list as the attributes of *cell_layout*. The attributes should be in attribute/column order, as in gtk_cell_layout_add_attribute(). All existing attributes are removed, and replaced with the new attributes.

cell_layout : A GtkCellLayout.
cell : A GtkCellRenderer.
... : A NULL-terminated list of attributes.

Since 2.4

---

gtk_cell_layout_add_attribute ()

```
void        gtk_cell_layout_add_attribute   (GtkCellLayout *cell_layout,
                                             GtkCellRenderer *cell,
                                             const gchar *attribute,
                                             gint column);
```

Adds an attribute mapping to the list in *cell_layout*. The *column* is the column of the model to get a value from, and the *attribute* is the parameter on *cell* to be set from the value. So for example if column 2 of the model contains strings, you could have the "text" attribute of a GtkCellRendererText get its values from column 2.

cell_layout : A GtkCellLayout.
cell : A GtkCellRenderer.
attribute : An attribute on the renderer.
column : The column position on the model to get the attribute from.

Since 2.4

---

gtk_cell_layout_set_cell_data_func ()

```
void        gtk_cell_layout_set_cell_data_func
                                            (GtkCellLayout *cell_layout,
                                             GtkCellRenderer *cell,
                                             GtkCellLayoutDataFunc func,
                                             gpointer func_data,
                                             GDestroyNotify destroy);
```

Sets the GtkCellLayoutDataFunc to use for *cell_layout*. This function is used instead of the standard attributes mapping for setting the column value, and should set the value of *cell_layout*'s cell renderer(s) as appropriate. *func* may be NULL to remove and older one.

cell_layout : A GtkCellLayout.
cell : A GtkCellRenderer.
func : The GtkCellLayoutDataFunc to use.
func_data : The user data for *func*.
destroy : The destroy notification for *func_data*.

Since 2.4

---

gtk_cell_layout_clear_attributes ()

```
void        gtk_cell_layout_clear_attributes
                                            (GtkCellLayout *cell_layout,
                                             GtkCellRenderer *cell);
```

Clears all existing attributes previously set with gtk_cell_layout_set_attributes().

cell_layout : A GtkCellLayout.
cell : A GtkCellRenderer to clear the attribute mapping on.

Since 2.4

<< GtkTreeModelFilter **GtkCellRenderer >>**

GtkCellRenderer

GtkCellRenderer — An object for rendering a single cell on a GdkDrawable

Synopsis

```
#include <gtk/gtk.h>

enum        GtkCellRendererState;
enum        GtkCellRendererMode;
            GtkCellRenderer;
void        gtk_cell_renderer_get_size      (GtkCellRenderer *cell,
                                             GtkWidget *widget,
                                             GdkRectangle *cell_area,
                                             gint *x_offset,
                                             gint *y_offset,
                                             gint *width,
                                             gint *height);
void        gtk_cell_renderer_render        (GtkCellRenderer *cell,
                                             GdkWindow *window,
                                             GtkWidget *widget,
                                             GdkRectangle *background_area,
                                             GdkRectangle *cell_area,
                                             GdkRectangle *expose_area,
                                             GtkCellRendererState flags);
gboolean    gtk_cell_renderer_activate      (GtkCellRenderer *cell,
                                             GdkEvent *event,
                                             GtkWidget *widget,
                                             const gchar *path,
                                             GdkRectangle *background_area,
                                             GdkRectangle *cell_area,
                                             GtkCellRendererState flags);
GtkCellEditable* gtk_cell_renderer_start_editing
                                            (GtkCellRenderer *cell,
                                             GdkEvent *event,
```

```
                                                GtkWidget *widget,
                                                const gchar *path,
                                                GdkRectangle *background_area,
                                                GdkRectangle *cell_area,
                                                GtkCellRendererState flags);
void        gtk_cell_renderer_editing_canceled
                                               (GtkCellRenderer *cell);
void        gtk_cell_renderer_get_fixed_size
                                               (GtkCellRenderer *cell,
                                                gint *width,
                                                gint *height);
void        gtk_cell_renderer_set_fixed_size
                                               (GtkCellRenderer *cell,
                                                gint width,
                                                gint height);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkCellRenderer
               +----GtkCellRendererText
               +----GtkCellRendererPixbuf
               +----GtkCellRendererToggle
               +----GtkCellRendererProgress
```

Properties

```
  "cell-background"      gchararray           : Write
  "cell-background-gdk"  GdkColor             : Read / Write
  "cell-background-set"  gboolean             : Read / Write
  "height"               gint                 : Read / Write
  "is-expanded"          gboolean             : Read / Write
  "is-expander"          gboolean             : Read / Write
  "mode"                 GtkCellRendererMode  : Read / Write
```

```
  "sensitive"           gboolean            : Read / Write
  "visible"             gboolean            : Read / Write
  "width"               gint                : Read / Write
  "xalign"              gfloat              : Read / Write
  "xpad"                guint               : Read / Write
  "yalign"              gfloat              : Read / Write
  "ypad"                guint               : Read / Write
```

Signal Prototypes

```
"editing-canceled"
            void        user_function      (GtkCellRenderer *renderer,
                                            gpointer user_data);
"editing-started"
            void        user_function      (GtkCellRenderer *renderer,
                                            GtkCellEditable *editable,
                                            gchar *path,
                                            gpointer user_data);
```

Description

The GtkCellRenderer is a base class of a set of objects used for rendering a cell to a GdkDrawable. These objects are used primarily by the GtkTreeView widget, though they aren't tied to them in any specific way. It is worth noting that GtkCellRenderer is not a GtkWidget and cannot be treated as such.

The primary use of a GtkCellRenderer is for drawing a certain graphical elements on a GdkDrawable. Typically, one cell renderer is used to draw many cells on the screen. To this extent, it isn't expected that a CellRenderer keep any permanent state around. Instead, any state is set just prior to use using GObjects property system. Then, the cell is measured using *gtk_cell_renderer_get_size*. Finally, the cell is rendered in the correct location using *gtk_cell_renderer_render*.

There are a number of rules that must be followed when writing a new GtkCellRenderer. First and formost, it's important that a certain set of properties will always yield a cell renderer of the same size, barring a GtkStyle change. The GtkCellRenderer also has a number of generic properties that are expected to be honored by all children.

Details

enum GtkCellRendererState

```
typedef enum
{
  GTK_CELL_RENDERER_SELECTED    = 1 << 0,
  GTK_CELL_RENDERER_PRELIT      = 1 << 1,
  GTK_CELL_RENDERER_INSENSITIVE = 1 << 2,
  /* this flag means the cell is in the sort column/row */
  GTK_CELL_RENDERER_SORTED      = 1 << 3,
  GTK_CELL_RENDERER_FOCUSED     = 1 << 4
} GtkCellRendererState;
```

Tells how a cell is to be rendererd.

GTK_CELL_RENDERER_SELECTED	The cell is currently selected, and probably has a selection colored background to render to.
GTK_CELL_RENDERER_PRELIT	The mouse is hovering over the cell.
GTK_CELL_RENDERER_INSENSITIVE	The cell is drawn in an insensitive manner
GTK_CELL_RENDERER_SORTED	The cell is in a sorted row
GTK_CELL_RENDERER_FOCUSED	

---

enum GtkCellRendererMode

```
typedef enum
{
  GTK_CELL_RENDERER_MODE_INERT,
  GTK_CELL_RENDERER_MODE_ACTIVATABLE,
  GTK_CELL_RENDERER_MODE_EDITABLE
} GtkCellRendererMode;
```

Identifies how the user can interact with a particular cell.

GTK_CELL_RENDERER_MODE_INERT	The cell is just for display and cannot be interacted with. Note that this doesn't mean that eg. the row being drawn can't be selected -- just that a particular element of it cannot be individually modified.
GTK_CELL_RENDERER_MODE_ACTIVATABLE	The cell can be clicked.
GTK_CELL_RENDERER_MODE_EDITABLE	The cell can be edited or otherwise modified.

GtkCellRenderer

```
typedef struct _GtkCellRenderer GtkCellRenderer;
```

gtk_cell_renderer_get_size ()

```
void        gtk_cell_renderer_get_size      (GtkCellRenderer *cell,
                                             GtkWidget *widget,
                                             GdkRectangle *cell_area,
                                             gint *x_offset,
                                             gint *y_offset,
                                             gint *width,
                                             gint *height);
```

Obtains the width and height needed to render the cell. Used by view widgets to determine the appropriate size for the cell_area passed to gtk_cell_renderer_render(). If *cell_area* is not NULL, fills in the x and y offsets (if set) of the cell relative to this location. Please note that the values set in *width* and *height*, as well as those in *x_offset* and *y_offset* are inclusive of the xpad and ypad properties.

cell :	a GtkCellRenderer
widget :	the widget the renderer is rendering to
cell_area :	The area a cell will be allocated, or NULL
x_offset :	location to return x offset of cell relative to *cell_area*, or NULL
y_offset :	location to return y offset of cell relative to *cell_area*, or NULL
width :	location to return width needed to render a cell, or NULL
height :	location to return height needed to render a cell, or NULL

gtk_cell_renderer_render ()

```
void        gtk_cell_renderer_render        (GtkCellRenderer *cell,
                                             GdkWindow *window,
                                             GtkWidget *widget,
```

```
                                                GdkRectangle *background_area,
                                                GdkRectangle *cell_area,
                                                GdkRectangle *expose_area,
                                                GtkCellRendererState flags);
```

Invokes the virtual render function of the GtkCellRenderer. The three passed-in rectangles are areas of *window*. Most renderers will draw within *cell_area*; the xalign, yalign, xpad, and ypad fields of the GtkCellRenderer should be honored with respect to *cell_area*. *background_area* includes the blank space around the cell, and also the area containing the tree expander; so the *background_area* rectangles for all cells tile to cover the entire *window*. *expose_area* is a clip rectangle.

cell :	a GtkCellRenderer
window :	a GdkDrawable to draw to
widget :	the widget owning *window*
background_area :	entire cell area (including tree expanders and maybe padding on the sides)
cell_area :	area normally rendered by a cell renderer
expose_area :	area that actually needs updating
flags :	flags that affect rendering

---

gtk_cell_renderer_activate ()

```
gboolean    gtk_cell_renderer_activate      (GtkCellRenderer *cell,
                                             GdkEvent *event,
                                             GtkWidget *widget,
                                             const gchar *path,
                                             GdkRectangle *background_area,
                                             GdkRectangle *cell_area,
                                             GtkCellRendererState flags);
```

Passes an activate event to the cell renderer for possible processing. Some cell renderers may use events; for example, GtkCellRendererToggle toggles when it gets a mouse click.

cell :	a GtkCellRenderer
event :	a GdkEvent
widget :	widget that received the event
path :	widget-dependent string representation of the event location; e.g. for GtkTreeView, a string representation of GtkTreePath

background_area :	background area as passed to *gtk_cell_renderer_render*
cell_area :	cell area as passed to *gtk_cell_renderer_render*
flags :	render flags
Returns :	TRUE if the event was consumed/handled

---

gtk_cell_renderer_start_editing ()

```
GtkCellEditable* gtk_cell_renderer_start_editing
                                                (GtkCellRenderer *cell,
                                                 GdkEvent *event,
                                                 GtkWidget *widget,
                                                 const gchar *path,
                                                 GdkRectangle *background_area,
                                                 GdkRectangle *cell_area,
                                                 GtkCellRendererState flags);
```

Passes an activate event to the cell renderer for possible processing.

cell :	a GtkCellRenderer
event :	a GdkEvent
widget :	widget that received the event
path :	widget-dependent string representation of the event location; e.g. for GtkTreeView, a string representation of GtkTreePath
background_area :	background area as passed to *gtk_cell_renderer_render*
cell_area :	cell area as passed to *gtk_cell_renderer_render*
flags :	render flags
Returns :	A new GtkCellEditable, or NULL

---

gtk_cell_renderer_editing_canceled ()

```
void        gtk_cell_renderer_editing_canceled
                                                (GtkCellRenderer *cell);
```

Causes the cell renderer to emit the "editing-canceled" signal. This function is for use only by implementations of cell renderers that need to notify the client program that an editing process was canceled and the changes were not

committed.

cell : A GtkCellRenderer

Since 2.4

---

gtk_cell_renderer_get_fixed_size ()

```
void        gtk_cell_renderer_get_fixed_size
                                            (GtkCellRenderer *cell,
                                             gint *width,
                                             gint *height);
```

Fills in *width* and *height* with the appropriate size of *cell*.

cell : A GtkCellRenderer
width : location to fill in with the fixed width of the widget, or NULL
height : location to fill in with the fixed height of the widget, or NULL

---

gtk_cell_renderer_set_fixed_size ()

```
void        gtk_cell_renderer_set_fixed_size
                                            (GtkCellRenderer *cell,
                                             gint width,
                                             gint height);
```

Sets the renderer size to be explicit, independent of the properties set.

cell : A GtkCellRenderer
width : the width of the cell renderer, or -1
height : the height of the cell renderer, or -1

Properties

The "cell-background" property

"cell-background"	gchararray	: Write

Cell background color as a string.

Default value: NULL

The "cell-background-gdk" property

"cell-background-gdk"	GdkColor	: Read / Write

Cell background color as a GdkColor.

The "cell-background-set" property

"cell-background-set"	gboolean	: Read / Write

Whether this tag affects the cell background color.

Default value: FALSE

The "height" property

"height"	gint	: Read / Write

The fixed height.

Allowed values: >= -1

Default value: -1

The "is-expanded" property

```
"is-expanded"          gboolean              : Read / Write
```

Row is an expander row, and is expanded.

Default value: FALSE

The "is-expander" property

```
"is-expander"          gboolean              : Read / Write
```

Row has children.

Default value: FALSE

The "mode" property

```
"mode"                 GtkCellRendererMode   : Read / Write
```

Editable mode of the CellRenderer.

Default value: GTK_CELL_RENDERER_MODE_INERT

The "sensitive" property

```
"sensitive"            gboolean              : Read / Write
```

Display the cell sensitive.

Default value: TRUE

The "visible" property

```
  "visible"              gboolean              : Read / Write
```

Display the cell.

Default value: TRUE

The "width" property

```
  "width"                gint                  : Read / Write
```

The fixed width.

Allowed values: >= -1

Default value: -1

The "xalign" property

```
  "xalign"               gfloat                : Read / Write
```

The x-align.

Allowed values: [0,1]

Default value: 0.5

The "xpad" property

```
  "xpad"                guint                 : Read / Write
```

The xpad.

Default value: 0

The "yalign" property

```
  "yalign"              gfloat                : Read / Write
```

The y-align.

Allowed values: [0,1]

Default value: 0.5

The "ypad" property

```
  "ypad"                guint                 : Read / Write
```

The ypad.

Default value: 0

Signals

The "editing-canceled" signal

```
void        user_function                   (GtkCellRenderer *renderer,
                                             gpointer user_data);
```

This signal gets emitted when the user cancels the process of editing a cell. For example, an editable cell renderer could be written to cancel editing when the user presses Escape.

See also: `gtk_cell_renderer_editing_canceled()`

renderer : the object which received the signal
user_data : user data set when the signal handler was connected.

Since 2.4

The "editing-started" signal

```
void        user_function                   (GtkCellRenderer *renderer,
                                             GtkCellEditable *editable,
                                             gchar *path,
                                             gpointer user_data);
```

This signal gets emitted when a cell starts to be edited. The indended use of this signal is to do special setup on *editable*, e.g. adding a GtkEntryCompletion or setting up additional columns in a GtkComboBox.

Note that GTK+ doesn't guarantee that cell renderers will continue to use the same kind of widget for editing in future releases, therefore you should check the type of *editable* before doing any specific setup, as in the following example:

```
static void
text_editing_started (GtkCellRenderer *cell,
                      GtkCellEditable *editable,
                      const gchar     *path,
                      gpointer         data)
{
  if (GTK_IS_ENTRY (editable))
    {
      GtkEntry *entry = GTK_ENTRY (editable);

      /* ... create a GtkEntryCompletion */

      gtk_entry_set_completion (entry, completion);
    }
}
```

renderer : the object which received the signal
editable : the GtkCellEditable
path : the path identifying the edited cell
user_data : user data set when the signal handler was connected.

Since 2.6

See Also

GtkCellRendererText,GtkCellRendererPixbuf,GtkCellRendererToggle

<< GtkCellLayout **GtkCellEditable >>**

GtkCellEditable

GtkCellEditable — Interface for widgets which can are used for editing cells

Synopsis

```
#include <gtk/gtk.h>

            GtkCellEditable;
            GtkCellEditableIface;
void        gtk_cell_editable_start_editing (GtkCellEditable *cell_editable,
                                             GdkEvent *event);
void        gtk_cell_editable_editing_done  (GtkCellEditable *cell_editable);
void        gtk_cell_editable_remove_widget (GtkCellEditable *cell_editable);
```

Object Hierarchy

```
  GInterface
   +----GtkCellEditable
```

Prerequisites

GtkCellEditable requires GtkWidget.

Known Implementations

GtkCellEditable is implemented by GtkEntry, GtkComboBox, GtkSpinButton and GtkComboBoxEntry.

Signal Prototypes

```
"editing-done"
            void        user_function      (GtkCellEditable *celleditable,
                                            gpointer user_data);
"remove-widget"
            void        user_function      (GtkCellEditable *celleditable,
                                            gpointer user_data);
```

Description

Details

GtkCellEditable

```
typedef struct _GtkCellEditable GtkCellEditable;
```

GtkCellEditableIface

```
typedef struct {
  GTypeInterface g_iface;

  /* signals */
  void (* editing_done)  (GtkCellEditable *cell_editable);
  void (* remove_widget) (GtkCellEditable *cell_editable);

  /* virtual table */
  void (* start_editing) (GtkCellEditable *cell_editable,
                          GdkEvent        *event);
} GtkCellEditableIface;
```

---

gtk_cell_editable_start_editing ()

```
void        gtk_cell_editable_start_editing (GtkCellEditable *cell_editable,
                                             GdkEvent *event);
```

Begins editing on a *cell_editable*. *event* is the GdkEvent that began the editing process. It may be NULL, in the instance that editing was initiated through programatic means.

cell_editable : A GtkCellEditable
event : A GdkEvent, or NULL

---

gtk_cell_editable_editing_done ()

```
void        gtk_cell_editable_editing_done  (GtkCellEditable *cell_editable);
```

Emits the "editing_done" signal. This signal is a sign for the cell renderer to update its value from the cell.

cell_editable : A GtkTreeEditable

---

gtk_cell_editable_remove_widget ()

```
void        gtk_cell_editable_remove_widget (GtkCellEditable *cell_editable);
```

Emits the "remove_widget" signal. This signal is meant to indicate that the cell is finished editing, and the widget may now be destroyed.

cell_editable : A GtkTreeEditable

Signals

The "editing-done" signal

```
void        user_function                  (GtkCellEditable *celleditable,
                                            gpointer user_data);
```

celleditable : the object which received the signal.
user_data : user data set when the signal handler was connected.

The "remove-widget" signal

```
void        user_function                   (GtkCellEditable *celleditable,
                                             gpointer user_data);
```

celleditable : the object which received the signal.

user_data : user data set when the signal handler was connected.

<< GtkCellRenderer **GtkCellRendererCombo >>**

GtkCellRendererCombo

GtkCellRendererCombo —

Synopsis

```
#include <gtk/gtk.h>

            GtkCellRendererCombo;
GtkCellRenderer* gtk_cell_renderer_combo_new
                                                    (void);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkCellRenderer
               +----GtkCellRendererText
                     +----GtkCellRendererCombo
```

Properties

```
  "has-entry"              gboolean              : Read / Write
  "model"                  GtkTreeModel          : Read / Write
  "text-column"            gint                  : Read / Write
```

Description

Details

GtkCellRendererCombo

```
typedef struct _GtkCellRendererCombo GtkCellRendererCombo;
```

---

gtk_cell_renderer_combo_new ()

```
GtkCellRenderer* gtk_cell_renderer_combo_new
                                                (void);
```

Creates a new GtkCellRendererCombo Adjust how text is drawn using object properties. Object properties can be set globally (with `g_object_set()`). Also, with GtkTreeViewColumn, you can bind a property to a value in a GtkTreeModel. For example, you can bind the "text" property on the cell renderer to a string value in the model, thus rendering a different string in each row of the GtkTreeView.

Returns : the new cell renderer

Since 2.6

Properties

The "has-entry" property

```
  "has-entry"           gboolean              : Read / Write
```

If %FALSE, don't allow to enter strings other than the chosen ones.

Default value: TRUE

The "model" property

"model"	GtkTreeModel	: Read / Write

The :model property holds a tree model containing the possible values for the combo box. Use the :text_column property to specify the column holding the values.

Since 2.6

The "text-column" property

"text-column"	gint	: Read / Write

A column in the data source model to get the strings from.

Allowed values: >= -1

Default value: -1

<< GtkCellEditable **GtkCellRendererPixbuf >>**

GtkCellRendererPixbuf

GtkCellRendererPixbuf — Renders a pixbuf in a cell

Synopsis

```
#include <gtk/gtk.h>

            GtkCellRendererPixbuf;
GtkCellRenderer* gtk_cell_renderer_pixbuf_new
                                                (void);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkCellRenderer
               +----GtkCellRendererPixbuf
```

Properties

```
  "pixbuf"                GdkPixbuf             : Read / Write
  "pixbuf-expander-closed" GdkPixbuf              : Read / Write
  "pixbuf-expander-open" GdkPixbuf             : Read / Write
  "stock-detail"          gchararray            : Read / Write
```

```
  "stock-id"            gchararray          : Read / Write
  "stock-size"          guint               : Read / Write
```

Description

Details

GtkCellRendererPixbuf

```
typedef struct _GtkCellRendererPixbuf GtkCellRendererPixbuf;
```

---

gtk_cell_renderer_pixbuf_new ()

```
GtkCellRenderer* gtk_cell_renderer_pixbuf_new
                                                (void);
```

Creates a new GtkCellRendererPixbuf. Adjust rendering parameters using object properties. Object properties can be set globally (with `g_object_set()`). Also, with GtkTreeViewColumn, you can bind a property to a value in a GtkTreeModel. For example, you can bind the "pixbuf" property on the cell renderer to a pixbuf value in the model, thus rendering a different image in each row of the GtkTreeView.

Returns : the new cell renderer

Properties

The "pixbuf" property

```
  "pixbuf"              GdkPixbuf           : Read / Write
```

The pixbuf to render.

The "pixbuf-expander-closed" property

```
"pixbuf-expander-closed" GdkPixbuf             : Read / Write
```

Pixbuf for closed expander.

The "pixbuf-expander-open" property

```
"pixbuf-expander-open" GdkPixbuf               : Read / Write
```

Pixbuf for open expander.

The "stock-detail" property

```
"stock-detail"         gchararray              : Read / Write
```

Render detail to pass to the theme engine.

Default value: NULL

The "stock-id" property

```
"stock-id"             gchararray              : Read / Write
```

The stock ID of the stock icon to render.

Default value: NULL

The "stock-size" property

```
  "stock-size"           guint                 : Read / Write
```

The GtkIconSize value that specifies the size of the rendered icon.

Default value: 1

<< GtkCellRendererCombo **GtkCellRendererProgress >>**

GtkCellRendererProgress

GtkCellRendererProgress — Renders numbers as progress bars

Synopsis

```
#include <gtk/gtk.h>

            GtkCellRendererProgress;
GtkCellRenderer* gtk_cell_renderer_progress_new
                                                (void);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkCellRenderer
               +----GtkCellRendererProgress
```

Properties

```
  "text"                 gchararray            : Read / Write
  "value"                gint                  : Read / Write
```

Description

Details

GtkCellRendererProgress

```
typedef struct _GtkCellRendererProgress GtkCellRendererProgress;
```

---

gtk_cell_renderer_progress_new ()

```
GtkCellRenderer* gtk_cell_renderer_progress_new
                                                (void);
```

Creates a new GtkCellRendererProgress.

Returns : the new cell renderer

Since 2.6

Properties

The "text" property

```
  "text"                 gchararray           : Read / Write
```

The "text" property determines the label which will be drawn over the progress bar. Setting this property to NULL causes the default label to be displayed. Setting this property to an empty string causes no label to be displayed.

Default value: NULL

Since 2.6

The "value" property

"value"	gint	: Read / Write

The "value" property determines the percentage to which the progress bar will be "filled in".

Allowed values: [0,100]

Default value: 0

Since 2.6

<< GtkCellRendererPixbuf **GtkCellRendererText >>**

GtkCellRendererText

GtkCellRendererText — Renders text in a cell

Synopsis

```
#include <gtk/gtk.h>

            GtkCellRendererText;
GtkCellRenderer* gtk_cell_renderer_text_new (void);
void        gtk_cell_renderer_text_set_fixed_height_from_font
                                            (GtkCellRendererText *renderer,
                                             gint number_of_rows);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkCellRenderer
               +----GtkCellRendererText
                     +----GtkCellRendererCombo
```

Properties

```
  "attributes"           PangoAttrList         : Read / Write
  "background"           gchararray            : Write
  "background-gdk"       GdkColor              : Read / Write
  "background-set"       gboolean              : Read / Write
  "editable"             gboolean              : Read / Write
  "editable-set"         gboolean              : Read / Write
  "ellipsize"            PangoEllipsizeMode    : Read / Write
  "ellipsize-set"        gboolean              : Read / Write
```

```
  "family"              gchararray            : Read / Write
  "family-set"          gboolean              : Read / Write
  "font"                gchararray            : Read / Write
  "font-desc"           PangoFontDescription  : Read / Write
  "foreground"          gchararray            : Write
  "foreground-gdk"      GdkColor              : Read / Write
  "foreground-set"      gboolean              : Read / Write
  "language"            gchararray            : Read / Write
  "language-set"        gboolean              : Read / Write
  "markup"              gchararray            : Write
  "rise"                gint                  : Read / Write
  "rise-set"            gboolean              : Read / Write
  "scale"               gdouble               : Read / Write
  "scale-set"           gboolean              : Read / Write
  "single-paragraph-mode" gboolean             : Read / Write
  "size"                gint                  : Read / Write
  "size-points"         gdouble               : Read / Write
  "size-set"            gboolean              : Read / Write
  "stretch"             PangoStretch          : Read / Write
  "stretch-set"         gboolean              : Read / Write
  "strikethrough"       gboolean              : Read / Write
  "strikethrough-set"   gboolean              : Read / Write
  "style"               PangoStyle            : Read / Write
  "style-set"           gboolean              : Read / Write
  "text"                gchararray            : Read / Write
  "underline"           PangoUnderline        : Read / Write
  "underline-set"       gboolean              : Read / Write
  "variant"             PangoVariant          : Read / Write
  "variant-set"         gboolean              : Read / Write
  "weight"              gint                  : Read / Write
  "weight-set"          gboolean              : Read / Write
```

Signal Prototypes

```
"edited"    void        user_function      (GtkCellRendererText *cellrenderertext,
                                            gchar *arg1,
                                            gchar *arg2,
                                            gpointer user_data);
```

Description

Details

GtkCellRendererText

```
typedef struct _GtkCellRendererText GtkCellRendererText;
```

---

gtk_cell_renderer_text_new ()

```
GtkCellRenderer* gtk_cell_renderer_text_new (void);
```

Creates a new GtkCellRendererText. Adjust how text is drawn using object properties. Object properties can be set globally (with `g_object_set()`). Also, with GtkTreeViewColumn, you can bind a property to a value in a GtkTreeModel. For example, you can bind the "text" property on the cell renderer to a string value in the model, thus rendering a different string in each row of the GtkTreeView

Returns : the new cell renderer

---

gtk_cell_renderer_text_set_fixed_height_from_font ()

```
void        gtk_cell_renderer_text_set_fixed_height_from_font
                                            (GtkCellRendererText *renderer,
                                             gint number_of_rows);
```

Sets the height of a renderer to explicitly be determined by the "font" and "y_pad" property set on it. Further changes in these properties do not affect the height, so they must be accompanied by a subsequent call to this function. Using this function is unflexible, and should really only be used if calculating the size of a cell is too slow (ie, a massive number of cells displayed). If *`number_of_rows`* is -1, then the fixed height is unset, and the height is determined by the properties again.

`renderer` : A GtkCellRendererText

`number_of_rows` : Number of rows of text each cell renderer is allocated, or -1

Properties

The "`attributes`" property

```
"attributes"            PangoAttrList           : Read / Write
```

A list of style attributes to apply to the text of the renderer.

The "background" property

```
"background"            gchararray              : Write
```

Background color as a string.

Default value: NULL

The "background-gdk" property

```
"background-gdk"        GdkColor                : Read / Write
```

Background color as a GdkColor.

The "background-set" property

```
"background-set"        gboolean                : Read / Write
```

Whether this tag affects the background color.

Default value: FALSE

The "editable" property

```
"editable"              gboolean                : Read / Write
```

Whether the text can be modified by the user.

Default value: FALSE

The "editable-set" property

```
"editable-set"         gboolean              : Read / Write
```

Whether this tag affects text editability.

Default value: FALSE

The "ellipsize" property

```
"ellipsize"            PangoEllipsizeMode    : Read / Write
```

The preferred place to ellipsize the string, if the cell renderer does not have enough room to display the entire string, if at all.

Default value: PANGO_ELLIPSIZE_NONE

The "ellipsize-set" property

```
"ellipsize-set"        gboolean              : Read / Write
```

Whether this tag affects the ellipsize mode.

Default value: FALSE

The "family" property

```
"family"               gchararray            : Read / Write
```

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

Default value: NULL

The "family-set" property

```
"family-set"           gboolean              : Read / Write
```

Whether this tag affects the font family.

Default value: FALSE

The "font" property

```
"font"                 gchararray            : Read / Write
```

Font description as a string.

Default value: NULL

The "font-desc" property

```
"font-desc"            PangoFontDescription  : Read / Write
```

Font description as a PangoFontDescription struct.

The "foreground" property

```
"foreground"           gchararray            : Write
```

Foreground color as a string.

Default value: NULL

The "foreground-gdk" property

```
"foreground-gdk"       GdkColor              : Read / Write
```

Foreground color as a GdkColor.

The "foreground-set" property

```
"foreground-set"       gboolean              : Read / Write
```

Whether this tag affects the foreground color.

Default value: FALSE

The "language" property

```
"language"             gchararray            : Read / Write
```

The language this text is in, as an ISO code. Pango can use this as a hint when rendering the text. If you don't understand this parameter, you probably don't need it.

Default value: NULL

The "language-set" property

```
"language-set"         gboolean              : Read / Write
```

Whether this tag affects the language the text is rendered as.

Default value: FALSE

The "markup" property

```
"markup"               gchararray            : Write
```

Marked up text to render.

Default value: NULL

The "rise" property

```
"rise"                 gint                  : Read / Write
```

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

Allowed values: >= -2147483647

Default value: 0

The "rise-set" property

```
"rise-set"             gboolean              : Read / Write
```

Whether this tag affects the rise.

Default value: FALSE

The "scale" property

```
"scale"                gdouble               : Read / Write
```

Font scaling factor.

Allowed values: >= 0

Default value: 1

The "scale-set" property

```
"scale-set"            gboolean              : Read / Write
```

Whether this tag scales the font size by a factor.

Default value: FALSE

The "single-paragraph-mode" property

```
"single-paragraph-mode" gboolean              : Read / Write
```

Whether or not to keep all text in a single paragraph.

Default value: FALSE

The "size" property

```
"size"                 gint                  : Read / Write
```

Font size.

Allowed values: >= 0

Default value: 0

The "size-points" property

```
"size-points"          gdouble               : Read / Write
```

Font size in points.

Allowed values: >= 0

Default value: 0

The "size-set" property

"size-set"	gboolean	: Read / Write

Whether this tag affects the font size.

Default value: FALSE

The "stretch" property

"stretch"	PangoStretch	: Read / Write

Font stretch.

Default value: PANGO_STRETCH_NORMAL

The "stretch-set" property

"stretch-set"	gboolean	: Read / Write

Whether this tag affects the font stretch.

Default value: FALSE

The "strikethrough" property

"strikethrough"	gboolean	: Read / Write

Whether to strike through the text.

Default value: FALSE

The "strikethrough-set" property

```
  "strikethrough-set"    gboolean              : Read / Write
```

Whether this tag affects strikethrough.

Default value: FALSE

The "style" property

```
  "style"                PangoStyle            : Read / Write
```

Font style.

Default value: PANGO_STYLE_NORMAL

The "style-set" property

```
  "style-set"            gboolean              : Read / Write
```

Whether this tag affects the font style.

Default value: FALSE

The "text" property

```
  "text"                 gchararray            : Read / Write
```

Text to render.

Default value: NULL

The "underline" property

```
"underline"             PangoUnderline          : Read / Write
```

Style of underline for this text.

Default value: PANGO_UNDERLINE_NONE

The "underline-set" property

```
"underline-set"         gboolean                : Read / Write
```

Whether this tag affects underlining.

Default value: FALSE

The "variant" property

```
"variant"               PangoVariant            : Read / Write
```

Font variant.

Default value: PANGO_VARIANT_NORMAL

The "variant-set" property

```
"variant-set"           gboolean                : Read / Write
```

Whether this tag affects the font variant.

Default value: FALSE

The "weight" property

```
"weight"                gint                    : Read / Write
```

Font weight.

Allowed values: >= 0

Default value: 400

The "weight-set" property

```
  "weight-set"           gboolean              : Read / Write
```

Whether this tag affects the font weight.

Default value: FALSE

Signals

The "edited" signal

```
void        user_function                  (GtkCellRendererText *cellrenderertext,
                                            gchar *arg1,
                                            gchar *arg2,
                                            gpointer user_data);
```

cellrenderertext : the object which received the signal.
arg1 :
arg2 :
user_data : user data set when the signal handler was connected.

<< GtkCellRendererProgress **GtkCellRendererToggle >>**

GtkCellRendererToggle

GtkCellRendererToggle — Renders a toggle button in a cell

Synopsis

```
#include <gtk/gtk.h>

            GtkCellRendererToggle;
GtkCellRenderer* gtk_cell_renderer_toggle_new
                                            (void);
gboolean    gtk_cell_renderer_toggle_get_radio
                                            (GtkCellRendererToggle *toggle);
void        gtk_cell_renderer_toggle_set_radio
                                            (GtkCellRendererToggle *toggle,
                                             gboolean radio);
gboolean    gtk_cell_renderer_toggle_get_active
                                            (GtkCellRendererToggle *toggle);
void        gtk_cell_renderer_toggle_set_active
                                            (GtkCellRendererToggle *toggle,
                                             gboolean setting);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkCellRenderer
               +----GtkCellRendererToggle
```

Properties

```
  "activatable"          gboolean              : Read / Write
```

```
  "active"              gboolean              : Read / Write
  "inconsistent"        gboolean              : Read / Write
  "radio"               gboolean              : Read / Write
```

Signal Prototypes

```
"toggled"   void        user_function      (GtkCellRendererToggle *cell_renderer,
                                            gchar *path,
                                            gpointer user_data);
```

Description

Details

GtkCellRendererToggle

```
typedef struct _GtkCellRendererToggle GtkCellRendererToggle;
```

---

gtk_cell_renderer_toggle_new ()

```
GtkCellRenderer* gtk_cell_renderer_toggle_new
                                            (void);
```

Creates a new GtkCellRendererToggle. Adjust rendering parameters using object properties. Object properties can be set globally (with `g_object_set()`). Also, with GtkTreeViewColumn, you can bind a property to a value in a GtkTreeModel. For example, you can bind the "active" property on the cell renderer to a boolean value in the model, thus causing the check button to reflect the state of the model.

Returns : the new cell renderer

---

gtk_cell_renderer_toggle_get_radio ()

```
gboolean    gtk_cell_renderer_toggle_get_radio
                                            (GtkCellRendererToggle *toggle);
```

Returns wether we're rendering radio toggles rather than checkboxes.

toggle : a GtkCellRendererToggle

Returns : TRUE if we're rendering radio toggles rather than checkboxes

---

gtk_cell_renderer_toggle_set_radio ()

```
void        gtk_cell_renderer_toggle_set_radio
                                            (GtkCellRendererToggle *toggle,
                                             gboolean radio);
```

If *radio* is TRUE, the cell renderer renders a radio toggle (i.e. a toggle in a group of mutually-exclusive toggles). If FALSE, it renders a check toggle (a standalone boolean option). This can be set globally for the cell renderer, or changed just before rendering each cell in the model (for GtkTreeView, you set up a per-row setting using GtkTreeViewColumn to associate model columns with cell renderer properties).

toggle : a GtkCellRendererToggle

radio : TRUE to make the toggle look like a radio button

---

gtk_cell_renderer_toggle_get_active ()

```
gboolean    gtk_cell_renderer_toggle_get_active
                                            (GtkCellRendererToggle *toggle);
```

Returns whether the cell renderer is active. See `gtk_cell_renderer_toggle_set_active()`.

toggle : a GtkCellRendererToggle

Returns : TRUE if the cell renderer is active.

---

gtk_cell_renderer_toggle_set_active ()

```
void        gtk_cell_renderer_toggle_set_active
                                            (GtkCellRendererToggle *toggle,
                                             gboolean setting);
```

Activates or deactivates a cell renderer.

toggle : a GtkCellRendererToggle.
setting : the value to set.

Properties

The "activatable" property

```
"activatable"          gboolean              : Read / Write
```

The toggle button can be activated.

Default value: TRUE

The "active" property

```
"active"               gboolean              : Read / Write
```

The toggle state of the button.

Default value: FALSE

The "inconsistent" property

```
"inconsistent"         gboolean              : Read / Write
```

The inconsistent state of the button.

Default value: FALSE

The "radio" property

```
"radio"                gboolean              : Read / Write
```

Draw the toggle button as a radio button.

Default value: FALSE

Signals

The "toggled" signal

```
void        user_function                   (GtkCellRendererToggle *cell_renderer,
                                             gchar *path,
                                             gpointer user_data);
```

The ::toggled signal is emitted when the cell is toggled.

cell_renderer :	the object which received the signal
path :	string representation of GtkTreePath describing the event location
user_data :	user data set when the signal handler was connected.

<< GtkCellRendererText **GtkListStore >>**

GtkListStore

GtkListStore — A list-like data structure that can be used with the GtkTreeView

Synopsis

```
#include <gtk/gtk.h>

            GtkListStore;
GtkListStore* gtk_list_store_new            (gint n_columns,
                                             ...);
GtkListStore* gtk_list_store_newv           (gint n_columns,
                                             GType *types);
void        gtk_list_store_set_column_types (GtkListStore *list_store,
                                             gint n_columns,
                                             GType *types);
void        gtk_list_store_set              (GtkListStore *list_store,
                                             GtkTreeIter *iter,
                                             ...);
void        gtk_list_store_set_valist       (GtkListStore *list_store,
                                             GtkTreeIter *iter,
                                             va_list var_args);
void        gtk_list_store_set_value        (GtkListStore *list_store,
                                             GtkTreeIter *iter,
                                             gint column,
                                             GValue *value);
gboolean    gtk_list_store_remove           (GtkListStore *list_store,
                                             GtkTreeIter *iter);
void        gtk_list_store_insert           (GtkListStore *list_store,
                                             GtkTreeIter *iter,
                                             gint position);
void        gtk_list_store_insert_before    (GtkListStore *list_store,
                                             GtkTreeIter *iter,
```

```
                                                    GtkTreeIter *sibling);
void        gtk_list_store_insert_after     (GtkListStore *list_store,
                                             GtkTreeIter *iter,
                                             GtkTreeIter *sibling);
void        gtk_list_store_prepend          (GtkListStore *list_store,
                                             GtkTreeIter *iter);
void        gtk_list_store_append           (GtkListStore *list_store,
                                             GtkTreeIter *iter);
void        gtk_list_store_clear            (GtkListStore *list_store);
gboolean    gtk_list_store_iter_is_valid    (GtkListStore *list_store,
                                             GtkTreeIter *iter);
void        gtk_list_store_reorder          (GtkListStore *store,
                                             gint *new_order);
void        gtk_list_store_swap             (GtkListStore *store,
                                             GtkTreeIter *a,
                                             GtkTreeIter *b);
void        gtk_list_store_move_before      (GtkListStore *store,
                                             GtkTreeIter *iter,
                                             GtkTreeIter *position);
void        gtk_list_store_move_after       (GtkListStore *store,
                                             GtkTreeIter *iter,
                                             GtkTreeIter *position);
```

Object Hierarchy

```
  GObject
   +----GtkListStore
```

Implemented Interfaces

GtkListStore implements GtkTreeModel, GtkTreeDragSource, GtkTreeDragDest and GtkTreeSortable.

Description

The GtkListStore object is a list model for use with a GtkTreeView widget. It implements the GtkTreeModel interface, and consequentialy, can use all of the methods available there. It also implements the GtkTreeSortable interface so it can be sorted by the view. Finally, it also implements the tree drag and drop interfaces.

The GtkListStore can accept most GObject types as a column type, though it can't accept all custom types. Internally, it will keep a copy of data passed in (such as a string or a boxed pointer). Columns that accept GObjects are handled a little differently. The GtkListStore will keep a reference to the object instead of copying the value. As a result, if the object is modified, it is up to the application writer to call *gtk_tree_model_row_changed* to emit the "row_changed" signal. This most commonly effects lists with GdkPixbufs stored.

Example 5. Creating a simple list store.

```
enum {
  COLUMN_STRING,
  COLUMN_INT,
  COLUMN_BOOLEAN,
  N_COLUMNS
};

{
  GtkListStore *list_store;
  GtkTreePath *path;
  GtkTreeIter iter;
  gint i;

  list_store = gtk_list_store_new (N_COLUMNS,
                                   G_TYPE_STRING,
                                   G_TYPE_INT,
                                   G_TYPE_BOOLEAN);

  for (i = 0; i < 10; i++)
    {
      gchar *some_data;

      some_data = get_some_data (i);

      /* Add a new row to the model */
      gtk_list_store_append (list_store, &iter);
      gtk_list_store_set (list_store, &iter,
                          COLUMN_STRING, some_data,
                          COLUMN_INT, i,
```

```
                            COLUMN_BOOLEAN,  FALSE,
                            -1);

      /* As the store will keep a copy of the string internally, we
       * free some_data.
       */
      g_free (some_data);
    }

  /* Modify a particular row */
  path = gtk_tree_path_new_from_string ("4");
  gtk_tree_model_get_iter (GTK_TREE_MODEL (list_store),
                           &iter,
                           path);
  gtk_tree_path_free (path);
  gtk_list_store_set (list_store, &iter,
                      COLUMN_BOOLEAN, TRUE,
                      -1);
}
```

Performance Considerations

Internally, the GtkListStore was implemented with a linked list with a tail pointer prior to GTK+ 2.6. As a result, it was fast at data insertion and deletion, and not fast at random data access. The GtkListStore sets the GTK_TREE_MODEL_ITERS_PERSIST flag, which means that GtkTreeIters can be cached while the row exists. Thus, if access to a particular row is needed often and your code is expected to run on older versions of GTK+, it is worth keeping the iter around.

Details

GtkListStore

```
typedef struct _GtkListStore GtkListStore;
```

---

gtk_list_store_new ()

```
GtkListStore* gtk_list_store_new           (gint n_columns,
```

```
                                                ...);
```

Creates a new list store as with *n_columns* columns each of the types passed in. Note that only types derived from standard GObject fundamental types are supported.

As an example, `gtk_tree_store_new (3, G_TYPE_INT, G_TYPE_STRING, GDK_TYPE_PIXBUF);` will create a new GtkListStore with three columns, of type int, string and GdkPixbuf respectively.

n_columns : number of columns in the list store
... : all GType types for the columns, from first to last
Returns : a new GtkListStore

---

gtk_list_store_newv ()

```
GtkListStore* gtk_list_store_newv           (gint n_columns,
                                             GType *types);
```

Non-vararg creation function. Used primarily by language bindings.

n_columns : number of columns in the list store
types : an array of GType types for the columns, from first to last
Returns : a new GtkListStore

---

gtk_list_store_set_column_types ()

```
void        gtk_list_store_set_column_types (GtkListStore *list_store,
                                             gint n_columns,
                                             GType *types);
```

This function is meant primarily for GObjects that inherit from GtkListStore, and should only be used when constructing a new GtkListStore. It will not function after a row has been added, or a method on the GtkTreeModel interface is called.

list_store : A GtkListStore
n_columns : Number of columns for the list store
types : An array length n of GTypes

---

gtk_list_store_set ()

```
void        gtk_list_store_set              (GtkListStore *list_store,
                                             GtkTreeIter *iter,
                                             ...);
```

Sets the value of one or more cells in the row referenced by *iter*. The variable argument list should contain integer column numbers, each column number followed by the value to be set. The list is terminated by a -1. For example, to set column 0 with type `G_TYPE_STRING` to "Foo", you would write `gtk_list_store_set (store, iter, 0, "Foo", -1)`.

list_store : a GtkListStore
iter : row iterator
... : pairs of column number and value, terminated with -1

---

gtk_list_store_set_valist ()

```
void        gtk_list_store_set_valist       (GtkListStore *list_store,
                                             GtkTreeIter *iter,
                                             va_list var_args);
```

See `gtk_list_store_set()`; this version takes a va_list for use by language bindings.

list_store : A GtkListStore
iter : A valid GtkTreeIter for the row being modified
var_args : va_list of column/value pairs

---

gtk_list_store_set_value ()

```
void        gtk_list_store_set_value        (GtkListStore *list_store,
                                             GtkTreeIter *iter,
                                             gint column,
                                             GValue *value);
```

Sets the data in the cell specified by *iter* and *column*. The type of *value* must be convertible to the type of the column.

list_store : A GtkListStore
iter : A valid GtkTreeIter for the row being modified
column : column number to modify
value : new value for the cell

---

gtk_list_store_remove ()

```
gboolean    gtk_list_store_remove           (GtkListStore *list_store,
                                             GtkTreeIter *iter);
```

Removes the given row from the list store. After being removed, *iter* is set to be the next valid row, or invalidated if it pointed to the last row in *list_store*.

list_store : A GtkListStore
iter : A valid GtkTreeIter
Returns : TRUE if *iter* is valid, FALSE if not.

---

gtk_list_store_insert ()

```
void        gtk_list_store_insert           (GtkListStore *list_store,
                                             GtkTreeIter *iter,
                                             gint position);
```

Creates a new row at *position*. *iter* will be changed to point to this new row. If *position* is larger than the number of rows on the list, then the new row will be appended to the list. The row will be empty before this function is called. To fill in values, you need to call gtk_list_store_set() or gtk_list_store_set_value().

list_store : A GtkListStore
iter : An unset GtkTreeIter to set to the new row
position : position to insert the new row

---

gtk_list_store_insert_before ()

```
void        gtk_list_store_insert_before    (GtkListStore *list_store,
                                             GtkTreeIter *iter,
                                             GtkTreeIter *sibling);
```

Inserts a new row before *sibling*. If *sibling* is NULL, then the row will be appended to the end of the list. *iter* will be changed to point to this new row. The row will be empty before this function is called. To fill in values, you need to call gtk_list_store_set() or gtk_list_store_set_value().

list_store : A GtkListStore
iter : An unset GtkTreeIter to set to the new row
sibling : A valid GtkTreeIter, or NULL

---

gtk_list_store_insert_after ()

```
void        gtk_list_store_insert_after     (GtkListStore *list_store,
                                             GtkTreeIter *iter,
                                             GtkTreeIter *sibling);
```

Inserts a new row after *sibling*. If *sibling* is NULL, then the row will be prepended to the beginning of the list. *iter* will be changed to point to this new row. The row will be empty after this function is called. To fill in values, you need to call gtk_list_store_set() or

gtk_list_store_set_value().

list_store : A GtkListStore
iter : An unset GtkTreeIter to set to the new row
sibling : A valid GtkTreeIter, or NULL

---

gtk_list_store_prepend ()

```
void        gtk_list_store_prepend          (GtkListStore *list_store,
                                             GtkTreeIter *iter);
```

Prepends a new row to *list_store*. *iter* will be changed to point to this new row. The row will be empty after this function is called. To fill in values, you need to call gtk_list_store_set() or gtk_list_store_set_value().

list_store : A GtkListStore
iter : An unset GtkTreeIter to set to the prepend row

---

gtk_list_store_append ()

```
void        gtk_list_store_append           (GtkListStore *list_store,
                                             GtkTreeIter *iter);
```

Appends a new row to *list_store*. *iter* will be changed to point to this new row. The row will be empty after this function is called. To fill in values, you need to call gtk_list_store_set() or gtk_list_store_set_value().

list_store : A GtkListStore
iter : An unset GtkTreeIter to set to the appended row

---

gtk_list_store_clear ()

```
void        gtk_list_store_clear            (GtkListStore *list_store);
```

Removes all rows from the list store.

list_store : a GtkListStore.

---

gtk_list_store_iter_is_valid ()

```
gboolean    gtk_list_store_iter_is_valid    (GtkListStore *list_store,
                                             GtkTreeIter *iter);
```

WARNING: This function is slow. Only use it for debugging and/or testing purposes.

Checks if the given iter is a valid iter for this GtkListStore.

list_store : A GtkListStore.
iter : A GtkTreeIter.
Returns : TRUE if the iter is valid, FALSE if the iter is invalid.

Since 2.2

---

gtk_list_store_reorder ()

```
void        gtk_list_store_reorder          (GtkListStore *store,
                                             gint *new_order);
```

Reorders *store* to follow the order indicated by *new_order*. Note that this function only works with unsorted stores.

store : A GtkListStore.

new_order : an array of integers mapping the new position of each child to its old position before the re-ordering, i.e. *new_order*[newpos] = oldpos.

Since 2.2

---

gtk_list_store_swap ()

```
void        gtk_list_store_swap             (GtkListStore *store,
                                             GtkTreeIter *a,
                                             GtkTreeIter *b);
```

Swaps *a* and *b* in *store*. Note that this function only works with unsorted stores.

store : A GtkListStore.

a : A GtkTreeIter.

b : Another GtkTreeIter.

Since 2.2

---

gtk_list_store_move_before ()

```
void        gtk_list_store_move_before      (GtkListStore *store,
                                             GtkTreeIter *iter,
                                             GtkTreeIter *position);
```

Moves *iter* in *store* to the position before *position*. Note that this function only works with unsorted stores. If *position* is NULL, *iter* will be moved to the end of the list.

store : A GtkListStore.

iter : A GtkTreeIter.

position : A GtkTreeIter, or NULL.

Since 2.2

---

gtk_list_store_move_after ()

```
void        gtk_list_store_move_after       (GtkListStore *store,
                                             GtkTreeIter *iter,
                                             GtkTreeIter *position);
```

Moves *iter* in *store* to the position after *position*. Note that this function only works with unsorted stores. If *position* is NULL, *iter* will be moved to the start of the list.

store : A GtkListStore.

iter : A GtkTreeIter.

position : A GtkTreeIter or NULL.

Since 2.2

See Also

GtkTreeModel, GtkTreeStore

<< GtkCellRendererToggle **GtkTreeStore >>**

GtkTreeStore

GtkTreeStore — A tree-like data structure that can be used with the GtkTreeView

Synopsis

```
#include <gtk/gtk.h>

            GtkTreeStore;
GtkTreeStore* gtk_tree_store_new           (gint n_columns,
                                            ...);
GtkTreeStore* gtk_tree_store_newv          (gint n_columns,
                                            GType *types);
void        gtk_tree_store_set_column_types (GtkTreeStore *tree_store,
                                            gint n_columns,
                                            GType *types);
void        gtk_tree_store_set_value       (GtkTreeStore *tree_store,
                                            GtkTreeIter *iter,
                                            gint column,
                                            GValue *value);
void        gtk_tree_store_set             (GtkTreeStore *tree_store,
                                            GtkTreeIter *iter,
                                            ...);
void        gtk_tree_store_set_valist      (GtkTreeStore *tree_store,
                                            GtkTreeIter *iter,
                                            va_list var_args);
gboolean    gtk_tree_store_remove          (GtkTreeStore *tree_store,
                                            GtkTreeIter *iter);
void        gtk_tree_store_insert          (GtkTreeStore *tree_store,
                                            GtkTreeIter *iter,
                                            GtkTreeIter *parent,
                                            gint position);
void        gtk_tree_store_insert_before   (GtkTreeStore *tree_store,
```

```
                                                GtkTreeIter *iter,
                                                GtkTreeIter *parent,
                                                GtkTreeIter *sibling);
void        gtk_tree_store_insert_after     (GtkTreeStore *tree_store,
                                                GtkTreeIter *iter,
                                                GtkTreeIter *parent,
                                                GtkTreeIter *sibling);
void        gtk_tree_store_prepend          (GtkTreeStore *tree_store,
                                                GtkTreeIter *iter,
                                                GtkTreeIter *parent);
void        gtk_tree_store_append           (GtkTreeStore *tree_store,
                                                GtkTreeIter *iter,
                                                GtkTreeIter *parent);
gboolean    gtk_tree_store_is_ancestor      (GtkTreeStore *tree_store,
                                                GtkTreeIter *iter,
                                                GtkTreeIter *descendant);
gint        gtk_tree_store_iter_depth       (GtkTreeStore *tree_store,
                                                GtkTreeIter *iter);
void        gtk_tree_store_clear            (GtkTreeStore *tree_store);
gboolean    gtk_tree_store_iter_is_valid    (GtkTreeStore *tree_store,
                                                GtkTreeIter *iter);
void        gtk_tree_store_reorder          (GtkTreeStore *tree_store,
                                                GtkTreeIter *parent,
                                                gint *new_order);
void        gtk_tree_store_swap             (GtkTreeStore *tree_store,
                                                GtkTreeIter *a,
                                                GtkTreeIter *b);
void        gtk_tree_store_move_before      (GtkTreeStore *tree_store,
                                                GtkTreeIter *iter,
                                                GtkTreeIter *position);
void        gtk_tree_store_move_after       (GtkTreeStore *tree_store,
                                                GtkTreeIter *iter,
                                                GtkTreeIter *position);
```

Object Hierarchy

```
GObject
 +----GtkTreeStore
```

Implemented Interfaces

GtkTreeStore implements GtkTreeModel, GtkTreeDragSource, GtkTreeDragDest and GtkTreeSortable.

Description

The GtkTreeStore object is a list model for use with a GtkTreeView widget. It implements the GtkTreeModel interface, and consequentialy, can use all of the methods available there. It also implements the GtkTreeSortable interface so it can be sorted by the view. Finally, it also implements the tree drag and drop interfaces.

Details

GtkTreeStore

```
typedef struct _GtkTreeStore GtkTreeStore;
```

---

gtk_tree_store_new ()

```
GtkTreeStore* gtk_tree_store_new           (gint n_columns,
                                            ...);
```

Creates a new tree store as with *n_columns* columns each of the types passed in. Note that only types derived from standard GObject fundamental types are supported.

As an example, `gtk_tree_store_new (3, G_TYPE_INT, G_TYPE_STRING, GDK_TYPE_PIXBUF);` will create a new GtkTreeStore with three columns, of type int, string and GdkPixbuf respectively.

n_columns : number of columns in the tree store
... : all GType types for the columns, from first to last
Returns : a new GtkTreeStore

---

gtk_tree_store_newv ()

```
GtkTreeStore* gtk_tree_store_newv           (gint n_columns,
                                             GType *types);
```

Non vararg creation function. Used primarily by language bindings.

n_columns : number of columns in the tree store
types : an array of GType types for the columns, from first to last
Returns : a new GtkTreeStore

---

gtk_tree_store_set_column_types ()

```
void        gtk_tree_store_set_column_types (GtkTreeStore *tree_store,
                                             gint n_columns,
                                             GType *types);
```

This function is meant primarily for GObjects that inherit from GtkTreeStore, and should only be used when constructing a new GtkTreeStore. It will not function after a row has been added, or a method on the GtkTreeModel interface is called.

tree_store : A GtkTreeStore
n_columns : Number of columns for the tree store
types : An array of GType types, one for each column

---

gtk_tree_store_set_value ()

```
void        gtk_tree_store_set_value        (GtkTreeStore *tree_store,
                                             GtkTreeIter *iter,
                                             gint column,
                                             GValue *value);
```

Sets the data in the cell specified by *iter* and *column*. The type of *value* must be convertible to the type of the column.

tree_store : a GtkTreeStore
iter : A valid GtkTreeIter for the row being modified
column : column number to modify
value : new value for the cell

---

gtk_tree_store_set ()

```
void        gtk_tree_store_set              (GtkTreeStore *tree_store,
                                             GtkTreeIter *iter,
                                             ...);
```

Sets the value of one or more cells in the row referenced by *iter*. The variable argument list should contain integer column numbers, each column number followed by the value to be set. The list is terminated by a -1. For example, to set column 0 with type `G_TYPE_STRING` to "Foo", you would write `gtk_tree_store_set (store, iter, 0, "Foo", -1)`.

tree_store : A GtkTreeStore
iter : A valid GtkTreeIter for the row being modified
... : pairs of column number and value, terminated with -1

---

gtk_tree_store_set_valist ()

```
void        gtk_tree_store_set_valist       (GtkTreeStore *tree_store,
                                             GtkTreeIter *iter,
                                             va_list var_args);
```

See gtk_tree_store_set(); this version takes a va_list for use by language bindings.

tree_store : A GtkTreeStore
iter : A valid GtkTreeIter for the row being modified
var_args : va_list of column/value pairs

---

gtk_tree_store_remove ()

```
gboolean    gtk_tree_store_remove           (GtkTreeStore *tree_store,
                                             GtkTreeIter *iter);
```

Removes *iter* from *tree_store*. After being removed, *iter* is set to the next valid row at that level, or invalidated if it previously pointed to the last one.

tree_store : A GtkTreeStore
iter : A valid GtkTreeIter
Returns : TRUE if *iter* is still valid, FALSE if not.

---

gtk_tree_store_insert ()

```
void        gtk_tree_store_insert           (GtkTreeStore *tree_store,
                                             GtkTreeIter *iter,
                                             GtkTreeIter *parent,
                                             gint position);
```

Creates a new row at *position*. If parent is non-NULL, then the row will be made a child of *parent*. Otherwise, the row will be created at the toplevel. If *position* is larger than the number of rows at that level, then the new row will be inserted to the end of the list. *iter* will be changed to point to this new row. The row will be empty after this function is called. To fill in values, you need to call gtk_tree_store_set() or gtk_tree_store_set_value().

tree_store : A GtkTreeStore

iter :	An unset GtkTreeIter to set to the new row
parent :	A valid GtkTreeIter, or NULL
position :	position to insert the new row

---

gtk_tree_store_insert_before ()

```
void        gtk_tree_store_insert_before    (GtkTreeStore *tree_store,
                                             GtkTreeIter *iter,
                                             GtkTreeIter *parent,
                                             GtkTreeIter *sibling);
```

Inserts a new row before *sibling*. If *sibling* is NULL, then the row will be appended to *parent* 's children. If *parent* and *sibling* are NULL, then the row will be appended to the toplevel. If both *sibling* and *parent* are set, then *parent* must be the parent of *sibling*. When *sibling* is set, *parent* is optional.

iter will be changed to point to this new row. The row will be empty after this function is called. To fill in values, you need to call gtk_tree_store_set() or gtk_tree_store_set_value().

tree_store :	A GtkTreeStore
iter :	An unset GtkTreeIter to set to the new row
parent :	A valid GtkTreeIter, or NULL
sibling :	A valid GtkTreeIter, or NULL

---

gtk_tree_store_insert_after ()

```
void        gtk_tree_store_insert_after     (GtkTreeStore *tree_store,
                                             GtkTreeIter *iter,
                                             GtkTreeIter *parent,
                                             GtkTreeIter *sibling);
```

Inserts a new row after *sibling*. If *sibling* is NULL, then the row will be prepended to *parent* 's children. If *parent* and *sibling* are NULL, then the row will be prepended to the toplevel. If both *sibling* and *parent* are set, then *parent* must be the parent of *sibling*. When *sibling* is set,

parent is optional.

iter will be changed to point to this new row. The row will be empty after this function is called. To fill in values, you need to call gtk_tree_store_set() or gtk_tree_store_set_value().

tree_store : A GtkTreeStore
iter : An unset GtkTreeIter to set to the new row
parent : A valid GtkTreeIter, or NULL
sibling : A valid GtkTreeIter, or NULL

---

gtk_tree_store_prepend ()

```
void        gtk_tree_store_prepend          (GtkTreeStore *tree_store,
                                             GtkTreeIter *iter,
                                             GtkTreeIter *parent);
```

Prepends a new row to *tree_store*. If *parent* is non-NULL, then it will prepend the new row before the first child of *parent*, otherwise it will prepend a row to the top level. *iter* will be changed to point to this new row. The row will be empty after this function is called. To fill in values, you need to call gtk_tree_store_set() or gtk_tree_store_set_value().

tree_store : A GtkTreeStore
iter : An unset GtkTreeIter to set to the prepended row
parent : A valid GtkTreeIter, or NULL

---

gtk_tree_store_append ()

```
void        gtk_tree_store_append           (GtkTreeStore *tree_store,
                                             GtkTreeIter *iter,
                                             GtkTreeIter *parent);
```

Appends a new row to *tree_store*. If *parent* is non-NULL, then it will append the new row after the last child of *parent*, otherwise it will append a row to the top level. *iter* will be changed to point to this

new row. The row will be empty after this function is called. To fill in values, you need to call `gtk_tree_store_set()` or `gtk_tree_store_set_value()`.

tree_store : A GtkTreeStore
iter : An unset GtkTreeIter to set to the appended row
parent : A valid GtkTreeIter, or NULL

---

gtk_tree_store_is_ancestor ()

```
gboolean    gtk_tree_store_is_ancestor      (GtkTreeStore *tree_store,
                                             GtkTreeIter *iter,
                                             GtkTreeIter *descendant);
```

Returns TRUE if *iter* is an ancestor of *descendant*. That is, *iter* is the parent (or grandparent or great-grandparent) of *descendant*.

tree_store : A GtkTreeStore
iter : A valid GtkTreeIter
descendant : A valid GtkTreeIter
Returns : TRUE, if *iter* is an ancestor of *descendant*

---

gtk_tree_store_iter_depth ()

```
gint        gtk_tree_store_iter_depth       (GtkTreeStore *tree_store,
                                             GtkTreeIter *iter);
```

Returns the depth of *iter*. This will be 0 for anything on the root level, 1 for anything down a level, etc.

tree_store : A GtkTreeStore
iter : A valid GtkTreeIter
Returns : The depth of *iter*

---

gtk_tree_store_clear ()

```
void        gtk_tree_store_clear            (GtkTreeStore *tree_store);
```

Removes all rows from *tree_store*

tree_store : a GtkTreeStore

---

gtk_tree_store_iter_is_valid ()

```
gboolean    gtk_tree_store_iter_is_valid    (GtkTreeStore *tree_store,
                                             GtkTreeIter *iter);
```

WARNING: This function is slow. Only use it for debugging and/or testing purposes.

Checks if the given iter is a valid iter for this GtkTreeStore.

tree_store : A GtkTreeStore.
iter : A GtkTreeIter.
Returns : TRUE if the iter is valid, FALSE if the iter is invalid.

Since 2.2

---

gtk_tree_store_reorder ()

```
void        gtk_tree_store_reorder          (GtkTreeStore *tree_store,
                                             GtkTreeIter *parent,
                                             gint *new_order);
```

Reorders the children of *parent* in *tree_store* to follow the order indicated by *new_order*. Note that this function only works with unsorted stores.

tree_store : A GtkTreeStore.

parent : A GtkTreeIter.

new_order : an array of integers mapping the new position of each child to its old position before the re-ordering, i.e. *new_order*[newpos] = oldpos.

Since 2.2

---

gtk_tree_store_swap ()

```
void        gtk_tree_store_swap             (GtkTreeStore *tree_store,
                                             GtkTreeIter *a,
                                             GtkTreeIter *b);
```

Swaps *a* and *b* in the same level of *tree_store*. Note that this function only works with unsorted stores.

tree_store : A GtkTreeStore.

a : A GtkTreeIter.

b : Another GtkTreeIter.

Since 2.2

---

gtk_tree_store_move_before ()

```
void        gtk_tree_store_move_before      (GtkTreeStore *tree_store,
                                             GtkTreeIter *iter,
                                             GtkTreeIter *position);
```

Moves *iter* in *tree_store* to the position before *position*. *iter* and *position* should be in the same level. Note that this function only works with unsorted stores. If *position* is NULL, *iter* will be moved to the end of the level.

tree_store : A GtkTreeStore.
iter : A GtkTreeIter.
position : A GtkTreeIter or NULL.

Since 2.2

---

gtk_tree_store_move_after ()

```
void        gtk_tree_store_move_after       (GtkTreeStore *tree_store,
                                             GtkTreeIter *iter,
                                             GtkTreeIter *position);
```

Moves *iter* in *tree_store* to the position after *position*. *iter* and *position* should be in the same level. Note that this function only works with unsorted stores. If *position* is NULL, *iter* will be moved to the start of the level.

tree_store : A GtkTreeStore.
iter : A GtkTreeIter.
position : A GtkTreeIter.

Since 2.2

See Also

GtkTreeModel, GtkTreeStore

<< GtkListStore **Menus, Combo Box, Toolbar >>**

Menus, Combo Box, Toolbar

GtkComboBox - A widget used to choose from a list of items
GtkComboBoxEntry - A text entry field with a dropdown list
GtkMenu - A menu widget
GtkMenuBar - A subclass widget for GtkMenuShell which holds GtkMenuItem widgets
GtkMenuItem - The widget used for item in menus
GtkMenuShell - A base class for menu objects
GtkImageMenuItem - A menu item with an icon
GtkRadioMenuItem - A choice from multiple check menu items
GtkCheckMenuItem - A menu item with a check box
GtkSeparatorMenuItem - A separator used in menus
GtkTearoffMenuItem - A menu item used to tear off and reattach its menu
GtkToolbar - Create bars of buttons and other widgets
GtkToolItem - The base class of widgets that can be added to GtkToolbar
GtkSeparatorToolItem - A toolbar item that separates groups of other toolbar items
GtkToolButton - A GtkToolItem subclass that displays buttons
GtkMenuToolButton - A GtkToolItem containing a button with an additional dropdown menu
GtkToggleToolButton - A GtkToolItem containing a toggle button
GtkRadioToolButton - A toolbar item that contains a radio button

GtkComboBox

GtkComboBox — A widget used to choose from a list of items

Synopsis

```
#include <gtk/gtk.h>

            GtkComboBox;
GtkWidget*  gtk_combo_box_new               (void);
GtkWidget*  gtk_combo_box_new_with_model    (GtkTreeModel *model);
gint        gtk_combo_box_get_wrap_width    (GtkComboBox *combo_box);
void        gtk_combo_box_set_wrap_width    (GtkComboBox *combo_box,
                                             gint width);
gint        gtk_combo_box_get_row_span_column
                                            (GtkComboBox *combo_box);
void        gtk_combo_box_set_row_span_column
                                            (GtkComboBox *combo_box,
                                             gint row_span);
gint        gtk_combo_box_get_column_span_column
                                            (GtkComboBox *combo_box);
void        gtk_combo_box_set_column_span_column
                                            (GtkComboBox *combo_box,
                                             gint column_span);
gint        gtk_combo_box_get_active        (GtkComboBox *combo_box);
void        gtk_combo_box_set_active        (GtkComboBox *combo_box,
                                             gint index_);
gboolean    gtk_combo_box_get_active_iter   (GtkComboBox *combo_box,
                                             GtkTreeIter *iter);
void        gtk_combo_box_set_active_iter   (GtkComboBox *combo_box,
                                             GtkTreeIter *iter);
GtkTreeModel* gtk_combo_box_get_model       (GtkComboBox *combo_box);
void        gtk_combo_box_set_model         (GtkComboBox *combo_box,
                                             GtkTreeModel *model);
GtkWidget*  gtk_combo_box_new_text          (void);
void        gtk_combo_box_append_text       (GtkComboBox *combo_box,
```

```
                                                const gchar *text);
void        gtk_combo_box_insert_text       (GtkComboBox *combo_box,
                                             gint position,
                                             const gchar *text);
void        gtk_combo_box_prepend_text      (GtkComboBox *combo_box,
                                             const gchar *text);
void        gtk_combo_box_remove_text       (GtkComboBox *combo_box,
                                             gint position);
gchar*      gtk_combo_box_get_active_text   (GtkComboBox *combo_box);
void        gtk_combo_box_popup             (GtkComboBox *combo_box);
void        gtk_combo_box_popdown           (GtkComboBox *combo_box);
AtkObject*  gtk_combo_box_get_popup_accessible
                                            (GtkComboBox *combo_box);
GtkTreeViewRowSeparatorFunc gtk_combo_box_get_row_separator_func
                                            (GtkComboBox *combo_box);
void        gtk_combo_box_set_row_separator_func
                                            (GtkComboBox *combo_box,
                                             GtkTreeViewRowSeparatorFunc func,
                                             gpointer data,
                                             GtkDestroyNotify destroy);
void        gtk_combo_box_set_add_tearoffs  (GtkComboBox *combo_box,
                                             gboolean add_tearoffs);
gboolean    gtk_combo_box_get_add_tearoffs  (GtkComboBox *combo_box);
void        gtk_combo_box_set_focus_on_click
                                            (GtkComboBox *combo,
                                             gboolean focus_on_click);
gboolean    gtk_combo_box_get_focus_on_click
                                            (GtkComboBox *combo);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkWidget
               +----GtkContainer
                     +----GtkBin
                           +----GtkComboBox
                                 +----GtkComboBoxEntry
```

Implemented Interfaces

GtkComboBox implements AtkImplementorIface, GtkCellLayout and GtkCellEditable.

Properties

```
  "active"               gint                 : Read / Write
  "add-tearoffs"         gboolean             : Read / Write
  "column-span-column"   gint                 : Read / Write
  "focus-on-click"       gboolean             : Read / Write
  "has-frame"            gboolean             : Read / Write
  "model"                GtkTreeModel         : Read / Write
  "row-span-column"      gint                 : Read / Write
  "wrap-width"           gint                 : Read / Write
```

Style Properties

```
  "appears-as-list"      gboolean             : Read
```

Signal Prototypes

```
"changed"   void        user_function      (GtkComboBox *combobox,
                                            gpointer user_data);
```

Description

Details

GtkComboBox

```
typedef struct _GtkComboBox GtkComboBox;
```

---

gtk_combo_box_new ()

```
GtkWidget*  gtk_combo_box_new               (void);
```

Creates a new empty GtkComboBox.

Returns : A new GtkComboBox.

Since 2.4

---

gtk_combo_box_new_with_model ()

```
GtkWidget*  gtk_combo_box_new_with_model    (GtkTreeModel *model);
```

Creates a new GtkComboBox with the model initialized to *model*.

model : A GtkTreeModel.

Returns : A new GtkComboBox.

Since 2.4

---

gtk_combo_box_get_wrap_width ()

```
gint        gtk_combo_box_get_wrap_width    (GtkComboBox *combo_box);
```

Returns the wrap width which is used to determine the number of columns for the popup menu. If the wrap width is larger than 1, the combo box is in table mode.

combo_box : A GtkComboBox.

Returns : the wrap width.

Since 2.6

gtk_combo_box_set_wrap_width ()

```
void        gtk_combo_box_set_wrap_width    (GtkComboBox *combo_box,
                                             gint width);
```

Sets the wrap width of *combo_box* to be *width*. The wrap width is basically the preferred number of columns when you want to the popup to be layed out in a table.

combo_box : A GtkComboBox.
width : Preferred number of columns.

Since 2.4

gtk_combo_box_get_row_span_column ()

```
gint        gtk_combo_box_get_row_span_column
                                            (GtkComboBox *combo_box);
```

Returns the column with row span information for *combo_box*.

combo_box : A GtkComboBox.
Returns : the row span column.

Since 2.6

gtk_combo_box_set_row_span_column ()

```
void        gtk_combo_box_set_row_span_column
                                            (GtkComboBox *combo_box,
                                             gint row_span);
```

Sets the column with row span information for *combo_box* to be *row_span*. The row span column contains integers which indicate how many rows an item should span.

combo_box : A GtkComboBox.

row_span : A column in the model passed during construction.

Since 2.4

---

gtk_combo_box_get_column_span_column ()

```
gint        gtk_combo_box_get_column_span_column
                                            (GtkComboBox *combo_box);
```

Returns the column with column span information for *combo_box*.

combo_box : A GtkComboBox.

Returns : the column span column.

Since 2.6

---

gtk_combo_box_set_column_span_column ()

```
void        gtk_combo_box_set_column_span_column
                                            (GtkComboBox *combo_box,
                                             gint column_span);
```

Sets the column with column span information for *combo_box* to be *column_span*. The column span column contains integers which indicate how many columns an item should span.

combo_box : A GtkComboBox.

column_span : A column in the model passed during construction.

Since 2.4

---

gtk_combo_box_get_active ()

```
gint        gtk_combo_box_get_active        (GtkComboBox *combo_box);
```

Returns the index of the currently active item, or -1 if there's no active item.

combo_box : A GtkComboBox.
Returns : An integer which is the index of the currently active item, or -1 if there's no active item.

Since 2.4

---

gtk_combo_box_set_active ()

```
void        gtk_combo_box_set_active        (GtkComboBox *combo_box,
                                             gint index_);
```

Sets the active item of *combo_box* to be the item at *index*.

combo_box : A GtkComboBox.
*index_* : An index in the model passed during construction, or -1 to have no active item.

Since 2.4

---

gtk_combo_box_get_active_iter ()

```
gboolean    gtk_combo_box_get_active_iter   (GtkComboBox *combo_box,
                                             GtkTreeIter *iter);
```

Sets *iter* to point to the current active item, if it exists.

combo_box : A GtkComboBox
iter : The uninitialized GtkTreeIter.
Returns : TRUE, if *iter* was set

Since 2.4

---

gtk_combo_box_set_active_iter ()

```
void        gtk_combo_box_set_active_iter   (GtkComboBox *combo_box,
                                             GtkTreeIter *iter);
```

Sets the current active item to be the one referenced by *iter*. *iter* must correspond to a path of depth one.

combo_box : A GtkComboBox
iter : The GtkTreeIter.

Since 2.4

---

gtk_combo_box_get_model ()

```
GtkTreeModel* gtk_combo_box_get_model       (GtkComboBox *combo_box);
```

Returns the GtkTreeModel which is acting as data source for *combo_box*.

combo_box : A GtkComboBox.
Returns : A GtkTreeModel which was passed during construction.

Since 2.4

---

gtk_combo_box_set_model ()

```
void        gtk_combo_box_set_model         (GtkComboBox *combo_box,
                                             GtkTreeModel *model);
```

Sets the model used by *combo_box* to be *model*. Will unset a previously set model (if applicable). If model is NULL, then it will unset the model.

Note that this function does not clear the cell renderers, you have to call `gtk_combo_box_cell_layout_clear()` yourself if you need to set up different cell renderers for the new model.

combo_box : A GtkComboBox.
model : A GtkTreeModel.

Since 2.4

---

gtk_combo_box_new_text ()

```
GtkWidget*  gtk_combo_box_new_text          (void);
```

Convenience function which constructs a new text combo box, which is a GtkComboBox just displaying strings. If you use this function to create a text combo box, you should only manipulate its data source with the following convenience functions: `gtk_combo_box_append_text()`, `gtk_combo_box_insert_text()`, `gtk_combo_box_prepend_text()` and `gtk_combo_box_remove_text()`.

Returns : A new text combo box.

Since 2.4

---

gtk_combo_box_append_text ()

```
void        gtk_combo_box_append_text       (GtkComboBox *combo_box,
                                             const gchar *text);
```

Appends *string* to the list of strings stored in *combo_box*. Note that you can only use this function with combo boxes constructed with `gtk_combo_box_new_text()`.

combo_box : A GtkComboBox constructed using `gtk_combo_box_new_text()`.
text : A string.

Since 2.4

---

gtk_combo_box_insert_text ()

```
void        gtk_combo_box_insert_text       (GtkComboBox *combo_box,
                                             gint position,
                                             const gchar *text);
```

Inserts *string* at *position* in the list of strings stored in *combo_box*. Note that you can only use this function with combo boxes constructed with gtk_combo_box_new_text().

combo_box : A GtkComboBox constructed using gtk_combo_box_new_text().
position : An index to insert *text*.
text : A string.

Since 2.4

---

gtk_combo_box_prepend_text ()

```
void        gtk_combo_box_prepend_text      (GtkComboBox *combo_box,
                                             const gchar *text);
```

Prepends *string* to the list of strings stored in *combo_box*. Note that you can only use this function with combo boxes constructed with gtk_combo_box_new_text().

combo_box : A GtkComboBox constructed with gtk_combo_box_new_text().
text : A string.

Since 2.4

---

gtk_combo_box_remove_text ()

```
void        gtk_combo_box_remove_text       (GtkComboBox *combo_box,
                                             gint position);
```

Removes the string at *position* from *combo_box*. Note that you can only use this function with combo boxes constructed with gtk_combo_box_new_text().

combo_box : A GtkComboBox constructed with gtk_combo_box_new_text().
position : Index of the item to remove.

Since 2.4

---

gtk_combo_box_get_active_text ()

```
gchar*      gtk_combo_box_get_active_text   (GtkComboBox *combo_box);
```

Returns the currently active string in *combo_box* or NULL if none is selected. Note that you can only use this function with combo boxes constructed with gtk_combo_box_new_text().

combo_box : A GtkComboBox constructed with gtk_combo_box_new_text().
Returns : a newly allocated string containing the currently active text.

Since 2.6

---

gtk_combo_box_popup ()

```
void        gtk_combo_box_popup             (GtkComboBox *combo_box);
```

Pops up the menu or dropdown list of *combo_box*.

This function is mostly intended for use by accessibility technologies; applications should have little use for it.

combo_box : a GtkComboBox

Since 2.4

---

gtk_combo_box_popdown ()

```
void        gtk_combo_box_popdown           (GtkComboBox *combo_box);
```

Hides the menu or dropdown list of *combo_box*.

This function is mostly intended for use by accessibility technologies; applications should have little use for it.

combo_box : a GtkComboBox

Since 2.4

---

gtk_combo_box_get_popup_accessible ()

```
AtkObject*  gtk_combo_box_get_popup_accessible
                                            (GtkComboBox *combo_box);
```

Gets the accessible object corresponding to the combo box's popup.

This function is mostly intended for use by accessibility technologies; applications should have little use for it.

combo_box : a GtkComboBox

Returns : the accessible object corresponding to the combo box's popup.

Since 2.6

---

gtk_combo_box_get_row_separator_func ()

```
GtkTreeViewRowSeparatorFunc gtk_combo_box_get_row_separator_func
                                            (GtkComboBox *combo_box);
```

Returns the current row separator function.

combo_box : a GtkComboBox

Returns : the current row separator function.

Since 2.6

---

gtk_combo_box_set_row_separator_func ()

```
void        gtk_combo_box_set_row_separator_func
                                            (GtkComboBox *combo_box,
                                             GtkTreeViewRowSeparatorFunc func,
                                             gpointer data,
                                             GtkDestroyNotify destroy);
```

Sets the row separator function, which is used to determine whether a row should be drawn as a separator. If the row separator function is NULL, no separators are drawn. This is the default value.

combo_box : a GtkComboBox
func : a GtkTreeViewRowSeparatorFunc
data : user data to pass to *func*, or NULL
destroy : destroy notifier for *data*, or NULL

Since 2.6

---

gtk_combo_box_set_add_tearoffs ()

```
void        gtk_combo_box_set_add_tearoffs  (GtkComboBox *combo_box,
                                             gboolean add_tearoffs);
```

Sets whether the popup menu should have a tearoff menu item.

combo_box : a GtkComboBox
add_tearoffs : TRUE to add tearoff menu items

Since 2.6

---

gtk_combo_box_get_add_tearoffs ()

```
gboolean    gtk_combo_box_get_add_tearoffs  (GtkComboBox *combo_box);
```

Gets the current value of the :add-tearoffs property.

combo_box : a GtkComboBox

Returns : the current value of the :add-tearoffs property.

---

gtk_combo_box_set_focus_on_click ()

```
void        gtk_combo_box_set_focus_on_click
                                            (GtkComboBox *combo,
                                             gboolean focus_on_click);
```

Sets whether the combo box will grab focus when it is clicked with the mouse. Making mouse clicks not grab focus is useful in places like toolbars where you don't want the keyboard focus removed from the main area of the application.

combo : a GtkComboBox

focus_on_click : whether the combo box grabs focus when clicked with the mouse

Since 2.6

---

gtk_combo_box_get_focus_on_click ()

```
gboolean    gtk_combo_box_get_focus_on_click
                                            (GtkComboBox *combo);
```

Returns whether the combo box grabs focus when it is clicked with the mouse. See gtk_combo_box_set_focus_on_click().

combo : a GtkComboBox

Returns : TRUE if the combo box grabs focus when it is clicked with the mouse.

Since 2.6

Properties

The "active" property

```
"active"              gint                  : Read / Write
```

The item which is currently active.

Allowed values: >= -1

Default value: -1

The "add-tearoffs" property

```
"add-tearoffs"        gboolean              : Read / Write
```

The "add-tearoffs" property controls whether generated menus have tearoff menu items.

Note that this only affects menu style combo boxes.

Default value: FALSE

Since 2.6

The "column-span-column" property

```
"column-span-column"  gint                  : Read / Write
```

TreeModel column containing the column span values.

Allowed values: >= -1

Default value: -1

The "focus-on-click" property

"focus-on-click"	gboolean	: Read / Write

Whether the combo box grabs focus when it is clicked with the mouse.

Default value: TRUE

The "has-frame" property

"has-frame"	gboolean	: Read / Write

The :has-frame property controls whether a frame is drawn around the entry.

Default value: TRUE

Since 2.6

The "model" property

"model"	GtkTreeModel	: Read / Write

The model for the combo box.

The "row-span-column" property

"row-span-column"	gint	: Read / Write

TreeModel column containing the row span values.

Allowed values: >= -1

Default value: -1

The "wrap-width" property

```
  "wrap-width"          gint                : Read / Write
```

Wrap width for layouting the items in a grid.

Allowed values: >= 0

Default value: 0

Style Properties

The "appears-as-list" style property

```
  "appears-as-list"     gboolean            : Read
```

Whether dropdowns should look like lists rather than menus.

Default value: FALSE

Signals

The "changed" signal

```
void        user_function                   (GtkComboBox *combobox,
                                             gpointer user_data);
```

combobox : the object which received the signal.
user_data : user data set when the signal handler was connected.

<< Menus, Combo Box, Toolbar **GtkComboBoxEntry >>**

GtkComboBoxEntry

GtkComboBoxEntry — A text entry field with a dropdown list

Synopsis

```
#include <gtk/gtk.h>


            GtkComboBoxEntry;
GtkWidget*  gtk_combo_box_entry_new         (void);
GtkWidget*  gtk_combo_box_entry_new_with_model
                                            (GtkTreeModel *model,
                                             gint text_column);
GtkWidget*  gtk_combo_box_entry_new_text    (void);
void        gtk_combo_box_entry_set_text_column
                                            (GtkComboBoxEntry *entry_box,
                                             gint text_column);
gint        gtk_combo_box_entry_get_text_column
                                            (GtkComboBoxEntry *entry_box);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkWidget
               +----GtkContainer
                     +----GtkBin
                           +----GtkComboBox
                                 +----GtkComboBoxEntry
```

Implemented Interfaces

GtkComboBoxEntry implements AtkImplementorIface, GtkCellLayout and GtkCellEditable.

Properties

```
  "text-column"          gint                  : Read / Write
```

Description

Details

GtkComboBoxEntry

```
typedef struct _GtkComboBoxEntry GtkComboBoxEntry;
```

---

gtk_combo_box_entry_new ()

```
GtkWidget*  gtk_combo_box_entry_new         (void);
```

Creates a new GtkComboBoxEntry which has a GtkEntry as child. After construction, you should set a model using `gtk_combo_box_set_model()` and a text_column * using `gtk_combo_box_entry_set_text_column()`.

Returns : A new GtkComboBoxEntry.

Since 2.4

---

gtk_combo_box_entry_new_with_model ()

```
GtkWidget*  gtk_combo_box_entry_new_with_model
                                              (GtkTreeModel *model,
                                               gint text_column);
```

Creates a new GtkComboBoxEntry which has a GtkEntry as child and a list of strings as popup. You can get the GtkEntry from a GtkComboBoxEntry using GTK_ENTRY (GTK_BIN (combo_box_entry)->child). To add and remove strings from the list, just modify *model* using its data manipulation API.

model : A GtkTreeModel.
text_column : A column in *model* to get the strings from.
Returns : A new GtkComboBoxEntry.

Since 2.4

---

gtk_combo_box_entry_new_text ()

```
GtkWidget*  gtk_combo_box_entry_new_text    (void);
```

Convenience function which constructs a new editable text combo box, which is a GtkComboBoxEntry just displaying strings. If you use this function to create a text combo box, you should only manipulate its data source with the following convenience functions: `gtk_combo_box_append_text()`, `gtk_combo_box_insert_text()`, `gtk_combo_box_prepend_text()` and `gtk_combo_box_remove_text()`.

Returns : A new text GtkComboBoxEntry.

Since 2.4

---

gtk_combo_box_entry_set_text_column ()

```
void        gtk_combo_box_entry_set_text_column
                                              (GtkComboBoxEntry *entry_box,
                                               gint text_column);
```

Sets the model column which *entry_box* should use to get strings from to be *text_column*.

entry_box : A GtkComboBoxEntry.
text_column : A column in *model* to get the strings from.

Since 2.4.

---

gtk_combo_box_entry_get_text_column ()

```
gint        gtk_combo_box_entry_get_text_column
                                            (GtkComboBoxEntry *entry_box);
```

Returns the column which *entry_box* is using to get the strings from.

entry_box : A GtkComboBoxEntry.
Returns : A column in the data source model of *entry_box*.

Since 2.4

Properties

The "text-column" property

```
  "text-column"          gint                  : Read / Write
```

A column in the data source model to get the strings from.

Allowed values: >= -1

Default value: -1

GtkMenu

GtkMenu — A menu widget

Synopsis

```
#include <gtk/gtk.h>


            GtkMenu;
GtkWidget*  gtk_menu_new                    (void);
void        gtk_menu_set_screen             (GtkMenu *menu,
                                             GdkScreen *screen);
#define     gtk_menu_append                 (menu,child)
#define     gtk_menu_prepend                (menu,child)
#define     gtk_menu_insert                 (menu,child,pos)
void        gtk_menu_reorder_child          (GtkMenu *menu,
                                             GtkWidget *child,
                                             gint position);
void        gtk_menu_attach                 (GtkMenu *menu,
                                             GtkWidget *child,
                                             guint left_attach,
                                             guint right_attach,
                                             guint top_attach,
                                             guint bottom_attach);
void        gtk_menu_popup                  (GtkMenu *menu,
                                             GtkWidget *parent_menu_shell,
                                             GtkWidget *parent_menu_item,
                                             GtkMenuPositionFunc func,
                                             gpointer data,
                                             guint button,
                                             guint32 activate_time);
void        gtk_menu_set_accel_group        (GtkMenu *menu,
                                             GtkAccelGroup *accel_group);
GtkAccelGroup* gtk_menu_get_accel_group     (GtkMenu *menu);
void        gtk_menu_set_accel_path         (GtkMenu *menu,
                                             const gchar *accel_path);
```

```
void        gtk_menu_set_title              (GtkMenu *menu,
                                             const gchar *title);
gboolean    gtk_menu_get_tearoff_state      (GtkMenu *menu);
G_CONST_RETURN gchar* gtk_menu_get_title    (GtkMenu *menu);

void        gtk_menu_popdown                (GtkMenu *menu);
void        gtk_menu_reposition             (GtkMenu *menu);
GtkWidget*  gtk_menu_get_active             (GtkMenu *menu);
void        gtk_menu_set_active             (GtkMenu *menu,
                                             guint index_);
void        gtk_menu_set_tearoff_state      (GtkMenu *menu,
                                             gboolean torn_off);
void        gtk_menu_attach_to_widget       (GtkMenu *menu,
                                             GtkWidget *attach_widget,
                                             GtkMenuDetachFunc detacher);
void        gtk_menu_detach                 (GtkMenu *menu);
GtkWidget*  gtk_menu_get_attach_widget      (GtkMenu *menu);
GList*      gtk_menu_get_for_attach_widget  (GtkWidget *widget);
void        (*GtkMenuPositionFunc)          (GtkMenu *menu,
                                             gint *x,
                                             gint *y,
                                             gboolean *push_in,
                                             gpointer user_data);
void        (*GtkMenuDetachFunc)            (GtkWidget *attach_widget,
                                             GtkMenu *menu);
void        gtk_menu_set_monitor            (GtkMenu *menu,
                                             gint monitor_num);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkWidget
               +----GtkContainer
                     +----GtkMenuShell
                           +----GtkMenu
```

Implemented Interfaces

GtkMenu implements AtkImplementorIface.

Properties

```
  "tearoff-state"        gboolean              : Read / Write
  "tearoff-title"        gchararray            : Read / Write
```

Child Properties

```
  "bottom-attach"        gint                  : Read / Write
  "left-attach"          gint                  : Read / Write
  "right-attach"         gint                  : Read / Write
  "top-attach"           gint                  : Read / Write
```

Style Properties

```
  "horizontal-offset"    gint                  : Read
  "vertical-offset"      gint                  : Read
  "vertical-padding"     gint                  : Read
```

Signal Prototypes

```
"move-scroll"
            void        user_function      (GtkMenu *menu,
                                            GtkScrollType arg1,
                                            gpointer user_data);
```

Description

A GtkMenu is a GtkMenuShell that implements a drop down menu consisting of a list of GtkMenuItem objects which can be navigated and activated by the user to perform application functions.

A GtkMenu is most commonly dropped down by activating a GtkMenuItem in a GtkMenuBar or popped up by activating a GtkMenuItem in another GtkMenu.

A GtkMenu can also be popped up by activating a GtkOptionMenu. Other composite widgets such as the GtkNotebook can pop up a GtkMenu as well.

Applications can display a GtkMenu as a popup menu by calling the `gtk_menu_popup()` function. The example below shows how an application can pop up a menu when the 3rd mouse button is pressed.

Example 1. Connecting the popup signal handler.

```
    /* connect our handler which will popup the menu */
    g_signal_connect_swapped (window, "button_press_event",
        G_CALLBACK (my_popup_handler), menu);
```

Example 2. Signal handler which displays a popup menu.

```
static gint
my_popup_handler (GtkWidget *widget, GdkEvent *event)
{
  GtkMenu *menu;
  GdkEventButton *event_button;

  g_return_val_if_fail (widget != NULL, FALSE);
  g_return_val_if_fail (GTK_IS_MENU (widget), FALSE);
  g_return_val_if_fail (event != NULL, FALSE);

  /* The "widget" is the menu that was supplied when
   * g_signal_connect_swapped() was called.
   */
  menu = GTK_MENU (widget);

  if (event->type == GDK_BUTTON_PRESS)
    {
      event_button = (GdkEventButton *) event;
      if (event_button->button == 3)
        {
          gtk_menu_popup (menu, NULL, NULL, NULL, NULL,
                          event_button->button, event_button->time);
          return TRUE;
        }
    }

  return FALSE;
}
```

Details

GtkMenu

```
typedef struct _GtkMenu GtkMenu;
```

The GtkMenu struct contains private data only, and should be accessed using the functions below.

---

gtk_menu_new ()

```
GtkWidget*  gtk_menu_new                    (void);
```

Creates a new GtkMenu.

Returns : a new GtkMenu.

---

gtk_menu_set_screen ()

```
void        gtk_menu_set_screen             (GtkMenu *menu,
                                             GdkScreen *screen);
```

Sets the GdkScreen on which the menu will be displayed.

menu : a GtkMenu.

screen : a GdkScreen, or NULL if the screen should be determined by the widget the menu is attached to.

Since 2.2

---

gtk_menu_append()

```
#define gtk_menu_append(menu,child)     gtk_menu_shell_append  ((GtkMenuShell
*)(menu),(child))
```

Warning

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

Adds a new GtkMenuItem to the end of the menu's item list.

menu : a GtkMenu.
child : The GtkMenuItem to add.

---

gtk_menu_prepend()

```
#define gtk_menu_prepend(menu,child)    gtk_menu_shell_prepend ((GtkMenuShell
*)(menu),(child))
```

Warning

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

Adds a new GtkMenuItem to the beginning of the menu's item list.

menu : a GtkMenu.
child : The GtkMenuItem to add.

---

gtk_menu_insert()

```
#define gtk_menu_insert(menu,child,pos) gtk_menu_shell_insert ((GtkMenuShell
*)(menu),(child),(pos))
```

Warning

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

Adds a new GtkMenuItem to the menu's item list at the position indicated by *position*.

menu : a GtkMenu.

child : The GtkMenuItem to add.

pos :

---

gtk_menu_reorder_child ()

```
void        gtk_menu_reorder_child          (GtkMenu *menu,
                                             GtkWidget *child,
                                             gint position);
```

Moves a GtkMenuItem to a new position within the GtkMenu.

menu : a GtkMenu.

child : the GtkMenuItem to move.

position : the new position to place *child*. Positions are numbered from 0 to n-1.

---

gtk_menu_attach ()

```
void        gtk_menu_attach                 (GtkMenu *menu,
                                             GtkWidget *child,
                                             guint left_attach,
                                             guint right_attach,
                                             guint top_attach,
                                             guint bottom_attach);
```

Adds a new GtkMenuItem to a (table) menu. The number of 'cells' that an item will occupy is specified by *left_attach*, *right_attach*, *top_attach* and *bottom_attach*. These each represent the leftmost, rightmost, uppermost and lower column and row numbers of the table. (Columns and rows are indexed from zero).

Note that this function is not related to gtk_menu_detach().

menu : a GtkMenu.

child : a GtkMenuItem.

left_attach : The column number to attach the left side of the item to.

right_attach : The column number to attach the right side of the item to.
top_attach : The row number to attach the top of the item to.
bottom_attach : The row number to attach the bottom of the item to.

Since 2.4

---

gtk_menu_popup ()

```
void        gtk_menu_popup                  (GtkMenu *menu,
                                             GtkWidget *parent_menu_shell,
                                             GtkWidget *parent_menu_item,
                                             GtkMenuPositionFunc func,
                                             gpointer data,
                                             guint button,
                                             guint32 activate_time);
```

Displays a menu and makes it available for selection. Applications can use this function to display context-sensitive menus, and will typically supply NULL for the *parent_menu_shell*, *parent_menu_item*, *func* and *data* parameters. The default menu positioning function will position the menu at the current mouse cursor position.

The *button* parameter should be the mouse button pressed to initiate the menu popup. If the menu popup was initiated by something other than a mouse button press, such as a mouse button release or a keypress, *button* should be 0.

The *activate_time* parameter should be the time stamp of the event that initiated the popup. If such an event is not available, use gtk_get_current_event_time() instead.

menu : a GtkMenu.
parent_menu_shell : the menu shell containing the triggering menu item, or NULL
parent_menu_item : the menu item whose activation triggered the popup, or NULL
func : a user supplied function used to position the menu, or NULL
data : user supplied data to be passed to *func*.
button : the mouse button which was pressed to initiate the event.
activate_time : the time at which the activation event occurred.

---

gtk_menu_set_accel_group ()

```
void        gtk_menu_set_accel_group        (GtkMenu *menu,
                                             GtkAccelGroup *accel_group);
```

Set the GtkAccelGroup which holds global accelerators for the menu. This accelerator group needs to also be added to all windows that this menu is being used in with `gtk_window_add_accel_group()`, in order for those windows to support all the accelerators contained in this group.

menu : a GtkMenu.

accel_group : the GtkAccelGroup to be associated with the menu.

---

gtk_menu_get_accel_group ()

```
GtkAccelGroup* gtk_menu_get_accel_group     (GtkMenu *menu);
```

Gets the GtkAccelGroup which holds global accelerators for the menu. See `gtk_menu_set_accel_group()`.

menu : a GtkMenu.

Returns : the GtkAccelGroup associated with the menu.

---

gtk_menu_set_accel_path ()

```
void        gtk_menu_set_accel_path         (GtkMenu *menu,
                                             const gchar *accel_path);
```

Sets an accelerator path for this menu from which accelerator paths for its immediate children, its menu items, can be constructed. The main purpose of this function is to spare the programmer the inconvenience of having to call `gtk_menu_item_set_accel_path()` on each menu item that should support runtime user changable accelerators. Instead, by just calling `gtk_menu_set_accel_path()` on their parent, each menu item of this menu, that contains a label describing its purpose, automatically gets an accel path assigned. For example, a menu containing menu items "New" and "Exit", will, after `gtk_menu_set_accel_path (menu, "<Gnumeric-Sheet>/File");` has been called, assign its items the accel paths: `"<Gnumeric-Sheet>/File/New"` and `"<Gnumeric-Sheet>/File/Exit"`. Assigning accel paths to menu items then enables the user to change their accelerators at runtime. More details about accelerator paths and their default setups can be found at `gtk_accel_map_add_entry()`.

menu : a valid GtkMenu

accel_path : a valid accelerator path

---

gtk_menu_set_title ()

```
void        gtk_menu_set_title              (GtkMenu *menu,
                                             const gchar *title);
```

Sets the title string for the menu. The title is displayed when the menu is shown as a tearoff menu.

menu : a GtkMenu

title : a string containing the title for the menu.

---

gtk_menu_get_tearoff_state ()

```
gboolean    gtk_menu_get_tearoff_state      (GtkMenu *menu);
```

Returns whether the menu is torn off. See gtk_menu_set_tearoff_state().

menu : a GtkMenu

Returns : TRUE if the menu is currently torn off.

---

gtk_menu_get_title ()

```
G_CONST_RETURN gchar* gtk_menu_get_title    (GtkMenu *menu);
```

Returns the title of the menu. See gtk_menu_set_title().

menu : a GtkMenu

Returns : the title of the menu, or NULL if the menu has no title set on it. This string is owned by the widget and should not be modified or freed.

---

gtk_menu_popdown ()

```
void        gtk_menu_popdown                (GtkMenu *menu);
```

Removes the menu from the screen.

menu : a GtkMenu.

---

gtk_menu_reposition ()

```
void        gtk_menu_reposition             (GtkMenu *menu);
```

Repositions the menu according to its position function.

menu : a GtkMenu.

---

gtk_menu_get_active ()

```
GtkWidget*  gtk_menu_get_active             (GtkMenu *menu);
```

Returns the selected menu item from the menu. This is used by the GtkOptionMenu.

menu : a GtkMenu.

Returns : the GtkMenuItem that was last selected in the menu. If a selection has not yet been made, the first menu item is selected.

---

gtk_menu_set_active ()

```
void        gtk_menu_set_active             (GtkMenu *menu,
                                             guint index_);
```

Selects the specified menu item within the menu. This is used by the GtkOptionMenu and should not be used by anyone else.

menu : a GtkMenu.

*index_* : the index of the menu item to select. Index values are from 0 to n-1.

---

gtk_menu_set_tearoff_state ()

```
void        gtk_menu_set_tearoff_state      (GtkMenu *menu,
                                             gboolean torn_off);
```

Changes the tearoff state of the menu. A menu is normally displayed as drop down menu which persists as long as the menu is active. It can also be displayed as a tearoff menu which persists until it is closed or reattached.

menu : a GtkMenu.

torn_off : If TRUE, menu is displayed as a tearoff menu.

---

gtk_menu_attach_to_widget ()

```
void        gtk_menu_attach_to_widget       (GtkMenu *menu,
                                             GtkWidget *attach_widget,
                                             GtkMenuDetachFunc detacher);
```

Attaches the menu to the widget and provides a callback function that will be invoked when the menu calls `gtk_menu_detach()` during its destruction.

menu : a GtkMenu.

attach_widget : the GtkWidget that the menu will be attached to.

detacher : the user supplied callback function that will be called when the menu calls `gtk_menu_detach()`.

---

gtk_menu_detach ()

```
void        gtk_menu_detach                 (GtkMenu *menu);
```

Detaches the menu from the widget to which it had been attached. This function will call the callback function, *detacher*, provided when the `gtk_menu_attach_to_widget()` function was called.

menu : a GtkMenu.

---

gtk_menu_get_attach_widget ()

```
GtkWidget*  gtk_menu_get_attach_widget      (GtkMenu *menu);
```

Returns the GtkWidget that the menu is attached to.

menu : a GtkMenu.

Returns : the GtkWidget that the menu is attached to.

---

gtk_menu_get_for_attach_widget ()

```
GList*      gtk_menu_get_for_attach_widget  (GtkWidget *widget);
```

Returns a list of the menus which are attached to this widget. This list is owned by GTK+ and must not be modified.

widget : a GtkWidget

Returns : the list of menus attached to his widget.

Since 2.6

GtkMenuPositionFunc ()

```
void        (*GtkMenuPositionFunc)          (GtkMenu *menu,
                                             gint *x,
                                             gint *y,
                                             gboolean *push_in,
                                             gpointer user_data);
```

A user function supplied when calling `gtk_menu_popup()` which controls the positioning of the menu when it is displayed. The function sets the *x* and *y* parameters to the coordinates where the menu is to be drawn.

menu : a GtkMenu.

x : address of the gint representing the horizontal position where the menu shall be drawn. This is an output parameter.

y : address of the gint representing the vertical position where the menu shall be drawn. This is an output parameter.

push_in : whether the menu should be pushed in to be completely inside the screen instead of just clamped to the size to the screen.

user_data : the data supplied by the user in the `gtk_menu_popup()` *data* parameter.

GtkMenuDetachFunc ()

```
void        (*GtkMenuDetachFunc)            (GtkWidget *attach_widget,
                                             GtkMenu *menu);
```

A user function supplied when calling `gtk_menu_attach_to_widget()` which will be called when the menu is later detached from the widget.

attach_widget : the GtkWidget that the menu is being detached from.

menu : the GtkMenu being detached.

---

gtk_menu_set_monitor ()

```
void        gtk_menu_set_monitor            (GtkMenu *menu,
                                             gint monitor_num);
```

Informs GTK+ on which monitor a menu should be popped up. See `gdk_screen_get_monitor_geometry()`.

This function should be called from a GtkMenuPositionFunc if the menu should not appear on the same monitor as the pointer. This information can't be reliably inferred from the coordinates returned by a GtkMenuPositionFunc, since, for very long menus, these coordinates may extend beyond the monitor boundaries or even the screen boundaries.

menu : a GtkMenu

monitor_num : the number of the monitor on which the menu should be popped up

Since 2.4

Properties

The "tearoff-state" property

"tearoff-state"	gboolean	: Read / Write

A boolean that indicates whether the menu is torn-off.

Default value: FALSE

Since 2.6

The "tearoff-title" property

"tearoff-title"	gchararray	: Read / Write

A title that may be displayed by the window manager when this menu is torn-off.

Default value: ""

Child Properties

The "bottom-attach" child property

"bottom-attach"	gint	: Read / Write

The row number to attach the bottom of the child to.

Allowed values: >= -1

Default value: -1

The "left-attach" child property

```
"left-attach"          gint                : Read / Write
```

The column number to attach the left side of the child to.

Allowed values: >= -1

Default value: -1

The "right-attach" child property

```
"right-attach"         gint                : Read / Write
```

The column number to attach the right side of the child to.

Allowed values: >= -1

Default value: -1

The "top-attach" child property

```
"top-attach"           gint                : Read / Write
```

The row number to attach the top of the child to.

Allowed values: >= -1

Default value: -1

Style Properties

The "horizontal-offset" style property

```
```

```
  "horizontal-offset"    gint                  : Read
```

When the menu is a submenu, position it this number of pixels offset horizontally.

Default value: -2

The "vertical-offset" style property

```
  "vertical-offset"      gint                  : Read
```

When the menu is a submenu, position it this number of pixels offset vertically.

Default value: 0

The "vertical-padding" style property

```
  "vertical-padding"     gint                  : Read
```

Extra space at the top and bottom of the menu.

Allowed values: >= 0

Default value: 1

Signals

The "move-scroll" signal

```
void        user_function                  (GtkMenu *menu,
                                            GtkScrollType arg1,
                                            gpointer user_data);
```

menu : the object which received the signal.

arg1 :

user_data : user data set when the signal handler was connected.

<< GtkComboBoxEntry **GtkMenuBar >>**

GtkMenuBar

GtkMenuBar — A subclass widget for GtkMenuShell which holds GtkMenuItem widgets

Synopsis

```
#include <gtk/gtk.h>


            GtkMenuBar;
GtkWidget*  gtk_menu_bar_new                (void);
#define     gtk_menu_bar_append             (menu,child)
#define     gtk_menu_bar_prepend            (menu,child)
#define     gtk_menu_bar_insert             (menu,child,pos)
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkWidget
               +----GtkContainer
                     +----GtkMenuShell
                           +----GtkMenuBar
```

Implemented Interfaces

GtkMenuBar implements AtkImplementorIface.

Style Properties

```
  "internal-padding"     gint                 : Read
  "shadow-type"          GtkShadowType        : Read
```

Description

The GtkMenuBar is a subclass of GtkMenuShell which contains one to many GtkMenuItem. The result is a standard menu bar which can hold many menu items. GtkMenuBar allows for a shadow type to be set for aesthetic purposes. The shadow types are defined in the gtk_menu_bar_set_shadow_type function.

Details

GtkMenuBar

```
typedef struct _GtkMenuBar GtkMenuBar;
```

The GtkMenuBar struct contains the following fields. (These fields should be considered read-only. They should never be set by an application.)

---

gtk_menu_bar_new ()

```
GtkWidget*  gtk_menu_bar_new                (void);
```

Creates the new GtkMenuBar

Returns : the GtkMenuBar

---

gtk_menu_bar_append()

```
#define gtk_menu_bar_append(menu,child)     gtk_menu_shell_append  ((GtkMenuShell
*)(menu),(child))
```

Warning

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

Adds a new GtkMenuItem to the end of the GtkMenuBar

menu :

child : the GtkMenuItem to add

---

gtk_menu_bar_prepend()

```
#define gtk_menu_bar_prepend(menu,child)    gtk_menu_shell_prepend ((GtkMenuShell
*)(menu),(child))
```

Warning

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

Adds a new GtkMenuItem to the beginning of the GtkMenuBar

menu :
child : the GtkMenuItem to add

---

gtk_menu_bar_insert()

```
#define gtk_menu_bar_insert(menu,child,pos) gtk_menu_shell_insert ((GtkMenuShell
*)(menu),(child),(pos))
```

Warning

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

Adds a new GtkMenuItem to the GtkMenuBar at the position defined by *position*

menu :
child : the GtkMenuItem to add
pos :

Style Properties

The "internal-padding" style property

```
  "internal-padding"     gint                 : Read
```

Amount of border space between the menubar shadow and the menu items.

Allowed values: >= 0

Default value: 1

The "shadow-type" style property

```
  "shadow-type"          GtkShadowType         : Read
```

Style of bevel around the menubar.

Default value: GTK_SHADOW_OUT

See Also

GtkMenuShell, GtkMenu, GtkMenuItem

<< GtkMenu **GtkMenuItem >>**

GtkMenuItem

GtkMenuItem — The widget used for item in menus

Synopsis

```
#include <gtk/gtk.h>

            GtkMenuItem;
GtkWidget*  gtk_menu_item_new               (void);
GtkWidget*  gtk_menu_item_new_with_label    (const gchar *label);
GtkWidget*  gtk_menu_item_new_with_mnemonic (const gchar *label);
void        gtk_menu_item_set_right_justified
                                            (GtkMenuItem *menu_item,
                                             gboolean right_justified);
void        gtk_menu_item_set_submenu       (GtkMenuItem *menu_item,
                                             GtkWidget *submenu);
void        gtk_menu_item_set_accel_path    (GtkMenuItem *menu_item,
                                             const gchar *accel_path);
void        gtk_menu_item_remove_submenu    (GtkMenuItem *menu_item);
void        gtk_menu_item_select            (GtkMenuItem *menu_item);
void        gtk_menu_item_deselect          (GtkMenuItem *menu_item);
void        gtk_menu_item_activate          (GtkMenuItem *menu_item);
void        gtk_menu_item_toggle_size_request
                                            (GtkMenuItem *menu_item,
                                             gint *requisition);
void        gtk_menu_item_toggle_size_allocate
                                            (GtkMenuItem *menu_item,
                                             gint allocation);
#define     gtk_menu_item_right_justify     (menu_item)
gboolean    gtk_menu_item_get_right_justified
                                            (GtkMenuItem *menu_item);
GtkWidget*  gtk_menu_item_get_submenu       (GtkMenuItem *menu_item);
```

Object Hierarchy

```
GObject
 +----GtkObject
       +----GtkWidget
             +----GtkContainer
                   +----GtkBin
                         +----GtkItem
                               +----GtkMenuItem
                                     +----GtkCheckMenuItem
                                     +----GtkImageMenuItem
                                     +----GtkSeparatorMenuItem
                                     +----GtkTearoffMenuItem
```

Implemented Interfaces

GtkMenuItem implements AtkImplementorIface.

Style Properties

```
"arrow-spacing"         gint                : Read
"horizontal-padding"    gint                : Read
"selected-shadow-type" GtkShadowType        : Read
"toggle-spacing"        gint                : Read
```

Signal Prototypes

```
"activate"  void        user_function      (GtkMenuItem *menuitem,
                                            gpointer user_data);
"activate-item"
            void        user_function      (GtkMenuItem *menuitem,
                                            gpointer user_data);
"toggle-size-allocate"
            void        user_function      (GtkMenuItem *menuitem,
                                            gint arg1,
                                            gpointer user_data);
"toggle-size-request"
            void        user_function      (GtkMenuItem *menuitem,
                                            gpointer arg1,
```

```
                                                gpointer user_data);
```

Description

The GtkMenuItem widget and the derived widgets are the only valid childs for menus. Their function is to correctly handle highlighting, alignment, events and submenus.

As it derives from GtkBin it can hold any valid child widget, altough only a few are really useful.

Details

GtkMenuItem

```
typedef struct _GtkMenuItem GtkMenuItem;
```

---

gtk_menu_item_new ()

```
GtkWidget*  gtk_menu_item_new               (void);
```

Creates a new GtkMenuItem.

Returns : the newly created GtkMenuItem

---

gtk_menu_item_new_with_label ()

```
GtkWidget*  gtk_menu_item_new_with_label    (const gchar *label);
```

Creates a new GtkMenuItem whose child is a GtkLabel.

label : the text for the label
Returns : the newly created GtkMenuItem

---

gtk_menu_item_new_with_mnemonic ()

```
GtkWidget*  gtk_menu_item_new_with_mnemonic (const gchar *label);
```

Creates a new GtkMenuItem containing a label. The label will be created using `gtk_label_new_with_mnemonic()`, so underscores in *`label`* indicate the mnemonic for the menu item.

`label` : The text of the button, with an underscore in front of the mnemonic character
Returns : a new GtkMenuItem

---

gtk_menu_item_set_right_justified ()

```
void        gtk_menu_item_set_right_justified
                                            (GtkMenuItem *menu_item,
                                             gboolean right_justified);
```

Sets whether the menu item appears justified at the right side of a menu bar. This was traditionally done for "Help" menu items, but is now considered a bad idea. (If the widget layout is reversed for a right-to-left language like Hebrew or Arabic, right-justified-menu-items appear at the left.)

`menu_item` : a GtkMenuItem.
`right_justified` : if TRUE the menu item will appear at the far right if added to a menu bar.

---

gtk_menu_item_set_submenu ()

```
void        gtk_menu_item_set_submenu       (GtkMenuItem *menu_item,
                                             GtkWidget *submenu);
```

Sets the widget submenu, or changes it.

`menu_item` : the menu item widget
`submenu` : the submenu

---

gtk_menu_item_set_accel_path ()

```
void        gtk_menu_item_set_accel_path    (GtkMenuItem *menu_item,
                                             const gchar *accel_path);
```

Set the accelerator path on *menu_item*, through which runtime changes of the menu item's accelerator caused by the user can be identified and saved to persistant storage (see gtk_accel_map_save() on this). To setup a default accelerator for this menu item, call gtk_accel_map_add_entry() with the same *accel_path*. See also gtk_accel_map_add_entry() on the specifics of accelerator paths, and gtk_menu_set_accel_path() for a more convenient variant of this function.

This function is basically a convenience wrapper that handles calling gtk_widget_set_accel_path() with the appropriate accelerator group for the menu item.

Note that you do need to set an accelerator on the parent menu with gtk_menu_set_accel_group() for this to work.

menu_item : a valid GtkMenuItem

accel_path : accelerator path, corresponding to this menu item's functionality, or NULL to unset the current path.

---

gtk_menu_item_remove_submenu ()

```
void        gtk_menu_item_remove_submenu    (GtkMenuItem *menu_item);
```

Removes the widget's submenu.

menu_item : the menu item widget

---

gtk_menu_item_select ()

```
void        gtk_menu_item_select            (GtkMenuItem *menu_item);
```

Emits the "select" signal on the given item. Behaves exactly like gtk_item_select.

menu_item : the menu item

---

gtk_menu_item_deselect ()

```
void        gtk_menu_item_deselect          (GtkMenuItem *menu_item);
```

Emits the "deselect" signal on the given item. Behaves exactly like gtk_item_deselect.

menu_item : the menu item

---

gtk_menu_item_activate ()

```
void        gtk_menu_item_activate          (GtkMenuItem *menu_item);
```

Emits the "activate" signal on the given item

menu_item : the menu item

---

gtk_menu_item_toggle_size_request ()

```
void        gtk_menu_item_toggle_size_request
                                            (GtkMenuItem *menu_item,
                                             gint *requisition);
```

Emits the "toggle_size_request" signal on the given item.

menu_item : the menu item
requisition : the requisition to use as signal data.

---

gtk_menu_item_toggle_size_allocate ()

```
void        gtk_menu_item_toggle_size_allocate
                                            (GtkMenuItem *menu_item,
                                             gint allocation);
```

Emits the "toggle_size_allocate" signal on the given item.

menu_item : the menu item.
allocation : the allocation to use as signal data.

---

gtk_menu_item_right_justify()

```
#define gtk_menu_item_right_justify(menu_item) gtk_menu_item_set_right_justified
((menu_item), TRUE)
```

Warning

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

Sets the menu item to be right-justified. Only useful for menu bars.

menu_item : the menu item

---

gtk_menu_item_get_right_justified ()

```
gboolean    gtk_menu_item_get_right_justified
                                            (GtkMenuItem *menu_item);
```

Gets whether the menu item appears justified at the right side of the menu bar.

menu_item : a GtkMenuItem

Returns : TRUE if the menu item will appear at the far right if added to a menu bar.

---

gtk_menu_item_get_submenu ()

```
GtkWidget*  gtk_menu_item_get_submenu       (GtkMenuItem *menu_item);
```

Gets the submenu underneath this menu item, if any. See `gtk_menu_item_set_submenu()`.

menu_item : a GtkMenuItem

Returns : submenu for this menu item, or NULL if none.

Style Properties

The "arrow-spacing" style property

```
"arrow-spacing"        gint                  : Read
```

Space between label and arrow.

Allowed values: >= 0

Default value: 10

The "horizontal-padding" style property

```
"horizontal-padding"   gint                  : Read
```

Padding to left and right of the menu item.

Allowed values: >= 0

Default value: 3

The "selected-shadow-type" style property

```
"selected-shadow-type" GtkShadowType         : Read
```

Shadow type when item is selected.

Default value: GTK_SHADOW_NONE

The "toggle-spacing" style property

```
"toggle-spacing"       gint                  : Read
```

Space between icon and label.

Allowed values: >= 0

Default value: 5

Signals

The "activate" signal

```
void        user_function                   (GtkMenuItem *menuitem,
                                             gpointer user_data);
```

Emitted when the item is activated.

menuitem : the object which received the signal.
user_data : user data set when the signal handler was connected.

The "activate-item" signal

```
void        user_function                   (GtkMenuItem *menuitem,
                                             gpointer user_data);
```

Emitted when the item is activated, but also if the menu item has a submenu. For normal applications, the relevant signal is "activate".

menuitem : the object which received the signal.
user_data : user data set when the signal handler was connected.

The "toggle-size-allocate" signal

```
void        user_function                   (GtkMenuItem *menuitem,
                                             gint arg1,
                                             gpointer user_data);
```

menuitem : the object which received the signal.
arg1 :
user_data : user data set when the signal handler was connected.

The "toggle-size-request" signal

```
void        user_function                   (GtkMenuItem *menuitem,
                                             gpointer arg1,
                                             gpointer user_data);
```

menuitem : the object which received the signal.

arg1 :

user_data : user data set when the signal handler was connected.

See Also

GtkBin	for how to handle the child.
GtkItem	is the abstract class for all sorts of items.
GtkMenuShell	is always the parent of GtkMenuItem.

<< GtkMenuBar **GtkMenuShell >>**

GtkMenuShell

GtkMenuShell — A base class for menu objects

Synopsis

```
#include <gtk/gtk.h>

            GtkMenuShell;
void        gtk_menu_shell_append           (GtkMenuShell *menu_shell,
                                             GtkWidget *child);
void        gtk_menu_shell_prepend          (GtkMenuShell *menu_shell,
                                             GtkWidget *child);
void        gtk_menu_shell_insert           (GtkMenuShell *menu_shell,
                                             GtkWidget *child,
                                             gint position);
void        gtk_menu_shell_deactivate       (GtkMenuShell *menu_shell);
void        gtk_menu_shell_select_item      (GtkMenuShell *menu_shell,
                                             GtkWidget *menu_item);
void        gtk_menu_shell_select_first     (GtkMenuShell *menu_shell,
                                             gboolean search_sensitive);
void        gtk_menu_shell_deselect         (GtkMenuShell *menu_shell);
void        gtk_menu_shell_activate_item    (GtkMenuShell *menu_shell,
                                             GtkWidget *menu_item,
                                             gboolean force_deactivate);
void        gtk_menu_shell_cancel           (GtkMenuShell *menu_shell);
enum        GtkMenuDirectionType;
```

Object Hierarchy

```
  GObject
```

```
   +----GtkObject
         +----GtkWidget
               +----GtkContainer
                     +----GtkMenuShell
                           +----GtkMenuBar
                           +----GtkMenu
```

Implemented Interfaces

GtkMenuShell implements AtkImplementorIface.

Signal Prototypes

```
"activate-current"
            void        user_function      (GtkMenuShell *menushell,
                                            gboolean force_hide,
                                            gpointer user_data);
"cancel"    void        user_function      (GtkMenuShell *menushell,
                                            gpointer user_data);
"cycle-focus"
            void        user_function      (GtkMenuShell *menushell,
                                            GtkDirectionType arg1,
                                            gpointer user_data);
"deactivate"
            void        user_function      (GtkMenuShell *menushell,
                                            gpointer user_data);
"move-current"
            void        user_function      (GtkMenuShell *menushell,
                                            GtkMenuDirectionType direction,
                                            gpointer user_data);
"selection-done"
            void        user_function      (GtkMenuShell *menushell,
                                            gpointer user_data);
```

Description

A GtkMenuShell is the abstract base class used to derive the GtkMenu and GtkMenuBar subclasses.

A GtkMenuShell is a container of GtkMenuItem objects arranged in a list which can be navigated, selected, and activated by the user to perform application functions. A GtkMenuItem can have a submenu associated with it, allowing for nested hierarchical menus.

Details

GtkMenuShell

```
typedef struct _GtkMenuShell GtkMenuShell;
```

The GtkMenuShell-struct struct contains the following fields. (These fields should be considered read-only. They should never be set by an application.)

GList *children; The list of GtkMenuItem objects contained by this GtkMenuShell.

---

gtk_menu_shell_append ()

```
void        gtk_menu_shell_append           (GtkMenuShell *menu_shell,
                                             GtkWidget *child);
```

Adds a new GtkMenuItem to the end of the menu shell's item list.

menu_shell : a GtkMenuShell.

child : The GtkMenuItem to add.

---

gtk_menu_shell_prepend ()

```
void        gtk_menu_shell_prepend          (GtkMenuShell *menu_shell,
                                             GtkWidget *child);
```

Adds a new GtkMenuItem to the beginning of the menu shell's item list.

menu_shell : a GtkMenuShell.

child : The GtkMenuItem to add.

---

gtk_menu_shell_insert ()

```
void        gtk_menu_shell_insert           (GtkMenuShell *menu_shell,
                                             GtkWidget *child,
                                             gint position);
```

Adds a new GtkMenuItem to the menu shell's item list at the position indicated by *position*.

menu_shell : a GtkMenuShell.

child : The GtkMenuItem to add.

position : The position in the item list where *child* is added. Positions are numbered from 0 to n-1.

---

gtk_menu_shell_deactivate ()

```
void        gtk_menu_shell_deactivate       (GtkMenuShell *menu_shell);
```

Deactivates the menu shell. Typically this results in the menu shell being erased from the screen.

menu_shell : a GtkMenuShell.

---

gtk_menu_shell_select_item ()

```
void        gtk_menu_shell_select_item      (GtkMenuShell *menu_shell,
                                             GtkWidget *menu_item);
```

Selects the menu item from the menu shell.

menu_shell : a GtkMenuShell.

menu_item : The GtkMenuItem to select.

gtk_menu_shell_select_first ()

```
void        gtk_menu_shell_select_first     (GtkMenuShell *menu_shell,
                                             gboolean search_sensitive);
```

Select the first visible or selectable child of the menu shell; don't select tearoff items unless the only item is a tearoff item.

menu_shell : a GtkMenuShell

search_sensitive : if TRUE, search for the first selectable menu item, otherwise select nothing if the first item isn't sensitive. This should be FALSE if the menu is being popped up initially.

Since 2.2

gtk_menu_shell_deselect ()

```
void        gtk_menu_shell_deselect         (GtkMenuShell *menu_shell);
```

Deselects the currently selected item from the menu shell, if any.

menu_shell : a GtkMenuShell.

gtk_menu_shell_activate_item ()

```
void        gtk_menu_shell_activate_item    (GtkMenuShell *menu_shell,
                                             GtkWidget *menu_item,
                                             gboolean force_deactivate);
```

Activates the menu item within the menu shell.

menu_shell : a GtkMenuShell.

menu_item : The GtkMenuItem to activate.

force_deactivate : If TRUE, force the deactivation of the menu shell after the menu item is activated.

---

gtk_menu_shell_cancel ()

```
void        gtk_menu_shell_cancel           (GtkMenuShell *menu_shell);
```

Cancels the selection within the menu shell.

menu_shell : a GtkMenuShell

Since 2.4

---

enum GtkMenuDirectionType

```
typedef enum
{
  GTK_MENU_DIR_PARENT,
  GTK_MENU_DIR_CHILD,
  GTK_MENU_DIR_NEXT,
  GTK_MENU_DIR_PREV
} GtkMenuDirectionType;
```

An enumeration representing directional movements within a menu.

GTK_MENU_DIR_PARENT	To the parent menu shell.
GTK_MENU_DIR_CHILD	To the submenu, if any, associated with the item.
GTK_MENU_DIR_NEXT	To the next menu item.
GTK_MENU_DIR_PREV	To the previous menu item.

Signals

The "activate-current" signal

```
void        user_function                   (GtkMenuShell *menushell,
                                             gboolean force_hide,
                                             gpointer user_data);
```

An action signal that activates the current menu item within the menu shell.

menushell : the object which received the signal.
force_hide : if TRUE, hide the menu after activating the menu item.
user_data : user data set when the signal handler was connected.

The "cancel" signal

```
void        user_function                   (GtkMenuShell *menushell,
                                             gpointer user_data);
```

An action signal which cancels the selection within the menu shell. Causes the GtkMenuShell::selection-done signal to be emitted.

menushell : the object which received the signal.
user_data : user data set when the signal handler was connected.

The "cycle-focus" signal

```
void        user_function                   (GtkMenuShell *menushell,
                                             GtkDirectionType arg1,
                                             gpointer user_data);
```

menushell : the object which received the signal.
arg1 :
user_data : user data set when the signal handler was connected.

The "deactivate" signal

```
void        user_function                   (GtkMenuShell *menushell,
                                             gpointer user_data);
```

This signal is emitted when a menu shell is deactivated.

menushell : the object which received the signal.
user_data : user data set when the signal handler was connected.

The "move-current" signal

```
void        user_function                   (GtkMenuShell *menushell,
                                             GtkMenuDirectionType direction,
                                             gpointer user_data);
```

An action signal which moves the current menu item in the direction specified by *direction*.

menushell : the object which received the signal.
direction : the direction to move.
user_data : user data set when the signal handler was connected.

The "selection-done" signal

```
void        user_function                   (GtkMenuShell *menushell,
                                             gpointer user_data);
```

This signal is emitted when a selection has been completed within a menu shell.

menushell : the object which received the signal.
user_data : user data set when the signal handler was connected.

<< GtkMenuItem **GtkImageMenuItem >>**

GtkImageMenuItem

GtkImageMenuItem — A menu item with an icon

Synopsis

```
#include <gtk/gtk.h>

            GtkImageMenuItem;
void        gtk_image_menu_item_set_image   (GtkImageMenuItem *image_menu_item,
                                             GtkWidget *image);
GtkWidget*  gtk_image_menu_item_get_image   (GtkImageMenuItem *image_menu_item);
GtkWidget*  gtk_image_menu_item_new         (void);
GtkWidget*  gtk_image_menu_item_new_from_stock
                                            (const gchar *stock_id,
                                             GtkAccelGroup *accel_group);
GtkWidget*  gtk_image_menu_item_new_with_label
                                            (const gchar *label);
GtkWidget*  gtk_image_menu_item_new_with_mnemonic
                                            (const gchar *label);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkWidget
               +----GtkContainer
                     +----GtkBin
                           +----GtkItem
                                 +----GtkMenuItem
                                       +----GtkImageMenuItem
```

Implemented Interfaces

GtkImageMenuItem implements AtkImplementorIface.

Properties

```
  "image"                GtkWidget             : Read / Write
```

Description

A GtkImageMenuItem is a menu item which has an icon next to the text label.

Note that the user can disable display of menu icons, so make sure to still fill in the text label.

Details

GtkImageMenuItem

```
typedef struct _GtkImageMenuItem GtkImageMenuItem;
```

---

gtk_image_menu_item_set_image ()

```
void        gtk_image_menu_item_set_image   (GtkImageMenuItem *image_menu_item,
                                             GtkWidget *image);
```

Sets the image of *image_menu_item* to the given widget.

image_menu_item : a GtkImageMenuItem.
image : a widget to set as the image for the menu item.

---

gtk_image_menu_item_get_image ()

```
GtkWidget*  gtk_image_menu_item_get_image   (GtkImageMenuItem *image_menu_item);
```

Gets the widget that is currently set as the image of *image_menu_item*. See gtk_image_menu_item_set_image().

image_menu_item : a GtkImageMenuItem.

Returns : the widget set as image of *image_menu_item*.

---

gtk_image_menu_item_new ()

```
GtkWidget*  gtk_image_menu_item_new         (void);
```

Creates a new GtkImageMenuItem with an empty label.

Returns : a new GtkImageMenuItem.

---

gtk_image_menu_item_new_from_stock ()

```
GtkWidget*  gtk_image_menu_item_new_from_stock
                                            (const gchar *stock_id,
                                             GtkAccelGroup *accel_group);
```

Creates a new GtkImageMenuItem containing the image and text from a stock item. Some stock ids have preprocessor macros like GTK_STOCK_OK and GTK_STOCK_APPLY.

If you want this menu item to have changeable accelerators, then pass in NULL for accel_group. Next call gtk_menu_item_set_accel_path() with an appropriate path for the menu item, use gtk_stock_lookup() to look up the standard accelerator for the stock item, and if one is found, call gtk_accel_map_add_entry() to register it.

stock_id : the name of the stock item.

accel_group : the GtkAccelGroup to add the menu items accelerator to, or NULL.

Returns : a new GtkImageMenuItem.

---

gtk_image_menu_item_new_with_label ()

```
GtkWidget*  gtk_image_menu_item_new_with_label
```

```
                                                (const gchar *label);
```

Creates a new GtkImageMenuItem containing a label.

label : the text of the menu item.
Returns : a new GtkImageMenuItem.

---

gtk_image_menu_item_new_with_mnemonic ()

```
GtkWidget*  gtk_image_menu_item_new_with_mnemonic
                                                (const gchar *label);
```

Creates a new GtkImageMenuItem containing a label. The label will be created using gtk_label_new_with_mnemonic(), so underscores in *label* indicate the mnemonic for the menu item.

label : the text of the menu item, with an underscore in front of the mnemonic character
Returns : a new GtkImageMenuItem

Properties

The "image" property

```
  "image"                GtkWidget             : Read / Write
```

Child widget to appear next to the menu text.

<< GtkMenuShell **GtkRadioMenuItem >>**

GtkRadioMenuItem

GtkRadioMenuItem — A choice from multiple check menu items

Synopsis

```
#include <gtk/gtk.h>


            GtkRadioMenuItem;
GtkWidget*  gtk_radio_menu_item_new         (GSList *group);
GtkWidget*  gtk_radio_menu_item_new_with_label
                                            (GSList *group,
                                             const gchar *label);
GtkWidget*  gtk_radio_menu_item_new_with_mnemonic
                                            (GSList *group,
                                             const gchar *label);
GtkWidget*  gtk_radio_menu_item_new_from_widget
                                            (GtkRadioMenuItem *group);
GtkWidget*  gtk_radio_menu_item_new_with_label_from_widget
                                            (GtkRadioMenuItem *group,
                                             const gchar *label);
GtkWidget*  gtk_radio_menu_item_new_with_mnemonic_from_widget
                                            (GtkRadioMenuItem *group,
                                             const gchar *label);
#define     gtk_radio_menu_item_group
void        gtk_radio_menu_item_set_group   (GtkRadioMenuItem *radio_menu_item,
                                             GSList *group);
GSList*     gtk_radio_menu_item_get_group   (GtkRadioMenuItem *radio_menu_item);
```

Object Hierarchy

```
  GObject
   +----GtkObject
```

```
          +----GtkWidget
                +----GtkContainer
                      +----GtkBin
                            +----GtkItem
                                  +----GtkMenuItem
                                        +----GtkCheckMenuItem
                                              +----GtkRadioMenuItem
```

Implemented Interfaces

GtkRadioMenuItem implements AtkImplementorIface.

Signal Prototypes

```
"group-changed"
            void        user_function      (GtkRadioMenuItem *radiomenuitem,
                                            gpointer user_data);
```

Description

A radio menu item is a check menu item that belongs to a group. At each instant exactly one of the radio menu items from a group is selected.

The group list does not need to be freed, as each GtkRadioMenuItem will remove itself and its list item when it is destroyed.

The correct way to create a group of radio menu items is approximatively this:

Example 3. How to create a group of radio menu items.

```
GSList *group = NULL;
GtkWidget *item;
gint i;

for (i = 0; i < 5; i++)
{
  item = gtk_radio_menu_item_new_with_label (group, "This is an example");
  group = gtk_radio_menu_item_get_group (GTK_RADIO_MENU_ITEM (item));
  if (i == 1)
    gtk_check_menu_item_set_active (GTK_CHECK_MENU_ITEM (item), TRUE);
}
```

Details

GtkRadioMenuItem

```
typedef struct _GtkRadioMenuItem GtkRadioMenuItem;
```

The structure contains only private data that must be accessed through the interface functions.

---

gtk_radio_menu_item_new ()

```
GtkWidget*  gtk_radio_menu_item_new         (GSList *group);
```

Creates a new GtkRadioMenuItem.

group : the group to which the radio menu item is to be attached
Returns : a new GtkRadioMenuItem

---

gtk_radio_menu_item_new_with_label ()

```
GtkWidget*  gtk_radio_menu_item_new_with_label
                                            (GSList *group,
                                             const gchar *label);
```

Creates a new GtkRadioMenuItem whose child is a simple GtkLabel.

group : the group to which the radio menu item is to be attached
label : the text for the label
Returns : a new GtkRadioMenuItem

---

gtk_radio_menu_item_new_with_mnemonic ()

```
GtkWidget*  gtk_radio_menu_item_new_with_mnemonic
                                            (GSList *group,
```

```
                                                const gchar *label);
```

Creates a new GtkRadioMenuItem containing a label. The label will be created using `gtk_label_new_with_mnemonic()`, so underscores in *`label`* indicate the mnemonic for the menu item.

`group` : group the radio menu item is inside
`label` : the text of the button, with an underscore in front of the mnemonic character
Returns : a new GtkRadioMenuItem

---

gtk_radio_menu_item_new_from_widget ()

```
GtkWidget*  gtk_radio_menu_item_new_from_widget
                                               (GtkRadioMenuItem *group);
```

Creates a new GtkRadioMenuItem adding it to the same group as *`group`*.

`group` : An existing GtkRadioMenuItem
Returns : The new GtkRadioMenuItem

Since 2.4

---

gtk_radio_menu_item_new_with_label_from_widget ()

```
GtkWidget*  gtk_radio_menu_item_new_with_label_from_widget
                                               (GtkRadioMenuItem *group,
                                                const gchar *label);
```

Creates a new GtkRadioMenuItem whose child is a simple GtkLabel. The new GtkRadioMenuItem is added to the same group as *`group`*.

`group` : an existing GtkRadioMenuItem
`label` : the text for the label
Returns : The new GtkRadioMenuItem

Since 2.4

gtk_radio_menu_item_new_with_mnemonic_from_widget ()

```
GtkWidget*  gtk_radio_menu_item_new_with_mnemonic_from_widget
                                                (GtkRadioMenuItem *group,
                                                 const gchar *label);
```

Creates a new GtkRadioMenuItem containing a label. The label will be created using `gtk_label_new_with_mnemonic()`, so underscores in label indicate the mnemonic for the menu item.

The new GtkRadioMenuItem is added to the same group as *`group`*.

`group` : An existing GtkRadioMenuItem

`label` : the text of the button, with an underscore in front of the mnemonic character

Returns : The new GtkRadioMenuItem

Since 2.4

---

gtk_radio_menu_item_group

```
#define gtk_radio_menu_item_group gtk_radio_menu_item_get_group
```

Warning

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

Deprecated compatibility macro. Use `gtk_radio_menu_item_get_group()` instead.

---

gtk_radio_menu_item_set_group ()

```
void        gtk_radio_menu_item_set_group   (GtkRadioMenuItem *radio_menu_item,
                                             GSList *group);
```

Sets the group of a radio menu item, or changes it.

radio_menu_item : a GtkRadioMenuItem.
group : the new group.

---

gtk_radio_menu_item_get_group ()

```
GSList*     gtk_radio_menu_item_get_group   (GtkRadioMenuItem *radio_menu_item);
```

Returns the group to which the radio menu item belongs, as a GList of GtkRadioMenuItem. The list belongs to GTK+ and should not be freed.

radio_menu_item : a GtkRadioMenuItem.
Returns : the group of *radio_menu_item*.

Signals

The "group-changed" signal

```
void        user_function                   (GtkRadioMenuItem *radiomenuitem,
                                             gpointer user_data);
```

radiomenuitem : the object which received the signal.
user_data : user data set when the signal handler was connected.

See Also

GtkMenuItem because a radio menu item is a menu item.
GtkCheckMenuItem to know how to handle the check.

<< GtkImageMenuItem **GtkCheckMenuItem >>**

GtkCheckMenuItem

GtkCheckMenuItem — A menu item with a check box

Synopsis

```
#include <gtk/gtk.h>

            GtkCheckMenuItem;
GtkWidget*  gtk_check_menu_item_new         (void);
GtkWidget*  gtk_check_menu_item_new_with_label
                                            (const gchar *label);
GtkWidget*  gtk_check_menu_item_new_with_mnemonic
                                            (const gchar *label);
#define     gtk_check_menu_item_set_state
gboolean    gtk_check_menu_item_get_active  (GtkCheckMenuItem *check_menu_item);
void        gtk_check_menu_item_set_active  (GtkCheckMenuItem *check_menu_item,
                                             gboolean is_active);
void        gtk_check_menu_item_set_show_toggle
                                            (GtkCheckMenuItem *menu_item,
                                             gboolean always);
void        gtk_check_menu_item_toggled     (GtkCheckMenuItem *check_menu_item);
gboolean    gtk_check_menu_item_get_inconsistent
                                            (GtkCheckMenuItem *check_menu_item);
void        gtk_check_menu_item_set_inconsistent
                                            (GtkCheckMenuItem *check_menu_item,
                                             gboolean setting);
void        gtk_check_menu_item_set_draw_as_radio
                                            (GtkCheckMenuItem *check_menu_item,
                                             gboolean draw_as_radio);
gboolean    gtk_check_menu_item_get_draw_as_radio
                                            (GtkCheckMenuItem *check_menu_item);
```

Object Hierarchy

```
GObject
 +----GtkObject
       +----GtkWidget
             +----GtkContainer
                   +----GtkBin
                         +----GtkItem
                               +----GtkMenuItem
                                     +----GtkCheckMenuItem
                                           +----GtkRadioMenuItem
```

Implemented Interfaces

GtkCheckMenuItem implements AtkImplementorIface.

Properties

```
"active"              gboolean            : Read / Write
"draw-as-radio"       gboolean            : Read / Write
"inconsistent"        gboolean            : Read / Write
```

Style Properties

```
"indicator-size"      gint                : Read
```

Signal Prototypes

```
"toggled"   void        user_function      (GtkCheckMenuItem *checkmenuitem,
                                            gpointer user_data);
```

Description

A GtkCheckMenuItem is a menu item that maintains the state of a boolean value in addition to a GtkMenuItem's usual role in activating application code.

A check box indicating the state of the boolean value is displayed at the left side of the GtkMenuItem. Activating the GtkMenuItem toggles the value.

Details

GtkCheckMenuItem

```
typedef struct _GtkCheckMenuItem GtkCheckMenuItem;
```

The GtkCheckMenuItem-struct struct contains the following fields. (These fields should be considered read-only. They should never be set by an application.)

guint active;	TRUE if the check box is active.

---

gtk_check_menu_item_new ()

```
GtkWidget*  gtk_check_menu_item_new         (void);
```

Creates a new GtkCheckMenuItem.

Returns : a new GtkCheckMenuItem.

---

gtk_check_menu_item_new_with_label ()

```
GtkWidget*  gtk_check_menu_item_new_with_label
                                            (const gchar *label);
```

Creates a new GtkCheckMenuItem with a label.

label : the string to use for the label.
Returns : a new GtkCheckMenuItem.

---

gtk_check_menu_item_new_with_mnemonic ()

```
GtkWidget*  gtk_check_menu_item_new_with_mnemonic
                                               (const gchar *label);
```

Creates a new GtkCheckMenuItem containing a label. The label will be created using `gtk_label_new_with_mnemonic()`, so underscores in *`label`* indicate the mnemonic for the menu item.

`label` : The text of the button, with an underscore in front of the mnemonic character
Returns : a new GtkCheckMenuItem

---

gtk_check_menu_item_set_state

```
#define gtk_check_menu_item_set_state           gtk_check_menu_item_set_active
```

Warning

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

This macro is provided to preserve compatibility with older code. New code should use `gtk_check_menu_item_set_active()` function instead.

---

gtk_check_menu_item_get_active ()

```
gboolean    gtk_check_menu_item_get_active  (GtkCheckMenuItem *check_menu_item);
```

Returns whether the check menu item is active. See `gtk_check_menu_item_set_active()`.

`check_menu_item` : a GtkCheckMenuItem
Returns : TRUE if the menu item is checked.

---

gtk_check_menu_item_set_active ()

```
void        gtk_check_menu_item_set_active  (GtkCheckMenuItem *check_menu_item,
                                             gboolean is_active);
```

Sets the active state of the menu item's check box.

check_menu_item : a GtkCheckMenuItem.

is_active : boolean value indicating whether the check box is active.

---

gtk_check_menu_item_set_show_toggle ()

```
void        gtk_check_menu_item_set_show_toggle
                                            (GtkCheckMenuItem *menu_item,
                                             gboolean always);
```

Warning

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

Controls whether the check box is shown at all times. Normally the check box is shown only when it is active or while the menu item is selected.

menu_item : a GtkCheckMenuItem.

always : boolean value indicating whether to always show the check box.

---

gtk_check_menu_item_toggled ()

```
void        gtk_check_menu_item_toggled     (GtkCheckMenuItem *check_menu_item);
```

Emits the GtkCheckMenuItem::toggled signal.

check_menu_item : a GtkCheckMenuItem.

---

gtk_check_menu_item_get_inconsistent ()

```
gboolean    gtk_check_menu_item_get_inconsistent
                                            (GtkCheckMenuItem *check_menu_item);
```

Retrieves the value set by `gtk_check_menu_item_set_inconsistent()`.

check_menu_item : a GtkCheckMenuItem

Returns : TRUE if inconsistent

---

gtk_check_menu_item_set_inconsistent ()

```
void        gtk_check_menu_item_set_inconsistent
                                        (GtkCheckMenuItem *check_menu_item,
                                         gboolean setting);
```

If the user has selected a range of elements (such as some text or spreadsheet cells) that are affected by a boolean setting, and the current values in that range are inconsistent, you may want to display the check in an "in between" state. This function turns on "in between" display. Normally you would turn off the inconsistent state again if the user explicitly selects a setting. This has to be done manually, `gtk_check_menu_item_set_inconsistent()` only affects visual appearance, it doesn't affect the semantics of the widget.

check_menu_item : a GtkCheckMenuItem

setting : TRUE to display an "inconsistent" third state check

---

gtk_check_menu_item_set_draw_as_radio ()

```
void        gtk_check_menu_item_set_draw_as_radio
                                        (GtkCheckMenuItem *check_menu_item,
                                         gboolean draw_as_radio);
```

Sets whether *check_menu_item* is drawn like a GtkRadioMenuItem

check_menu_item : a GtkCheckMenuItem

draw_as_radio : whether *check_menu_item* is drawn like a GtkRadioMenuItem

Since 2.4

---

gtk_check_menu_item_get_draw_as_radio ()

```
gboolean    gtk_check_menu_item_get_draw_as_radio
                                        (GtkCheckMenuItem *check_menu_item);
```

Returns whether *check_menu_item* looks like a GtkRadioMenuItem

check_menu_item : a GtkCheckMenuItem
Returns : Whether *check_menu_item* looks like a GtkRadioMenuItem

Since 2.4

Properties

The "active" property

```
"active"                gboolean                : Read / Write
```

Whether the menu item is checked.

Default value: FALSE

The "draw-as-radio" property

```
"draw-as-radio"         gboolean                : Read / Write
```

Whether the menu item looks like a radio menu item.

Default value: FALSE

The "inconsistent" property

```
"inconsistent"          gboolean                : Read / Write
```

Whether to display an "inconsistent" state.

Default value: FALSE

Style Properties

The "indicator-size" style property

```
  "indicator-size"       gint                  : Read
```

Size of check or radio indicator.

Allowed values: >= 0

Default value: 12

Signals

The "toggled" signal

```
void        user_function                 (GtkCheckMenuItem *checkmenuitem,
                                           gpointer user_data);
```

This signal is emitted when the state of the check box is changed.

A signal handler can examine the *active* field of the GtkCheckMenuItem-struct struct to discover the new state.

checkmenuitem : the object which received the signal.
user_data : user data set when the signal handler was connected.

<< GtkRadioMenuItem **GtkSeparatorMenuItem >>**

GtkSeparatorMenuItem

GtkSeparatorMenuItem — A separator used in menus

Synopsis

```
#include <gtk/gtk.h>

            GtkSeparatorMenuItem;
GtkWidget*  gtk_separator_menu_item_new     (void);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkWidget
               +----GtkContainer
                     +----GtkBin
                           +----GtkItem
                                 +----GtkMenuItem
                                       +----GtkSeparatorMenuItem
```

Implemented Interfaces

GtkSeparatorMenuItem implements AtkImplementorIface.

Description

The GtkSeparatorMenuItem is a separator used to group items within a menu. It displays a horizontal line with a shadow to make it appear sunken into the interface.

Details

GtkSeparatorMenuItem

```
typedef struct _GtkSeparatorMenuItem GtkSeparatorMenuItem;
```

The GtkSeparatorMenuItem-struct struct contains private data only, and should be accessed using the functions below.

---

gtk_separator_menu_item_new ()

```
GtkWidget*  gtk_separator_menu_item_new     (void);
```

Creates a new GtkSeparatorMenuItem.

Returns : a new GtkSeparatorMenuItem.

<< GtkCheckMenuItem **GtkTearoffMenuItem >>**

GtkTearoffMenuItem

GtkTearoffMenuItem — A menu item used to tear off and reattach its menu

Synopsis

```
#include <gtk/gtk.h>

            GtkTearoffMenuItem;
GtkWidget*  gtk_tearoff_menu_item_new       (void);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkWidget
               +----GtkContainer
                     +----GtkBin
                           +----GtkItem
                                 +----GtkMenuItem
                                       +----GtkTearoffMenuItem
```

Implemented Interfaces

GtkTearoffMenuItem implements AtkImplementorIface.

Description

A GtkTearoffMenuItem is a special GtkMenuItem which is used to tear off and reattach its menu.

When its menu is shown normally, the GtkTearoffMenuItem is drawn as a dotted line indicating that the menu can be torn off. Activating it causes its menu to be torn off and displayed in its own window as a tearoff menu.

When its menu is shown as a tearoff menu, the GtkTearoffMenuItem is drawn as a dotted line which has a left pointing arrow graphic indicating that the tearoff menu can be reattached. Activating it will erase the tearoff menu window.

Details

GtkTearoffMenuItem

```
typedef struct _GtkTearoffMenuItem GtkTearoffMenuItem;
```

The GtkTearoffMenuItem-struct struct contains private data only, and should be accessed using the functions below.

---

gtk_tearoff_menu_item_new ()

```
GtkWidget*  gtk_tearoff_menu_item_new       (void);
```

Creates a new GtkTearoffMenuItem.

Returns : a new GtkTearoffMenuItem.

See Also

GtkMenu for further discussion of menus in GTK.

<< GtkSeparatorMenuItem **GtkToolbar >>**

GtkToolbar

GtkToolbar — Create bars of buttons and other widgets

Synopsis

```
#include <gtk/gtk.h>

            GtkToolbar;
enum        GtkToolbarChildType;
enum        GtkToolbarSpaceStyle;
            GtkToolbarChild;
GtkWidget*  gtk_toolbar_new                 (void);
void        gtk_toolbar_insert              (GtkToolbar *toolbar,
                                             GtkToolItem *item,
                                             gint pos);
gint        gtk_toolbar_get_item_index      (GtkToolbar *toolbar,
                                             GtkToolItem *item);
gint        gtk_toolbar_get_n_items         (GtkToolbar *toolbar);
GtkToolItem* gtk_toolbar_get_nth_item       (GtkToolbar *toolbar,
                                             gint n);
gint        gtk_toolbar_get_drop_index      (GtkToolbar *toolbar,
                                             gint x,
                                             gint y);
void        gtk_toolbar_set_drop_highlight_item
                                            (GtkToolbar *toolbar,
                                             GtkToolItem *tool_item,
                                             gint index_);
void        gtk_toolbar_set_show_arrow      (GtkToolbar *toolbar,
                                             gboolean show_arrow);
void        gtk_toolbar_set_orientation     (GtkToolbar *toolbar,
                                             GtkOrientation orientation);
void        gtk_toolbar_set_tooltips        (GtkToolbar *toolbar,
                                             gboolean enable);
void        gtk_toolbar_unset_icon_size     (GtkToolbar *toolbar);
gboolean    gtk_toolbar_get_show_arrow      (GtkToolbar *toolbar);
GtkOrientation gtk_toolbar_get_orientation  (GtkToolbar *toolbar);
```

```
GtkToolbarStyle gtk_toolbar_get_style       (GtkToolbar *toolbar);
GtkIconSize gtk_toolbar_get_icon_size       (GtkToolbar *toolbar);
gboolean    gtk_toolbar_get_tooltips        (GtkToolbar *toolbar);
GtkReliefStyle gtk_toolbar_get_relief_style (GtkToolbar *toolbar);
GtkWidget*  gtk_toolbar_append_item         (GtkToolbar *toolbar,
                                             const char *text,
                                             const char *tooltip_text,
                                             const char *tooltip_private_text,
                                             GtkWidget *icon,
                                             GtkSignalFunc callback,
                                             gpointer user_data);
GtkWidget*  gtk_toolbar_prepend_item        (GtkToolbar *toolbar,
                                             const char *text,
                                             const char *tooltip_text,
                                             const char *tooltip_private_text,
                                             GtkWidget *icon,
                                             GtkSignalFunc callback,
                                             gpointer user_data);
GtkWidget*  gtk_toolbar_insert_item         (GtkToolbar *toolbar,
                                             const char *text,
                                             const char *tooltip_text,
                                             const char *tooltip_private_text,
                                             GtkWidget *icon,
                                             GtkSignalFunc callback,
                                             gpointer user_data,
                                             gint position);
void        gtk_toolbar_append_space        (GtkToolbar *toolbar);
void        gtk_toolbar_prepend_space       (GtkToolbar *toolbar);
void        gtk_toolbar_insert_space        (GtkToolbar *toolbar,
                                             gint position);
GtkWidget*  gtk_toolbar_append_element      (GtkToolbar *toolbar,
                                             GtkToolbarChildType type,
                                             GtkWidget *widget,
                                             const char *text,
                                             const char *tooltip_text,
                                             const char *tooltip_private_text,
                                             GtkWidget *icon,
                                             GtkSignalFunc callback,
                                             gpointer user_data);
GtkWidget*  gtk_toolbar_prepend_element     (GtkToolbar *toolbar,
                                             GtkToolbarChildType type,
                                             GtkWidget *widget,
                                             const char *text,
                                             const char *tooltip_text,
                                             const char *tooltip_private_text,
                                             GtkWidget *icon,
```

```
                                        GtkSignalFunc callback,
                                        gpointer user_data);
GtkWidget*  gtk_toolbar_insert_element  (GtkToolbar *toolbar,
                                        GtkToolbarChildType type,
                                        GtkWidget *widget,
                                        const char *text,
                                        const char *tooltip_text,
                                        const char *tooltip_private_text,
                                        GtkWidget *icon,
                                        GtkSignalFunc callback,
                                        gpointer user_data,
                                        gint position);
void        gtk_toolbar_append_widget   (GtkToolbar *toolbar,
                                        GtkWidget *widget,
                                        const char *tooltip_text,
                                        const char *tooltip_private_text);
void        gtk_toolbar_prepend_widget  (GtkToolbar *toolbar,
                                        GtkWidget *widget,
                                        const char *tooltip_text,
                                        const char *tooltip_private_text);
void        gtk_toolbar_insert_widget   (GtkToolbar *toolbar,
                                        GtkWidget *widget,
                                        const char *tooltip_text,
                                        const char *tooltip_private_text,
                                        gint position);
void        gtk_toolbar_set_style       (GtkToolbar *toolbar,
                                        GtkToolbarStyle style);
GtkWidget*  gtk_toolbar_insert_stock    (GtkToolbar *toolbar,
                                        const gchar *stock_id,
                                        const char *tooltip_text,
                                        const char *tooltip_private_text,
                                        GtkSignalFunc callback,
                                        gpointer user_data,
                                        gint position);
void        gtk_toolbar_set_icon_size   (GtkToolbar *toolbar,
                                        GtkIconSize icon_size);
void        gtk_toolbar_remove_space    (GtkToolbar *toolbar,
                                        gint position);
void        gtk_toolbar_unset_style     (GtkToolbar *toolbar);
```

Object Hierarchy

```
GObject
 +----GtkObject
       +----GtkWidget
             +----GtkContainer
                   +----GtkToolbar
```

Implemented Interfaces

GtkToolbar implements AtkImplementorIface.

Properties

```
"orientation"          GtkOrientation        : Read / Write
"show-arrow"           gboolean              : Read / Write
"toolbar-style"        GtkToolbarStyle       : Read / Write
```

Child Properties

```
"expand"               gboolean              : Read / Write
"homogeneous"          gboolean              : Read / Write
```

Style Properties

```
"button-relief"        GtkReliefStyle        : Read
"internal-padding"     gint                  : Read
"shadow-type"          GtkShadowType         : Read
"space-size"           gint                  : Read
"space-style"          GtkToolbarSpaceStyle  : Read
```

Signal Prototypes

```
"focus-home-or-end"
            gboolean    user_function      (GtkToolbar *toolbar,
```

```
                                                gboolean focus_home,
                                                gpointer user_data);
"move-focus"
            gboolean    user_function      (GtkToolbar *toolbar,
                                                GtkDirectionType dir,
                                                gpointer user_data);
"orientation-changed"
            void        user_function      (GtkToolbar *toolbar,
                                                GtkOrientation orientation,
                                                gpointer user_data);
"popup-context-menu"
            gboolean    user_function      (GtkToolbar *toolbar,
                                                gint x,
                                                gint y,
                                                gint button,
                                                gpointer user_data);
"style-changed"
            void        user_function      (GtkToolbar *toolbar,
                                                GtkToolbarStyle style,
                                                gpointer user_data);
```

Description

A toolbar is created with a call to `gtk_toolbar_new()`.

A toolbar can contain instances of a subclass of GtkToolItem. To add a GtkToolItem to the a toolbar, use `gtk_toolbar_insert()`. To remove an item from the toolbar use `gtk_container_remove()`. To add a button to the toolbar, add an instance of GtkToolButton.

Toolbar items can be visually grouped by adding instances of GtkSeparatorToolItem to the toolbar. If a GtkSeparatorToolItem has the "expand" property set to TRUE and the "draw" property set to FALSE the effect is to force all following items to the end of the toolbar.

Creating a context menu for the toolbar can be done by connecting to the GtkToolbar::popup-context-menu signal.

Details

GtkToolbar

```
typedef struct {
  gint             num_children;
  GList           *children;
```

```
  GtkOrientation   orientation;
  GtkToolbarStyle  style;
  GtkIconSize      icon_size;

  GtkTooltips     *tooltips;
} GtkToolbar;
```

The GtkToolbar struct only contains private data and should only be accessed through the function described below.

---

enum GtkToolbarChildType

```
typedef enum
{
  GTK_TOOLBAR_CHILD_SPACE,
  GTK_TOOLBAR_CHILD_BUTTON,
  GTK_TOOLBAR_CHILD_TOGGLEBUTTON,
  GTK_TOOLBAR_CHILD_RADIOBUTTON,
  GTK_TOOLBAR_CHILD_WIDGET
} GtkToolbarChildType;
```

Warning

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

GtkToolbarChildType is used to set the type of new elements that are added to a GtkToolbar.

`GTK_TOOLBAR_CHILD_SPACE`	a space in the style of the toolbar's GtkToolbarSpaceStyle.
`GTK_TOOLBAR_CHILD_BUTTON`	a GtkButton.
`GTK_TOOLBAR_CHILD_TOGGLEBUTTON`	a GtkToggleButton.
`GTK_TOOLBAR_CHILD_RADIOBUTTON`	a GtkRadioButton.
`GTK_TOOLBAR_CHILD_WIDGET`	a standard GtkWidget.

---

enum GtkToolbarSpaceStyle

```
typedef enum
{
  GTK_TOOLBAR_SPACE_EMPTY,
  GTK_TOOLBAR_SPACE_LINE
} GtkToolbarSpaceStyle;
```

GtkToolbarChild

```
typedef struct {
  GtkToolbarChildType type;
  GtkWidget *widget;
  GtkWidget *icon;
  GtkWidget *label;
} GtkToolbarChild;
```

Warning

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

---

gtk_toolbar_new ()

```
GtkWidget*  gtk_toolbar_new                 (void);
```

Creates a new toolbar.

Returns : the newly-created toolbar.

---

gtk_toolbar_insert ()

```
void        gtk_toolbar_insert              (GtkToolbar *toolbar,
                                             GtkToolItem *item,
                                             gint pos);
```

Insert a GtkToolItem into the toolbar at position *pos*. If *pos* is 0 the item is prepended to the start of the toolbar. If *pos* is negative, the item is appended to the end of the toolbar.

toolbar : a GtkToolbar
item : a GtkToolItem
pos : the position of the new item

Since 2.4

---

gtk_toolbar_get_item_index ()

```
gint        gtk_toolbar_get_item_index      (GtkToolbar *toolbar,
                                             GtkToolItem *item);
```

Returns the position of *item* on the toolbar, starting from 0. It is an error if *item* is not a child of the toolbar.

toolbar : a GtkToolbar
item : a GtkToolItem that is a child of *toolbar*
Returns : the position of item on the toolbar.

Since 2.4

---

gtk_toolbar_get_n_items ()

```
gint        gtk_toolbar_get_n_items         (GtkToolbar *toolbar);
```

Returns the number of items on the toolbar.

toolbar : a GtkToolbar
Returns : the number of items on the toolbar

Since 2.4

---

gtk_toolbar_get_nth_item ()

```
GtkToolItem* gtk_toolbar_get_nth_item       (GtkToolbar *toolbar,
                                             gint n);
```

Returns the *n*'s item on *toolbar*, or NULL if the toolbar does not contain an *n*'th item.

toolbar : a GtkToolbar
n : A position on the toolbar
Returns : The *n*'th GtkToolItem on *toolbar*, or NULL if there isn't an *n*th item.

Since 2.4

---

gtk_toolbar_get_drop_index ()

```
gint        gtk_toolbar_get_drop_index      (GtkToolbar *toolbar,
                                             gint x,
                                             gint y);
```

Returns the position corresponding to the indicated point on *toolbar*. This is useful when dragging items to the toolbar: this function returns the position a new item should be inserted.

x and *y* are in *toolbar* coordinates.

toolbar : a GtkToolbar
x : x coordinate of a point on the toolbar
y : y coordinate of a point on the toolbar
Returns : The position corresponding to the point (*x*, *y*) on the toolbar.

Since 2.4

---

gtk_toolbar_set_drop_highlight_item ()

```
void        gtk_toolbar_set_drop_highlight_item
                                            (GtkToolbar *toolbar,
                                             GtkToolItem *tool_item,
                                             gint index_);
```

Highlights *toolbar* to give an idea of what it would look like if *item* was added to *toolbar* at the position indicated by *index_*. If *item* is NULL, highlighting is turned off. In that case *index_* is ignored.

The *tool_item* passed to this function must not be part of any widget hierarchy. When an item is set as drop highlight item it can not added to any widget hierarchy or used as highlight item for another toolbar.

toolbar : a GtkToolbar
tool_item : a GtkToolItem, or NULL to turn of highlighting
*index_* : a position on *toolbar*

Since 2.4

---

gtk_toolbar_set_show_arrow ()

```
void        gtk_toolbar_set_show_arrow      (GtkToolbar *toolbar,
                                             gboolean show_arrow);
```

Sets whether to show an overflow menu when *toolbar* doesn't have room for all items on it. If TRUE, items that there are not room are available through an overflow menu.

toolbar : a GtkToolbar
show_arrow : Whether to show an overflow menu

Since 2.4

---

gtk_toolbar_set_orientation ()

```
void        gtk_toolbar_set_orientation     (GtkToolbar *toolbar,
                                             GtkOrientation orientation);
```

Sets whether a toolbar should appear horizontally or vertically.

toolbar : a GtkToolbar.
orientation : a new GtkOrientation.

---

gtk_toolbar_set_tooltips ()

```
void        gtk_toolbar_set_tooltips        (GtkToolbar *toolbar,
```

```
                                                gboolean enable);
```

Sets if the tooltips of a toolbar should be active or not.

toolbar : a GtkToolbar.
enable : set to FALSE to disable the tooltips, or TRUE to enable them.

---

gtk_toolbar_unset_icon_size ()

```
void        gtk_toolbar_unset_icon_size     (GtkToolbar *toolbar);
```

Warning

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

Unsets toolbar icon size set with `gtk_toolbar_set_icon_size()`, so that user preferences will be used to determine the icon size.

toolbar : a GtkToolbar

---

gtk_toolbar_get_show_arrow ()

```
gboolean    gtk_toolbar_get_show_arrow      (GtkToolbar *toolbar);
```

Returns whether the toolbar has an overflow menu. See `gtk_toolbar_set_show_arrow()`

toolbar : a GtkToolbar
Returns :

Since 2.4

---

gtk_toolbar_get_orientation ()

```
GtkOrientation gtk_toolbar_get_orientation  (GtkToolbar *toolbar);
```

Retrieves the current orientation of the toolbar. See `gtk_toolbar_set_orientation()`.

`toolbar` : a GtkToolbar
Returns : the orientation

---

gtk_toolbar_get_style ()

```
GtkToolbarStyle gtk_toolbar_get_style       (GtkToolbar *toolbar);
```

Retrieves whether the toolbar has text, icons, or both . See `gtk_toolbar_set_style()`.

`toolbar` : a GtkToolbar
Returns : the current style of *`toolbar`*

---

gtk_toolbar_get_icon_size ()

```
GtkIconSize gtk_toolbar_get_icon_size       (GtkToolbar *toolbar);
```

Retrieves the icon size fo the toolbar. See `gtk_toolbar_set_icon_size()`.

`toolbar` : a GtkToolbar
Returns : the current icon size for the icons on the toolbar.

---

gtk_toolbar_get_tooltips ()

```
gboolean    gtk_toolbar_get_tooltips        (GtkToolbar *toolbar);
```

Retrieves whether tooltips are enabled. See `gtk_toolbar_set_tooltips()`.

`toolbar` : a GtkToolbar
Returns : TRUE if tooltips are enabled

---

gtk_toolbar_get_relief_style ()

```
GtkReliefStyle gtk_toolbar_get_relief_style (GtkToolbar *toolbar);
```

Returns the relief style of buttons on *toolbar*. See gtk_button_set_relief().

toolbar : a GtkToolbar
Returns : The relief style of buttons on *toolbar*.

Since 2.4

---

gtk_toolbar_append_item ()

```
GtkWidget*  gtk_toolbar_append_item         (GtkToolbar *toolbar,
                                             const char *text,
                                             const char *tooltip_text,
                                             const char *tooltip_private_text,
                                             GtkWidget *icon,
                                             GtkSignalFunc callback,
                                             gpointer user_data);
```

Warning

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

Inserts a new item into the toolbar. You must specify the position in the toolbar where it will be inserted.

callback must be a pointer to a function taking a GtkWidget and a gpointer as arguments. Use the GTK_SIGNAL_FUNC() to cast the function to GtkSignalFunc.

toolbar : a GtkToolbar.
text : give your toolbar button a label.
tooltip_text : a string that appears when the user holds the mouse over this item.
tooltip_private_text : use with GtkTipsQuery.
icon : a GtkWidget that should be used as the button's icon.
callback : the function to be executed when the button is pressed.
user_data : a pointer to any data you wish to be passed to the callback.

Returns : the new toolbar item as a GtkWidget.

---

gtk_toolbar_prepend_item ()

```
GtkWidget*  gtk_toolbar_prepend_item        (GtkToolbar *toolbar,
                                             const char *text,
                                             const char *tooltip_text,
                                             const char *tooltip_private_text,
                                             GtkWidget *icon,
                                             GtkSignalFunc callback,
                                             gpointer user_data);
```

Warning

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

Adds a new button to the beginning (top or left edges) of the given toolbar.

`callback` must be a pointer to a function taking a GtkWidget and a gpointer as arguments. Use the `GTK_SIGNAL_FUNC()` to cast the function to GtkSignalFunc.

`toolbar` :	a GtkToolbar.
`text` :	give your toolbar button a label.
`tooltip_text` :	a string that appears when the user holds the mouse over this item.
`tooltip_private_text` :	use with GtkTipsQuery.
`icon` :	a GtkWidget that should be used as the button's icon.
`callback` :	the function to be executed when the button is pressed.
`user_data` :	a pointer to any data you wish to be passed to the callback.
Returns :	the new toolbar item as a GtkWidget.

---

gtk_toolbar_insert_item ()

```
GtkWidget*  gtk_toolbar_insert_item         (GtkToolbar *toolbar,
                                             const char *text,
                                             const char *tooltip_text,
                                             const char *tooltip_private_text,
                                             GtkWidget *icon,
                                             GtkSignalFunc callback,
```

```
                                                gpointer user_data,
                                                gint position);
```

Warning

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

Inserts a new item into the toolbar. You must specify the position in the toolbar where it will be inserted.

`callback` must be a pointer to a function taking a GtkWidget and a gpointer as arguments. Use the `GTK_SIGNAL_FUNC()` to cast the function to GtkSignalFunc.

`toolbar` :	a GtkToolbar.
`text` :	give your toolbar button a label.
`tooltip_text` :	a string that appears when the user holds the mouse over this item.
`tooltip_private_text` :	use with GtkTipsQuery.
`icon` :	a GtkWidget that should be used as the button's icon.
`callback` :	the function to be executed when the button is pressed.
`user_data` :	a pointer to any data you wish to be passed to the callback.
`position` :	the number of widgets to insert this item after.
Returns :	the new toolbar item as a GtkWidget.

---

gtk_toolbar_append_space ()

```
void        gtk_toolbar_append_space        (GtkToolbar *toolbar);
```

Warning

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

Adds a new space to the end of the toolbar.

`toolbar` : a GtkToolbar.

---

gtk_toolbar_prepend_space ()

```
void        gtk_toolbar_prepend_space       (GtkToolbar *toolbar);
```

Warning

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

Adds a new space to the beginning of the toolbar.

`toolbar` : a GtkToolbar.

---

gtk_toolbar_insert_space ()

```
void        gtk_toolbar_insert_space        (GtkToolbar *toolbar,
                                             gint position);
```

Warning

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

Inserts a new space in the toolbar at the specified position.

`toolbar` : a GtkToolbar

`position` : the number of widgets after which a space should be inserted.

---

gtk_toolbar_append_element ()

```
GtkWidget*  gtk_toolbar_append_element      (GtkToolbar *toolbar,
                                             GtkToolbarChildType type,
                                             GtkWidget *widget,
                                             const char *text,
                                             const char *tooltip_text,
                                             const char *tooltip_private_text,
                                             GtkWidget *icon,
                                             GtkSignalFunc callback,
                                             gpointer user_data);
```

Warning

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

Adds a new element to the end of a toolbar.

If *type* == GTK_TOOLBAR_CHILD_WIDGET, *widget* is used as the new element. If *type* == GTK_TOOLBAR_CHILD_RADIOBUTTON, *widget* is used to determine the radio group for the new element. In all other cases, *widget* must be NULL.

callback must be a pointer to a function taking a GtkWidget and a gpointer as arguments. Use the GTK_SIGNAL_FUNC() to cast the function to GtkSignalFunc.

toolbar :	a GtkToolbar.
type :	a value of type GtkToolbarChildType that determines what *widget* will be.
widget :	a GtkWidget, or NULL.
text :	the element's label.
tooltip_text :	the element's tooltip.
tooltip_private_text :	used for context-sensitive help about this toolbar element.
icon :	a GtkWidget that provides pictorial representation of the element's function.
callback :	the function to be executed when the button is pressed.
user_data :	any data you wish to pass to the callback.
Returns :	the new toolbar element as a GtkWidget.

---

gtk_toolbar_prepend_element ()

```
GtkWidget*  gtk_toolbar_prepend_element     (GtkToolbar *toolbar,
                                             GtkToolbarChildType type,
                                             GtkWidget *widget,
                                             const char *text,
                                             const char *tooltip_text,
                                             const char *tooltip_private_text,
                                             GtkWidget *icon,
                                             GtkSignalFunc callback,
                                             gpointer user_data);
```

Warning

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

Adds a new element to the beginning of a toolbar.

If *type* == GTK_TOOLBAR_CHILD_WIDGET, *widget* is used as the new element. If *type* == GTK_TOOLBAR_CHILD_RADIOBUTTON, *widget* is used to determine the radio group for the new element. In all other cases, *widget* must be NULL.

callback must be a pointer to a function taking a GtkWidget and a gpointer as arguments. Use the GTK_SIGNAL_FUNC() to cast the function to GtkSignalFunc.

toolbar :	a GtkToolbar.
type :	a value of type GtkToolbarChildType that determines what *widget* will be.
widget :	a GtkWidget, or NULL
text :	the element's label.
tooltip_text :	the element's tooltip.
tooltip_private_text :	used for context-sensitive help about this toolbar element.
icon :	a GtkWidget that provides pictorial representation of the element's function.
callback :	the function to be executed when the button is pressed.
user_data :	any data you wish to pass to the callback.
Returns :	the new toolbar element as a GtkWidget.

---

gtk_toolbar_insert_element ()

```
GtkWidget*  gtk_toolbar_insert_element      (GtkToolbar *toolbar,
                                             GtkToolbarChildType type,
                                             GtkWidget *widget,
                                             const char *text,
                                             const char *tooltip_text,
                                             const char *tooltip_private_text,
                                             GtkWidget *icon,
                                             GtkSignalFunc callback,
                                             gpointer user_data,
                                             gint position);
```

Warning

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

Inserts a new element in the toolbar at the given position.

If *type* == GTK_TOOLBAR_CHILD_WIDGET, *widget* is used as the new element. If *type* ==

GTK_TOOLBAR_CHILD_RADIOBUTTON, *widget* is used to determine the radio group for the new element. In all other cases, *widget* must be NULL.

callback must be a pointer to a function taking a GtkWidget and a gpointer as arguments. Use the GTK_SIGNAL_FUNC() to cast the function to GtkSignalFunc.

toolbar :	a GtkToolbar.
type :	a value of type GtkToolbarChildType that determines what *widget* will be.
widget :	a GtkWidget, or NULL.
text :	the element's label.
tooltip_text :	the element's tooltip.
tooltip_private_text :	used for context-sensitive help about this toolbar element.
icon :	a GtkWidget that provides pictorial representation of the element's function.
callback :	the function to be executed when the button is pressed.
user_data :	any data you wish to pass to the callback.
position :	the number of widgets to insert this element after.
Returns :	the new toolbar element as a GtkWidget.

---

gtk_toolbar_append_widget ()

```
void        gtk_toolbar_append_widget       (GtkToolbar *toolbar,
                                             GtkWidget *widget,
                                             const char *tooltip_text,
                                             const char *tooltip_private_text);
```

Warning

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

Adds a widget to the end of the given toolbar.

toolbar :	a GtkToolbar.
widget :	a GtkWidget to add to the toolbar.
tooltip_text :	the element's tooltip.
tooltip_private_text :	used for context-sensitive help about this toolbar element.

---

gtk_toolbar_prepend_widget ()

```
void        gtk_toolbar_prepend_widget      (GtkToolbar *toolbar,
                                             GtkWidget *widget,
                                             const char *tooltip_text,
                                             const char *tooltip_private_text);
```

Warning

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

Adds a widget to the beginning of the given toolbar.

toolbar :	a GtkToolbar.
widget :	a GtkWidget to add to the toolbar.
tooltip_text :	the element's tooltip.
tooltip_private_text :	used for context-sensitive help about this toolbar element.

---

gtk_toolbar_insert_widget ()

```
void        gtk_toolbar_insert_widget       (GtkToolbar *toolbar,
                                             GtkWidget *widget,
                                             const char *tooltip_text,
                                             const char *tooltip_private_text,
                                             gint position);
```

Warning

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

Inserts a widget in the toolbar at the given position.

toolbar :	a GtkToolbar.
widget :	a GtkWidget to add to the toolbar.
tooltip_text :	the element's tooltip.
tooltip_private_text :	used for context-sensitive help about this toolbar element.
position :	the number of widgets to insert this widget after.

---

gtk_toolbar_set_style ()

```
void        gtk_toolbar_set_style           (GtkToolbar *toolbar,
                                             GtkToolbarStyle style);
```

Alters the view of *toolbar* to display either icons only, text only, or both.

toolbar : a GtkToolbar.
style : the new style for *toolbar*.

---

gtk_toolbar_insert_stock ()

```
GtkWidget*  gtk_toolbar_insert_stock        (GtkToolbar *toolbar,
                                             const gchar *stock_id,
                                             const char *tooltip_text,
                                             const char *tooltip_private_text,
                                             GtkSignalFunc callback,
                                             gpointer user_data,
                                             gint position);
```

Warning

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

Inserts a stock item at the specified position of the toolbar. If *stock_id* is not a known stock item ID, it's inserted verbatim, except that underscores used to mark mnemonics are removed.

callback must be a pointer to a function taking a GtkWidget and a gpointer as arguments. Use the `GTK_SIGNAL_FUNC()` to cast the function to GtkSignalFunc.

toolbar : A GtkToolbar
stock_id : The id of the stock item you want to insert
tooltip_text : The text in the tooltip of the toolbar button
tooltip_private_text : The private text of the tooltip
callback : The callback called when the toolbar button is clicked.
user_data : user data passed to callback
position : The position the button shall be inserted at. -1 means at the end.
Returns : the inserted widget

---

gtk_toolbar_set_icon_size ()

```
void        gtk_toolbar_set_icon_size       (GtkToolbar *toolbar,
                                             GtkIconSize icon_size);
```

Warning

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

This function sets the size of stock icons in the toolbar. You can call it both before you add the icons and after they've been added. The size you set will override user preferences for the default icon size.

`toolbar` : A GtkToolbar

`icon_size` : The GtkIconSize that stock icons in the toolbar shall have.

---

gtk_toolbar_remove_space ()

```
void        gtk_toolbar_remove_space        (GtkToolbar *toolbar,
                                             gint position);
```

Warning

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

Removes a space from the specified position.

`toolbar` : a GtkToolbar.

`position` : the index of the space to remove.

---

gtk_toolbar_unset_style ()

```
void        gtk_toolbar_unset_style         (GtkToolbar *toolbar);
```

Unsets a toolbar style set with `gtk_toolbar_set_style()`, so that user preferences will be used to determine the toolbar style.

toolbar : a GtkToolbar

Properties

The "orientation" property

```
"orientation"          GtkOrientation        : Read / Write
```

The orientation of the toolbar.

Default value: GTK_ORIENTATION_HORIZONTAL

The "show-arrow" property

```
"show-arrow"           gboolean              : Read / Write
```

If an arrow should be shown if the toolbar doesn't fit.

Default value: TRUE

The "toolbar-style" property

```
"toolbar-style"        GtkToolbarStyle       : Read / Write
```

How to draw the toolbar.

Default value: GTK_TOOLBAR_ICONS

Child Properties

The "expand" child property

```
"expand"               gboolean              : Read / Write
```

Whether the item should receive extra space when the toolbar grows.

Default value: TRUE

The "homogeneous" child property

```
"homogeneous"          gboolean              : Read / Write
```

Whether the item should be the same size as other homogeneous items.

Default value: TRUE

Style Properties

The "button-relief" style property

```
"button-relief"        GtkReliefStyle        : Read
```

Type of bevel around toolbar buttons.

Default value: GTK_RELIEF_NONE

The "internal-padding" style property

```
"internal-padding"     gint                  : Read
```

Amount of border space between the toolbar shadow and the buttons.

Allowed values: >= 0

Default value: 0

The "shadow-type" style property

```
  "shadow-type"          GtkShadowType         : Read
```

Style of bevel around the toolbar.

Default value: GTK_SHADOW_OUT

The "space-size" style property

```
  "space-size"           gint                  : Read
```

Size of spacers.

Allowed values: >= 0

Default value: 12

The "space-style" style property

```
  "space-style"          GtkToolbarSpaceStyle  : Read
```

Whether spacers are vertical lines or just blank.

Default value: GTK_TOOLBAR_SPACE_LINE

Signals

The "focus-home-or-end" signal

```
gboolean    user_function                 (GtkToolbar *toolbar,
                                           gboolean focus_home,
                                           gpointer user_data);
```

A keybinding signal used internally by GTK+. This signal can't be used in application code

toolbar : the GtkToolbar which emitted the signal
focus_home : TRUE if the first item should be focused
user_data : user data set when the signal handler was connected.
Returns : TRUE if the signal was handled, FALSE if not

The "move-focus" signal

```
gboolean    user_function                   (GtkToolbar *toolbar,
                                             GtkDirectionType dir,
                                             gpointer user_data);
```

A keybinding signal used internally by GTK+. This signal can't be used in application code.

toolbar : the GtkToolbar which emitted the signal
dir : a GtkDirection
user_data : user data set when the signal handler was connected.
Returns : TRUE if the signal was handled, FALSE if not

The "orientation-changed" signal

```
void        user_function                   (GtkToolbar *toolbar,
                                             GtkOrientation orientation,
                                             gpointer user_data);
```

Emitted when the orientation of the toolbar changes.

toolbar : the object which emitted the signal
orientation : the new GtkOrientation of the toolbar
user_data : user data set when the signal handler was connected.

The "popup-context-menu" signal

```
gboolean    user_function                   (GtkToolbar *toolbar,
                                             gint x,
                                             gint y,
```

```
                                                gint button,
                                                gpointer user_data);
```

Emitted when the user right-clicks the toolbar or uses the keybinding to display a popup menu.

Application developers should handle this signal if they want to display a context menu on the toolbar. The context-menu should appear at the coordinates given by *x* and *y*. The mouse button number is given by the *button* parameter. If the menu was popped up using the keybaord, *button* is -1.

toolbar :	the GtkToolbar which emitted the signal
x :	the x coordinate of the point where the menu should appear
y :	the y coordinate of the point where the menu should appear
button :	the mouse button the user pressed, or -1
user_data :	user data set when the signal handler was connected.
Returns :	return TRUE if the signal was handled, FALSE if not

---

The "style-changed" signal

```
void        user_function                   (GtkToolbar *toolbar,
                                             GtkToolbarStyle style,
                                             gpointer user_data);
```

Emitted when the style of the toolbar changes.

toolbar :	The GtkToolbar which emitted the signal
style :	the new GtkToolbarStyle of the toolbar
user_data :	user data set when the signal handler was connected.

See Also

GtkToolItem Base class of widgets that can be added to a toolbar.

<< GtkTearoffMenuItem **GtkToolItem >>**

GtkToolItem

GtkToolItem — The base class of widgets that can be added to GtkToolbar

Synopsis

```
#include <gtk/gtk.h>

            GtkToolItem;
GtkToolItem* gtk_tool_item_new              (void);
void        gtk_tool_item_set_homogeneous   (GtkToolItem *tool_item,
                                             gboolean homogeneous);
gboolean    gtk_tool_item_get_homogeneous   (GtkToolItem *tool_item);
void        gtk_tool_item_set_expand        (GtkToolItem *tool_item,
                                             gboolean expand);
gboolean    gtk_tool_item_get_expand        (GtkToolItem *tool_item);
void        gtk_tool_item_set_tooltip       (GtkToolItem *tool_item,
                                             GtkTooltips *tooltips,
                                             const gchar *tip_text,
                                             const gchar *tip_private);
void        gtk_tool_item_set_use_drag_window
                                            (GtkToolItem *toolitem,
                                             gboolean use_drag_window);
gboolean    gtk_tool_item_get_use_drag_window
                                            (GtkToolItem *toolitem);
void        gtk_tool_item_set_visible_horizontal
                                            (GtkToolItem *toolitem,
                                             gboolean visible_horizontal);
gboolean    gtk_tool_item_get_visible_horizontal
                                            (GtkToolItem *toolitem);
void        gtk_tool_item_set_visible_vertical
                                            (GtkToolItem *toolitem,
                                             gboolean visible_vertical);
gboolean    gtk_tool_item_get_visible_vertical
```

```
                                                    (GtkToolItem *toolitem);
void        gtk_tool_item_set_is_important          (GtkToolItem *tool_item,
                                                     gboolean is_important);
gboolean    gtk_tool_item_get_is_important          (GtkToolItem *tool_item);
GtkIconSize gtk_tool_item_get_icon_size             (GtkToolItem *tool_item);
GtkOrientation gtk_tool_item_get_orientation
                                                    (GtkToolItem *tool_item);
GtkToolbarStyle gtk_tool_item_get_toolbar_style
                                                    (GtkToolItem *tool_item);
GtkReliefStyle gtk_tool_item_get_relief_style
                                                    (GtkToolItem *tool_item);
GtkWidget*  gtk_tool_item_retrieve_proxy_menu_item
                                                    (GtkToolItem *tool_item);
GtkWidget*  gtk_tool_item_get_proxy_menu_item
                                                    (GtkToolItem *tool_item,
                                                     const gchar *menu_item_id);
void        gtk_tool_item_set_proxy_menu_item
                                                    (GtkToolItem *tool_item,
                                                     const gchar *menu_item_id,
                                                     GtkWidget *menu_item);
void        gtk_tool_item_rebuild_menu              (GtkToolItem *tool_item);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkWidget
               +----GtkContainer
                     +----GtkBin
                           +----GtkToolItem
                                 +----GtkToolButton
                                 +----GtkSeparatorToolItem
```

Implemented Interfaces

GtkToolItem implements AtkImplementorIface.

Properties

```
  "is-important"          gboolean              : Read / Write
  "visible-horizontal"    gboolean              : Read / Write
  "visible-vertical"      gboolean              : Read / Write
```

Signal Prototypes

```
"create-menu-proxy"
            gboolean    user_function      (GtkToolItem *toolitem,
                                            gpointer user_data);
"set-tooltip"
            gboolean    user_function      (GtkToolItem *toolitem,
                                            GtkTooltips *tooltips,
                                            gchar *tip_text,
                                            gchar *tip_private,
                                            gpointer user_data);
"toolbar-reconfigured"
            void        user_function      (GtkToolItem *toolitem,
                                            gpointer user_data);
```

Description

GtkToolItems are widgets that can appear on a toolbar. To create a toolbar item that contain something else than a button, use `gtk_tool_item_new()`. Use `gtk_container_add()` to add a child widget to the tool item.

For toolbar items that contain buttons, see the GtkToolButton, GtkToggleToolButton and GtkRadioToolButton classes.

See the GtkToolbar class for a description of the toolbar widget.

Details

GtkToolItem

```
typedef struct _GtkToolItem GtkToolItem;
```

The GtkToolItem struct contains only private data. It should only be accessed through the functions described below.

---

gtk_tool_item_new ()

```
GtkToolItem* gtk_tool_item_new              (void);
```

Creates a new GtkToolItem

Returns : the new GtkToolItem

Since 2.4

---

gtk_tool_item_set_homogeneous ()

```
void        gtk_tool_item_set_homogeneous   (GtkToolItem *tool_item,
                                             gboolean homogeneous);
```

Sets whether *tool_item* is to be allocated the same size as other homogeneous items. The effect is that all homogeneous items will have the same width as the widest of the items.

tool_item : a GtkToolItem:

homogeneous : whether *tool_item* is the same size as other homogeneous items

Since 2.4

---

gtk_tool_item_get_homogeneous ()

```
gboolean    gtk_tool_item_get_homogeneous   (GtkToolItem *tool_item);
```

Returns whether *tool_item* is the same size as other homogeneous items. See gtk_tool_item_set_homogeneous().

tool_item : a GtkToolItem:

Returns : TRUE if the item is the same size as other homogeneous item.s

Since 2.4

---

gtk_tool_item_set_expand ()

```
void        gtk_tool_item_set_expand        (GtkToolItem *tool_item,
                                             gboolean expand);
```

Sets whether *tool_item* is allocated extra space when there is more room on the toolbar then needed for the items. The effect is that the item gets bigger when the toolbar gets bigger and smaller when the toolbar gets smaller.

tool_item : a GtkToolItem:

expand : Whether *tool_item* is allocated extra space

Since 2.4

---

gtk_tool_item_get_expand ()

```
gboolean    gtk_tool_item_get_expand        (GtkToolItem *tool_item);
```

Returns whether *tool_item* is allocated extra space. See gtk_tool_item_set_expand().

tool_item : a GtkToolItem:

Returns : TRUE if *tool_item* is allocated extra space.

Since 2.4

---

gtk_tool_item_set_tooltip ()

```
void        gtk_tool_item_set_tooltip       (GtkToolItem *tool_item,
                                             GtkTooltips *tooltips,
                                             const gchar *tip_text,
                                             const gchar *tip_private);
```

Sets the GtkTooltips object to be used for *tool_item*, the text to be displayed as tooltip on the item and the private text to be used. See gtk_tooltips_set_tip().

tool_item : a GtkToolItem:
tooltips : The GtkTooltips object to be used
tip_text : text to be used as tooltip text for *tool_item*
tip_private : text to be used as private tooltip text

Since 2.4

---

gtk_tool_item_set_use_drag_window ()

```
void        gtk_tool_item_set_use_drag_window
                                            (GtkToolItem *toolitem,
                                             gboolean use_drag_window);
```

Sets whether *toolitem* has a drag window. When TRUE the toolitem can be used as a drag source through gtk_drag_source_set(). When *toolitem* has a drag window it will intercept all events, even those that would otherwise be sent to a child of *toolitem*.

toolitem : a GtkToolItem
use_drag_window : Whether *toolitem* has a drag window.

Since 2.4

---

gtk_tool_item_get_use_drag_window ()

```
gboolean    gtk_tool_item_get_use_drag_window
                                            (GtkToolItem *toolitem);
```

Returns whether *toolitem* has a drag window. See gtk_tool_item_set_use_drag_window().

toolitem : a GtkToolItem
Returns : TRUE if *toolitem* uses a drag window.

Since 2.4

---

gtk_tool_item_set_visible_horizontal ()

```
void        gtk_tool_item_set_visible_horizontal
                                            (GtkToolItem *toolitem,
                                             gboolean visible_horizontal);
```

Sets whether *toolitem* is visible when the toolbar is docked horizontally.

toolitem : a GtkToolItem
visible_horizontal : Whether *toolitem* is visible when in horizontal mode

Since 2.4

---

gtk_tool_item_get_visible_horizontal ()

```
gboolean    gtk_tool_item_get_visible_horizontal
```

```
                                                    (GtkToolItem *toolitem);
```

Returns whether the *toolitem* is visible on toolbars that are docked horizontally.

toolitem : a GtkToolItem

Returns : TRUE if *toolitem* is visible on toolbars that are docked horizontally.

Since 2.4

---

gtk_tool_item_set_visible_vertical ()

```
void        gtk_tool_item_set_visible_vertical
                                                    (GtkToolItem *toolitem,
                                                     gboolean visible_vertical);
```

Sets whether *toolitem* is visible when the toolbar is docked vertically. Some tool items, such as text entries, are too wide to be useful on a vertically docked toolbar. If *visible_vertical* is FALSE *toolitem* will not appear on toolbars that are docked vertically.

toolitem : a GtkToolItem

visible_vertical : whether *toolitem* is visible when the toolbar is in vertical mode

Since 2.4

---

gtk_tool_item_get_visible_vertical ()

```
gboolean    gtk_tool_item_get_visible_vertical
                                                    (GtkToolItem *toolitem);
```

Returns whether *toolitem* is visible when the toolbar is docked vertically. See gtk_tool_item_set_visible_vertical().

toolitem : a GtkToolItem

Returns : Whether *toolitem* is visible when the toolbar is docked vertically

Since 2.4

---

gtk_tool_item_set_is_important ()

```
void        gtk_tool_item_set_is_important  (GtkToolItem *tool_item,
                                             gboolean is_important);
```

Sets whether *tool_item* should be considered important. The GtkToolButton class uses this property to determine whether to show or hide its label when the toolbar style is GTK_TOOLBAR_BOTH_HORIZ. The result is that only tool buttons with the "is_important" property set have labels, an effect known as "priority text"

tool_item : a GtkToolItem

is_important : whether the tool item should be considered important

Since 2.4

---

gtk_tool_item_get_is_important ()

```
gboolean    gtk_tool_item_get_is_important  (GtkToolItem *tool_item);
```

Returns whether *tool_item* is considered important. See gtk_tool_item_set_is_important()

tool_item : a GtkToolItem

Returns : TRUE if *tool_item* is considered important.

Since 2.4

---

gtk_tool_item_get_icon_size ()

```
GtkIconSize gtk_tool_item_get_icon_size     (GtkToolItem *tool_item);
```

Returns the icon size used for *tool_item*. Custom subclasses of GtkToolItem should call this function to find out what size icons they should use.

tool_item : a GtkToolItem:

Returns : a GtkIconSize indicating the icon size used for *tool_item*

Since 2.4

---

gtk_tool_item_get_orientation ()

```
GtkOrientation gtk_tool_item_get_orientation
                                            (GtkToolItem *tool_item);
```

Returns the orientation used for *tool_item*. Custom subclasses of GtkToolItem should call this function to find out what size icons they should use.

tool_item : a GtkToolItem:

Returns : a GtkOrientation indicating the orientation used for *tool_item*

Since 2.4

---

gtk_tool_item_get_toolbar_style ()

```
GtkToolbarStyle gtk_tool_item_get_toolbar_style
                                            (GtkToolItem *tool_item);
```

Returns the toolbar style used for *tool_item*. Custom subclasses of GtkToolItem should call this function in the handler of the GtkToolItem::toolbar_reconfigured signal to find out in what style the toolbar is displayed and change themselves accordingly

Possibilities are:

- GTK_TOOLBAR_BOTH, meaning the tool item should show both an icon and a label, stacked vertically
- GTK_TOOLBAR_ICONS, meaning the toolbar shows only icons
- GTK_TOOLBAR_TEXT, meaning the tool item should only show text
- GTK_TOOLBAR_BOTH_HORIZ, meaning the tool item should show both an icon and a label, arranged horizontally (however, note the GtkToolButton::has_text_horizontally that makes tool buttons not show labels when the toolbar style is GTK_TOOLBAR_BOTH_HORIZ.

tool_item : a GtkToolItem:

Returns : A GtkToolbarStyle indicating the toolbar style used for *tool_item*.

Since 2.4

---

gtk_tool_item_get_relief_style ()

```
GtkReliefStyle gtk_tool_item_get_relief_style
                                              (GtkToolItem *tool_item);
```

Returns the relief style of *tool_item*. See gtk_button_set_relief_style(). Custom subclasses of GtkToolItem should call this function in the handler of the GtkToolItem::toolbar_reconfigured signal to find out the relief style of buttons.

tool_item : a GtkToolItem:

Returns : a GtkReliefStyle indicating the relief style used for *tool_item*.

Since 2.4

---

gtk_tool_item_retrieve_proxy_menu_item ()

```
GtkWidget*  gtk_tool_item_retrieve_proxy_menu_item
                                              (GtkToolItem *tool_item);
```

Returns the GtkMenuItem that was last set by gtk_tool_item_set_proxy_menu_item(), ie. the

GtkMenuItem that is going to appear in the overflow menu.

tool_item : a GtkToolItem:

Returns : The GtkMenuItem that is going to appear in the overflow menu for *tool_item*.

Since 2.4

---

gtk_tool_item_get_proxy_menu_item ()

```
GtkWidget*  gtk_tool_item_get_proxy_menu_item
                                            (GtkToolItem *tool_item,
                                             const gchar *menu_item_id);
```

If *menu_item_id* matches the string passed to gtk_tool_item_set_proxy_menu_item() return the corresponding GtkMenuItem.

Custom subclasses of GtkToolItem should use this function to update their menu item when the GtkToolItem changes. That the *menu_item_id*s must match ensures that a GtkToolItem will not inadvertently change a menu item that they did not create.

tool_item : a GtkToolItem:

menu_item_id : a string used to identify the menu item

Returns : The GtkMenuItem passed to gtk_tool_item_set_proxy_menu_item(), if the *menu_item_id*s match.

Since 2.4

---

gtk_tool_item_set_proxy_menu_item ()

```
void        gtk_tool_item_set_proxy_menu_item
                                            (GtkToolItem *tool_item,
                                             const gchar *menu_item_id,
                                             GtkWidget *menu_item);
```

Sets the GtkMenuItem used in the toolbar overflow menu. The *menu_item_id* is used to identify the caller of this function and should also be used with gtk_tool_item_get_proxy_menu_item().

tool_item : a GtkToolItem:
menu_item_id : a string used to identify *menu_item*
menu_item : a GtkMenuItem to be used in the overflow menu

Since 2.4

---

gtk_tool_item_rebuild_menu ()

```
void        gtk_tool_item_rebuild_menu      (GtkToolItem *tool_item);
```

Calling this function signals to the toolbar that the overflow menu item for *tool_item* has changed. If the overflow menu is visible when this function it called, the menu will be rebuilt.

The function must be called when the tool item changes what it will do in response to the "create_menu_proxy" signal.

tool_item : a GtkToolItem

Since 2.6

Properties

The "is-important" property

```
  "is-important"         gboolean              : Read / Write
```

Whether the toolbar item is considered important. When TRUE, toolbar buttons show text in GTK_TOOLBAR_BOTH_HORIZ mode.

Default value: FALSE

The "visible-horizontal" property

```
  "visible-horizontal"   gboolean              : Read / Write
```

Whether the toolbar item is visible when the toolbar is in a horizontal orientation.

Default value: TRUE

The "visible-vertical" property

```
  "visible-vertical"     gboolean              : Read / Write
```

Whether the toolbar item is visible when the toolbar is in a vertical orientation.

Default value: TRUE

Signals

The "create-menu-proxy" signal

```
gboolean    user_function                   (GtkToolItem *toolitem,
                                             gpointer user_data);
```

This signal is emitted when the toolbar needs information from *tool_item* about whether the item should appear in the toolbar overflow menu. In response the tool item should either

- call gtk_tool_item_set_proxy_menu_item() with a NULL pointer and return TRUE to indicate that the item should not appear in the overflow menu
- call gtk_tool_item_set_proxy_menu_item() with a new menu item and return TRUE, or
- return FALSE to indicate that the signal was not handled by the item. This means that the item will not appear in the overflow menu unless a later handler installs a menu item.

The toolbar may cache the result of this signal. When the tool item changes how it will respond to this signal it must call gtk_tool_item_rebuild_menu() to invalidate the cache and ensure that the toolbar rebuilds

its overflow menu.

toolitem : the object the signal was emitted on
user_data : user data set when the signal handler was connected.
Returns : TRUE if the signal was handled, FALSE if not

The "set-tooltip" signal

```
gboolean    user_function                   (GtkToolItem *toolitem,
                                             GtkTooltips *tooltips,
                                             gchar *tip_text,
                                             gchar *tip_private,
                                             gpointer user_data);
```

This signal is emitted when the toolitem's tooltip changes. Application developers can use `gtk_tool_item_set_tooltip()` to set the item's tooltip.

toolitem : the object the signal was emitted on
tooltips : the GtkTooltips
tip_text : the tooltip text
tip_private : the tooltip private text
user_data : user data set when the signal handler was connected.
Returns : TRUE if the signal was handled, FALSE if not

The "toolbar-reconfigured" signal

```
void        user_function                   (GtkToolItem *toolitem,
                                             gpointer user_data);
```

This signal is emitted when some property of the toolbar that the item is a child of changes. For custom subclasses of GtkToolItem, the default handler of this signal use the functions

- `gtk_toolbar_get_orientation()`
- `gtk_toolbar_get_style()`
- `gtk_toolbar_get_icon_size()`

- gtk_toolbar_get_relief_style()

to find out what the toolbar should look like and change themselves accordingly.

toolitem: the object the signal was emitted on
user_data: user data set when the signal handler was connected.

See Also

GtkToolbar

The toolbar widget

GtkToolButton

A subclass of GtkToolItem that displays buttons on the toolbar

GtkSeparatorToolItem

A subclass of GtkToolItem that separates groups of items on a toolbar

<< GtkToolbar **GtkSeparatorToolItem >>**

GtkSeparatorToolItem

GtkSeparatorToolItem — A toolbar item that separates groups of other toolbar items

Synopsis

```
#include <gtk/gtk.h>

            GtkSeparatorToolItem;
GtkToolItem* gtk_separator_tool_item_new    (void);
void        gtk_separator_tool_item_set_draw
                                            (GtkSeparatorToolItem *item,
                                             gboolean draw);
gboolean    gtk_separator_tool_item_get_draw
                                            (GtkSeparatorToolItem *item);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkWidget
               +----GtkContainer
                     +----GtkBin
                           +----GtkToolItem
                                 +----GtkSeparatorToolItem
```

Implemented Interfaces

GtkSeparatorToolItem implements AtkImplementorIface.

Properties

```
  "draw"                  gboolean              : Read / Write
```

Description

A GtkSeparatorItem is a GtkToolItem that separates groups of other GtkToolItems. Depending on the theme, a GtkSeparatorToolItem will often look like a vertical line on horizontally docked toolbars.

If the property "expand" is TRUE and the property "draw" is FALSE, a GtkSeparatorToolItem will act as a "spring" that forces other items to the ends of the toolbar.

Use `gtk_separator_tool_item_new()` to create a new GtkSeparatorToolItem.

Details

GtkSeparatorToolItem

```
typedef struct _GtkSeparatorToolItem GtkSeparatorToolItem;
```

The GtkSeparatorToolItem struct contains only private data and should only be accessed through the functions described below.

---

gtk_separator_tool_item_new ()

```
GtkToolItem* gtk_separator_tool_item_new    (void);
```

Create a new GtkSeparatorToolItem

Returns : the new GtkSeparatorToolItem

Since 2.4

---

gtk_separator_tool_item_set_draw ()

```
void        gtk_separator_tool_item_set_draw
                                            (GtkSeparatorToolItem *item,
                                             gboolean draw);
```

When *separator_tool_items* is drawn as a vertical line, or just blank. Setting this FALSE along with gtk_tool_item_set_expand() is useful to create an item that forces following items to the end of the toolbar.

item : a GtkSeparatorToolItem

draw : whether *separator_tool_item* is drawn as a vertical iln

Since 2.4

---

gtk_separator_tool_item_get_draw ()

```
gboolean    gtk_separator_tool_item_get_draw
                                            (GtkSeparatorToolItem *item);
```

Returns whether *separator_tool_item* is drawn as a line, or just blank. See gtk_separator_tool_item_set_draw().

item : a GtkSeparatorToolItem

Returns : TRUE if *separator_tool_item* is drawn as a line, or just blank.

Since 2.4

Properties

The "draw" property

"draw"	gboolean	: Read / Write

Whether the separator is drawn, or just blank.

Default value: TRUE

See Also

GtkToolbar	The toolbar widget
GtkRadioToolButton	A toolbar item containing a radio button

<< GtkToolItem **GtkToolButton >>**

GtkToolButton

GtkToolButton — A GtkToolItem subclass that displays buttons

Synopsis

```
#include <gtk/gtk.h>

            GtkToolButton;
GtkToolItem* gtk_tool_button_new           (GtkWidget *icon_widget,
                                            const gchar *label);
GtkToolItem* gtk_tool_button_new_from_stock (const gchar *stock_id);
void        gtk_tool_button_set_label      (GtkToolButton *button,
                                            const gchar *label);
G_CONST_RETURN gchar* gtk_tool_button_get_label
                                           (GtkToolButton *button);
void        gtk_tool_button_set_use_underline
                                           (GtkToolButton *button,
                                            gboolean use_underline);
gboolean    gtk_tool_button_get_use_underline
                                           (GtkToolButton *button);
void        gtk_tool_button_set_stock_id   (GtkToolButton *button,
                                            const gchar *stock_id);
G_CONST_RETURN gchar* gtk_tool_button_get_stock_id
                                           (GtkToolButton *button);
void        gtk_tool_button_set_icon_widget (GtkToolButton *button,
                                            GtkWidget *icon_widget);
GtkWidget*  gtk_tool_button_get_icon_widget (GtkToolButton *button);
void        gtk_tool_button_set_label_widget
                                           (GtkToolButton *button,
                                            GtkWidget *label_widget);
GtkWidget*  gtk_tool_button_get_label_widget
```

```
                                                    (GtkToolButton *button);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkWidget
               +----GtkContainer
                     +----GtkBin
                           +----GtkToolItem
                                 +----GtkToolButton
                                       +----GtkMenuToolButton
                                       +----GtkToggleToolButton
```

Implemented Interfaces

GtkToolButton implements AtkImplementorIface.

Properties

```
  "icon-widget"          GtkWidget             : Read / Write
  "label"                gchararray            : Read / Write
  "label-widget"         GtkWidget             : Read / Write
  "stock-id"             gchararray            : Read / Write
  "use-underline"        gboolean              : Read / Write
```

Signal Prototypes

```
"clicked"   void        user_function      (GtkToolButton *toolbutton,
                                            gpointer user_data);
```

Description

GtkToolButtons are GtkToolItems containing buttons.

Use `gtk_tool_button_new()` to create a new GtkToolButton. Use `gtk_tool_button_new_with_stock()` to create a GtkToolButton containing a stock item.

The label of a GtkToolButton is determined by the properties "label_widget", "label", and "stock_id". If "label_widget" is non-`NULL`, then that widget is used as the label. Otherwise, if "label" is non-`NULL`, that string is used as the label. Otherwise, if "stock_id" is non-`NULL`, the label is determined by the stock item. Otherwise, the button does not have a label.

The icon of a GtkToolButton is determined by the properties "icon_widget" and "stock_id". If "icon_widget" is non-`NULL`, then that widget is used as the icon. Otherwise, if "stock_id" is non-`NULL`, the icon is determined by the stock item. Otherwise, the button does not have a label.

Details

GtkToolButton

```
typedef struct _GtkToolButton GtkToolButton;
```

The GtkToolButton struct contains only private. It should only be accessed with the function described below.

---

gtk_tool_button_new ()

```
GtkToolItem* gtk_tool_button_new           (GtkWidget *icon_widget,
                                            const gchar *label);
```

Creates a new `GtkToolButton` using *`icon_widget`* as icon and *`label`* as label.

icon_widget : a widget that will be used as icon widget, or NULL
label : a string that will be used as label, or NULL
Returns : A new GtkToolButton

Since 2.4

---

gtk_tool_button_new_from_stock ()

```
GtkToolItem* gtk_tool_button_new_from_stock (const gchar *stock_id);
```

Creates a new GtkToolButton containing the image and text from a stock item. Some stock ids have preprocessor macros like GTK_STOCK_OK and GTK_STOCK_APPLY.

It is an error if *stock_id* is not a name of a stock item.

stock_id : the name of the stock item
Returns : A new GtkToolButton

Since 2.4

---

gtk_tool_button_set_label ()

```
void        gtk_tool_button_set_label       (GtkToolButton *button,
                                             const gchar *label);
```

Sets *label* as the label used for the tool button. The "label" property only has an effect if not overridden by a non-NULL "label_widget" property. If both the "label_widget" and "label" properties are NULL, the label is determined by the "stock_id" property. If the "stock_id" property is also NULL, *button* will not have a label.

button : a GtkToolButton

label : a string that will be used as label, or NULL.

Since 2.4

---

gtk_tool_button_get_label ()

```
G_CONST_RETURN gchar* gtk_tool_button_get_label
                                                (GtkToolButton *button);
```

Returns the label used by the tool button, or NULL if the tool button doesn't have a label. or uses a the label from a stock item. The returned string is owned by GTK+, and must not be modified or freed.

button : a GtkToolButton

Returns : The label, or NULL

Since 2.4

---

gtk_tool_button_set_use_underline ()

```
void        gtk_tool_button_set_use_underline
                                                (GtkToolButton *button,
                                                 gboolean use_underline);
```

If set, an underline in the label property indicates that the next character should be used for the mnemonic accelerator key in the overflow menu. For example, if the label property is "_Open" and *use_underline* is TRUE, the label on the tool button will be "Open" and the item on the overflow menu will have an underlined 'O'.

Labels shown on tool buttons never have mnemonics on them; this property only affects the menu item on the overflow menu.

button : a GtkToolButton

use_underline : whether the button label has the form "_Open"

Since 2.4

---

gtk_tool_button_get_use_underline ()

```
gboolean    gtk_tool_button_get_use_underline
                                            (GtkToolButton *button);
```

Returns whether underscores in the label property are used as mnemonics on menu items on the overflow menu. See gtk_tool_button_set_use_underline().

button : a GtkToolButton

Returns : TRUE if underscores in the label property are used as mnemonics on menu items on the overflow menu.

Since 2.4

---

gtk_tool_button_set_stock_id ()

```
void        gtk_tool_button_set_stock_id    (GtkToolButton *button,
                                             const gchar *stock_id);
```

Sets the name of the stock item. See gtk_tool_button_new_from_stock(). The stock_id property only has an effect if not overridden by non-NULL "label" and "icon_widget" properties.

button : a GtkToolButton

stock_id : a name of a stock item, or NULL

Since 2.4

gtk_tool_button_get_stock_id ()

```
G_CONST_RETURN gchar* gtk_tool_button_get_stock_id
                                                    (GtkToolButton *button);
```

Returns the name of the stock item. See gtk_tool_button_set_stock_id(). The returned string is owned by GTK+ and must not be freed or modifed.

button : a GtkToolButton

Returns : the name of the stock item for *button*.

Since 2.4

gtk_tool_button_set_icon_widget ()

```
void        gtk_tool_button_set_icon_widget (GtkToolButton *button,
                                             GtkWidget *icon_widget);
```

Sets *icon* as the widget used as icon on *button*. If *icon_widget* is NULL the icon is determined by the "stock_id" property. If the "stock_id" property is also NULL, *button* will not have an icon.

button : a GtkToolButton

icon_widget : the widget used as icon, or NULL

Since 2.4

gtk_tool_button_get_icon_widget ()

```
GtkWidget*  gtk_tool_button_get_icon_widget (GtkToolButton *button);
```

Return the widget used as icon widget on *button*. See gtk_tool_button_set_icon_widget().

button : a GtkToolButton

Returns : The widget used as icon on *button*, or NULL.

Since 2.4

---

gtk_tool_button_set_label_widget ()

```
void        gtk_tool_button_set_label_widget
                                            (GtkToolButton *button,
                                             GtkWidget *label_widget);
```

Sets *label_widget* as the widget that will be used as the label for *button*. If *label_widget* is NULL the "label" property is used as label. If "label" is also NULL, the label in the stock item determined by the "stock_id" property is used as label. If "stock_id" is also NULL, *button* does not have a label.

button : a GtkToolButton

label_widget : the widget used as label, or NULL

Since 2.4

---

gtk_tool_button_get_label_widget ()

```
GtkWidget*  gtk_tool_button_get_label_widget
                                            (GtkToolButton *button);
```

Returns the widget used as label on *button*. See gtk_tool_button_set_label_widget().

button : a GtkToolButton

Returns : The widget used as label on *button*, or NULL.

Since 2.4

Properties

The "icon-widget" property

"icon-widget"	GtkWidget	: Read / Write

Icon widget to display in the item.

The "label" property

"label"	gchararray	: Read / Write

Text to show in the item.

Default value: NULL

The "label-widget" property

"label-widget"	GtkWidget	: Read / Write

Widget to use as the item label.

The "stock-id" property

```
  "stock-id"              gchararray            : Read / Write
```

The stock icon displayed on the item.

Default value: NULL

The "use-underline" property

```
  "use-underline"         gboolean              : Read / Write
```

If set, an underline in the label property indicates that the next character should be used for the mnemonic accelerator key in the overflow menu.

Default value: FALSE

Signals

The "clicked" signal

```
void        user_function                   (GtkToolButton *toolbutton,
                                             gpointer user_data);
```

This signal is emitted when the tool button is clicked with the mouse or activated with the keyboard.

toolbutton : the object that emitted the signal
user_data : user data set when the signal handler was connected.

See Also

GtkToolbar — The toolbar widget

GtkMenuToolButton	A subclass of GtkToolButton that displays on the toolbar a button with an additional dropdown menu
GtkToggleToolButton	A subclass of GtkToolButton that displays toggle buttons on the toolbar
GtkRadioToolButton	A subclass of GtkToolButton that displays radio buttons on the toolbar
GtkSeparatorToolItem	A subclass of GtkToolItem that separates groups of items on a toolbar

<< GtkSeparatorToolItem **GtkMenuToolButton >>**

GtkMenuToolButton

GtkMenuToolButton — A GtkToolItem containing a button with an additional dropdown menu

Synopsis

```
#include <gtk/gtk.h>


            GtkMenuToolButton;
GtkToolItem* gtk_menu_tool_button_new       (GtkWidget *icon_widget,
                                             const gchar *label);
GtkToolItem* gtk_menu_tool_button_new_from_stock
                                            (const gchar *stock_id);
void        gtk_menu_tool_button_set_menu   (GtkMenuToolButton *button,
                                             GtkWidget *menu);
GtkWidget*  gtk_menu_tool_button_get_menu   (GtkMenuToolButton *button);
void        gtk_menu_tool_button_set_arrow_tooltip
                                            (GtkMenuToolButton *button,
                                             GtkTooltips *tooltips,
                                             const gchar *tip_text,
                                             const gchar *tip_private);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkWidget
               +----GtkContainer
                     +----GtkBin
                           +----GtkToolItem
                                 +----GtkToolButton
                                       +----GtkMenuToolButton
```

Implemented Interfaces

GtkMenuToolButton implements AtkImplementorIface.

Properties

```
  "menu"                GtkMenu               : Read / Write
```

Signal Prototypes

```
"show-menu" void        user_function      (GtkMenuToolButton *menutoolbutton,
                                            gpointer user_data);
```

Description

A GtkMenuToolButton is a GtkToolItem that contains a button and a small additional button with an arrow. When clicked, the arrow button pops up a dropdown menu.

Use `gtk_menu_tool_button_new()` to create a new GtkMenuToolButton. Use `gtk_toggle_tool_button_new_from_stock()` to create a new GtkMenuToolButton containing a stock item.

Details

GtkMenuToolButton

```
typedef struct _GtkMenuToolButton GtkMenuToolButton;
```

The GtkMenuToolButton struct contains only private data and should only be accessed through the functions described below.

---

gtk_menu_tool_button_new ()

```
GtkToolItem* gtk_menu_tool_button_new       (GtkWidget *icon_widget,
                                             const gchar *label);
```

Creates a new GtkMenuToolButton using *icon_widget* as icon and *label* as label.

icon_widget : a widget that will be used as icon widget, or NULL
label : a string that will be used as label, or NULL
Returns : the new GtkMenuToolButton

Since 2.6

---

gtk_menu_tool_button_new_from_stock ()

```
GtkToolItem* gtk_menu_tool_button_new_from_stock
                                            (const gchar *stock_id);
```

Creates a new GtkMenuToolButton. The new GtkMenuToolButton will contain an icon and label from the stock item indicated by *stock_id*.

stock_id : the name of a stock item
Returns : the new GtkMenuToolButton

Since 2.6

---

gtk_menu_tool_button_set_menu ()

```
void        gtk_menu_tool_button_set_menu   (GtkMenuToolButton *button,
                                             GtkWidget *menu);
```

Sets the GtkMenu that is popped up when the user clicks on the arrow. If *menu* is NULL, the arrow button becomes insensitive.

button : a GtkMenuToolButton
menu : the GtkMenu associated with GtkMenuToolButton

Since 2.6

---

gtk_menu_tool_button_get_menu ()

```
GtkWidget*  gtk_menu_tool_button_get_menu   (GtkMenuToolButton *button);
```

Gets the GtkMenu associated with GtkMenuToolButton.

button : a GtkMenuToolButton

Returns : the GtkMenu associated with GtkMenuToolButton

Since 2.6

---

gtk_menu_tool_button_set_arrow_tooltip ()

```
void        gtk_menu_tool_button_set_arrow_tooltip
                                            (GtkMenuToolButton *button,
                                             GtkTooltips *tooltips,
                                             const gchar *tip_text,
                                             const gchar *tip_private);
```

button :
tooltips :
tip_text :
tip_private :

Properties

The "menu" property

```
  "menu"                GtkMenu               : Read / Write
```

The dropdown menu.

Signals

The "show-menu" signal

```
void        user_function                   (GtkMenuToolButton *menutoolbutton,
                                             gpointer user_data);
```

menutoolbutton : the object which received the signal.
user_data : user data set when the signal handler was connected.

See Also

GtkToolbar, GtkToolButton,

The toolbar widget

The parent class of GtkMenuToolButton. The properties "label_widget", "label", "icon_widget", and "stock_id" on GtkToolButton determine the label and icon used on GtkMenuToolButtons.

<< GtkToolButton **GtkToggleToolButton >>**

GtkToggleToolButton

GtkToggleToolButton — A GtkToolItem containing a toggle button

Synopsis

```
#include <gtk/gtk.h>


            GtkToggleToolButton;
GtkToolItem* gtk_toggle_tool_button_new     (void);
GtkToolItem* gtk_toggle_tool_button_new_from_stock
                                            (const gchar *stock_id);
void        gtk_toggle_tool_button_set_active
                                            (GtkToggleToolButton *button,
                                             gboolean is_active);
gboolean    gtk_toggle_tool_button_get_active
                                            (GtkToggleToolButton *button);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkWidget
               +----GtkContainer
                     +----GtkBin
                           +----GtkToolItem
                                 +----GtkToolButton
                                       +----GtkToggleToolButton
                                             +----GtkRadioToolButton
```

Implemented Interfaces

GtkToggleToolButton implements AtkImplementorIface.

Signal Prototypes

```
"toggled"   void        user_function      (GtkToggleToolButton *toggle_tool_button,
                                            gpointer user_data);
```

Description

A GtkToggleToolButton is a GtkToolItem that contains a toggle button.

Use `gtk_toggle_tool_button_new()` to create a new GtkToggleToolButton. Use `gtk_toggle_tool_button_new_from_stock()` to create a new GtkToggleToolButton containing a stock item.

Details

GtkToggleToolButton

```
typedef struct _GtkToggleToolButton GtkToggleToolButton;
```

The GtkToggleToolButton struct contains only private data and should only be accessed through the functions described below.

---

gtk_toggle_tool_button_new ()

```
GtkToolItem* gtk_toggle_tool_button_new     (void);
```

Returns a new GtkToggleToolButton

Returns : a newly created GtkToggleToolButton

Since 2.4

---

gtk_toggle_tool_button_new_from_stock ()

```
GtkToolItem* gtk_toggle_tool_button_new_from_stock
                                            (const gchar *stock_id);
```

Creates a new GtkToggleToolButton containing the image and text from a stock item. Some stock ids have preprocessor macros like GTK_STOCK_OK and GTK_STOCK_APPLY.

It is an error if *stock_id* is not a name of a stock item.

stock_id : the name of the stock item
Returns : A new GtkToggleToolButton

Since 2.4

---

gtk_toggle_tool_button_set_active ()

```
void        gtk_toggle_tool_button_set_active
                                            (GtkToggleToolButton *button,
                                             gboolean is_active);
```

Sets the status of the toggle tool button. Set to TRUE if you want the GtkToggleButton to be 'pressed in', and FALSE to raise it. This action causes the toggled signal to be emitted.

button : a GtkToggleToolButton
is_active : whether *button* should be active

Since 2.4

---

gtk_toggle_tool_button_get_active ()

```
gboolean    gtk_toggle_tool_button_get_active
                                            (GtkToggleToolButton *button);
```

Queries a GtkToggleToolButton and returns its current state. Returns TRUE if the toggle button is pressed in and FALSE if it is raised.

button : a GtkToggleToolButton
Returns : TRUE if the toggle tool button is pressed in, FALSE if not

Since 2.4

Signals

The "toggled" signal

```
void        user_function                   (GtkToggleToolButton *toggle_tool_button,
                                             gpointer user_data);
```

Emitted whenever the toggle tool button changes state.

toggle_tool_button : the object that emitted the signal
user_data : user data set when the signal handler was connected.

See Also

GtkToolbar, GtkToolButton, GtkSeparatorToolItem,

The toolbar widget

The parent class of GtkToggleToolButton. The properties "label_widget", "label", "icon_widget", and "stock_id" on GtkToolButton determine the label and icon used on GtkToggleToolButtons.

A subclass of GtkToolItem that separates groups of items on a toolbar.

<< GtkMenuToolButton **GtkRadioToolButton >>**

GtkRadioToolButton

GtkRadioToolButton — A toolbar item that contains a radio button

Synopsis

```
#include <gtk/gtk.h>


            GtkRadioToolButton;
GtkToolItem* gtk_radio_tool_button_new      (GSList *group);
GtkToolItem* gtk_radio_tool_button_new_from_stock
                                            (GSList *group,
                                             const gchar *stock_id);
GtkToolItem* gtk_radio_tool_button_new_from_widget
                                            (GtkRadioToolButton *group);
GtkToolItem* gtk_radio_tool_button_new_with_stock_from_widget
                                            (GtkRadioToolButton *group,
                                             const gchar *stock_id);
GSList*     gtk_radio_tool_button_get_group (GtkRadioToolButton *button);
void        gtk_radio_tool_button_set_group (GtkRadioToolButton *button,
                                             GSList *group);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkWidget
               +----GtkContainer
                     +----GtkBin
                           +----GtkToolItem
```

```
                                    +----GtkToolButton
                                          +----GtkToggleToolButton
                                                +----GtkRadioToolButton
```

Implemented Interfaces

GtkRadioToolButton implements AtkImplementorIface.

Properties

```
  "group"                GtkRadioToolButton    : Write
```

Description

A GtkRadioToolButton is a GtkToolItem that contains a radio button, that is, a button that is part of a group of toggle buttons where only one button can be active at a time.

Use `gtk_radio_tool_button_new()` to create a new GtkRadioToolButton. use `gtk_radio_tool_button_new_from_widget()` to create a new GtkRadioToolButton that is part of the same group as an existing GtkRadioToolButton. Use `gtk_radio_tool_button_new_from_stock()` or `gtk_radio_tool_button_new_from_widget_with_stock()` to create a new GtkRAdioToolButton containing a stock item.

Details

GtkRadioToolButton

```
typedef struct _GtkRadioToolButton GtkRadioToolButton;
```

The GtkRadioToolButton contains only private data and should only be accessed through the functions described below.

---

gtk_radio_tool_button_new ()

```
GtkToolItem* gtk_radio_tool_button_new      (GSList *group);
```

Creates a new GtkRadioToolButton, adding it to *group*.

group : An existing radio button group, or NULL if you are creating a new group
Returns : The new GtkRadioToolButton

Since 2.4

---

gtk_radio_tool_button_new_from_stock ()

```
GtkToolItem* gtk_radio_tool_button_new_from_stock
                                            (GSList *group,
                                             const gchar *stock_id);
```

Creates a new GtkRadioToolButton, adding it to *group*. The new GtkRadioToolButton will contain an icon and label from the stock item indicated by *stock_id*.

group : an existing radio button group, or NULL if you are creating a new group
stock_id : the name of a stock item
Returns : The new GtkRadioToolItem

Since 2.4

---

gtk_radio_tool_button_new_from_widget ()

```
GtkToolItem* gtk_radio_tool_button_new_from_widget
                                            (GtkRadioToolButton *group);
```

Creates a new GtkRadioToolButton adding it to the same group as *gruup*

group : An existing GtkRadioToolButton
Returns : The new GtkRadioToolButton

Since 2.4

---

gtk_radio_tool_button_new_with_stock_from_widget ()

```
GtkToolItem* gtk_radio_tool_button_new_with_stock_from_widget
                                                (GtkRadioToolButton *group,
                                                 const gchar *stock_id);
```

Creates a new GtkRadioToolButton adding it to the same group as *group*. The new GtkRadioToolButton will contain an icon and label from the stock item indicated by *stock_id*.

group : An existing GtkRadioToolButton.
stock_id : the name of a stock item
Returns : A new GtkRadioToolButton

Since 2.4

---

gtk_radio_tool_button_get_group ()

```
GSList*     gtk_radio_tool_button_get_group (GtkRadioToolButton *button);
```

Returns the radio button group *button* belongs to.

button : a GtkRadioToolButton
Returns : The group *button* belongs to.

Since 2.4

---

gtk_radio_tool_button_set_group ()

```
void        gtk_radio_tool_button_set_group (GtkRadioToolButton *button,
                                             GSList *group);
```

Adds *button* to *group*, removing it from the group it belonged to before.

button : a GtkRadioToolButton
group : an existing radio button group

Since 2.4

Properties

The "group" property

```
  "group"                GtkRadioToolButton    : Write
```

Sets a new group for a radio tool button.

Since 2.4

See Also

GtkToolbar	The toolbar widget
GtkToolButton	An ancestor class of GtkRadioToolButton. The properties "label_widget", "label", "icon_widget", and "stock_id" on GtkToolButton determine the label and icon used on a GtkRadioToolButton.
GtkSeparatorToolItem	A subclass of GtkToolItem that separates groups of items on a toolbar. It is usually a good idea to put a separator before and after a group of GtkRadioToolButtons on a GtkToolbar.

<< GtkToggleToolButton **Action-based menus and toolbars >>**

Action-based menus and toolbars

GtkUIManager - Constructing menus and toolbars from an XML description
GtkActionGroup - A group of actions
GtkAction - An action which can be triggered by a menu or toolbar item
GtkToggleAction - An action which can be toggled between two states
GtkRadioAction - An action of which only one in a group can be active

<< GtkRadioToolButton **GtkUIManager >>**

GtkUIManager

GtkUIManager — Constructing menus and toolbars from an XML description

Synopsis

```
#include <gtk/gtk.h>

            GtkUIManager;
GtkUIManager* gtk_ui_manager_new            (void);
void        gtk_ui_manager_set_add_tearoffs (GtkUIManager *self,
                                             gboolean add_tearoffs);
gboolean    gtk_ui_manager_get_add_tearoffs (GtkUIManager *self);
void        gtk_ui_manager_insert_action_group
                                            (GtkUIManager *self,
                                             GtkActionGroup *action_group,
                                             gint pos);
void        gtk_ui_manager_remove_action_group
                                            (GtkUIManager *self,
                                             GtkActionGroup *action_group);
GList*      gtk_ui_manager_get_action_groups
                                            (GtkUIManager *self);
GtkAccelGroup* gtk_ui_manager_get_accel_group
                                            (GtkUIManager *self);
GtkWidget*  gtk_ui_manager_get_widget       (GtkUIManager *self,
                                             const gchar *path);
GSList*     gtk_ui_manager_get_toplevels    (GtkUIManager *self,
                                             GtkUIManagerItemType types);
GtkAction*  gtk_ui_manager_get_action       (GtkUIManager *self,
                                             const gchar *path);
guint       gtk_ui_manager_add_ui_from_string
                                            (GtkUIManager *self,
                                             const gchar *buffer,
                                             gssize length,
                                             GError **error);
```

```
guint       gtk_ui_manager_add_ui_from_file (GtkUIManager *self,
                                             const gchar *filename,
                                             GError **error);
guint       gtk_ui_manager_new_merge_id     (GtkUIManager *self);
enum        GtkUIManagerItemType;
void        gtk_ui_manager_add_ui           (GtkUIManager *self,
                                             guint merge_id,
                                             const gchar *path,
                                             const gchar *name,
                                             const gchar *action,
                                             GtkUIManagerItemType type,
                                             gboolean top);
void        gtk_ui_manager_remove_ui        (GtkUIManager *self,
                                             guint merge_id);
gchar*      gtk_ui_manager_get_ui           (GtkUIManager *self);
void        gtk_ui_manager_ensure_update    (GtkUIManager *self);
```

Object Hierarchy

```
  GObject
   +----GtkUIManager
```

Properties

```
  "add-tearoffs"         gboolean             : Read / Write
  "ui"                   gchararray           : Read
```

Signal Prototypes

```
"actions-changed"
            void        user_function      (GtkUIManager *merge,
                                            gpointer user_data);
"add-widget"
```

```
            void        user_function      (GtkUIManager *merge,
                                            GtkWidget *widget,
                                            gpointer user_data);
"connect-proxy"
            void        user_function      (GtkUIManager *uimanager,
                                            GtkAction *action,
                                            GtkWidget *proxy,
                                            gpointer user_data);
"disconnect-proxy"
            void        user_function      (GtkUIManager *uimanager,
                                            GtkAction *action,
                                            GtkWidget *proxy,
                                            gpointer user_data);
"post-activate"
            void        user_function      (GtkUIManager *uimanager,
                                            GtkAction *action,
                                            gpointer user_data);
"pre-activate"
            void        user_function      (GtkUIManager *uimanager,
                                            GtkAction *action,
                                            gpointer user_data);
```

Description

A GtkUIManager constructs a user interface (menus and toolbars) from one or more UI definitions, which reference actions from one or more action groups.

UI Definitions

The UI definitions are specified in an XML format which can be roughly described by the following DTD.

```
<!ELEMENT ui          (menubar|toolbar|popup|accelerator)* >
<!ELEMENT menubar     (menuitem|separator|placeholder|menu)* >
<!ELEMENT menu        (menuitem|separator|placeholder|menu)* >
<!ELEMENT popup       (menuitem|separator|placeholder|menu)* >
<!ELEMENT toolbar     (toolitem|separator|placeholder)* >
<!ELEMENT placeholder (menuitem|toolitem|separator|placeholder|menu)* >
<!ELEMENT menuitem     EMPTY >
<!ELEMENT toolitem     EMPTY >
<!ELEMENT separator    EMPTY >
<!ELEMENT accelerator  EMPTY >
```

```
<!ATTLIST menubar      name               #IMPLIED
                       action             #IMPLIED >
<!ATTLIST toolbar      name               #IMPLIED
                       action             #IMPLIED >
<!ATTLIST popup        name               #IMPLIED
                       action             #IMPLIED >
<!ATTLIST placeholder  name               #IMPLIED
                       action             #IMPLIED >
<!ATTLIST separator    name               #IMPLIED
                       action             #IMPLIED >
<!ATTLIST menu         name               #IMPLIED
                       action             #REQUIRED
                       position (top|bot) #IMPLIED >
<!ATTLIST menuitem     name               #IMPLIED
                       action             #REQUIRED
                       position (top|bot) #IMPLIED >
<!ATTLIST toolitem     name               #IMPLIED
                       action             #REQUIRED
                       position (top|bot) #IMPLIED >
<!ATTLIST accelerator  name               #IMPLIED
                       action             #REQUIRED >
```

There are some additional restrictions beyond those specified in the DTD, e.g. every toolitem must have a toolbar in its anchestry and every menuitem must have a menubar or popup in its anchestry. Since a GMarkup parser is used to parse the UI description, it must not only be valid XML, but valid GMarkup.

If a name is not specified, it defaults to the action. If an action is not specified either, the element name is used. The name and action attributes must not contain '/' characters after parsing (since that would mess up path lookup) and must be usable as XML attributes when enclosed in doublequotes, thus they must not "" characters or references to the " entity.

Example 1. A UI definition

```
<ui>
  <menubar>
    <menu name="FileMenu" action="FileMenuAction">
      <menuitem name="New" action="New2Action" />
      <placeholder name="FileMenuAdditions" />
    </menu>
    <menu name="JustifyMenu" action="JustifyMenuAction">
      <menuitem name="Left" action="justify-left"/>
      <menuitem name="Centre" action="justify-center"/>
      <menuitem name="Right" action="justify-right"/>
      <menuitem name="Fill" action="justify-fill"/>
    </menu>
```

```
  </menubar>
  <toolbar action="toolbar1">
    <placeholder name="JustifyToolItems">
      <separator/>
      <toolitem name="Left" action="justify-left"/>
      <toolitem name="Centre" action="justify-center"/>
      <toolitem name="Right" action="justify-right"/>
      <toolitem name="Fill" action="justify-fill"/>
      <separator/>
    </placeholder>
  </toolbar>
</ui>
```

The constructed widget hierarchy is very similar to the element tree of the XML, with the exception that placeholders are merged into their parents. The correspondence of XML elements to widgets should be almost obvious:

menubar	a GtkMenuBar
toolbar	a GtkToolbar
popup	a toplevel GtkMenu
menu	a GtkMenu attached to a menuitem
menuitem	a GtkMenuItem subclass, the exact type depends on the action
toolitem	a GtkToolItem subclass, the exact type depends on the action
separator	a GtkSeparatorMenuItem or GtkSeparatorToolItem
accelerator	a keyboard accelerator

The "position" attribute determines where a constructed widget is positioned wrt. to its siblings in the partially constructed tree. If it is "top", the widget is prepended, otherwise it is appended.

UI Merging

The most remarkable feature of GtkUIManager is that it can overlay a set of menuitems and toolitems over another one, and demerge them later.

Merging is done based on the names of the XML elements. Each element is identified by a path which consists of the names of its anchestors, separated by slashes. For example, the menuitem named "Left" in the example above has the path `/ui/menubar/JustifyMenu/Left` and the toolitem with the same name has path `/ui/toolbar1/JustifyToolItems/Left`.

Accelerators

Every action has an accelerator path. Accelerators are installed together with menuitem proxies, but they can also be explicitly added with <accelerator> elements in the UI definition. This makes it possible to have accelerators for actions even if they have no visible proxies.

Smart Separators

The separators created by GtkUIManager are "smart", i.e. they do not show up in the UI unless they end up between two visible menu or tool items. Separators which are located at the very beginning or end of the menu or toolbar containing them, or multiple separators next to each other, are hidden. This is a useful feature, since the merging of UI elements from multiple sources can make it hard or impossible to determine in advance whether a separator will end up in such an unfortunate position.

Empty Menus

Submenus pose similar problems to separators inconnection with merging. It is impossible to know in advance whether they will end up empty after merging. GtkUIManager offers two ways to treat empty submenus:

- make them disappear by hiding the menu item they're attached to
- add an insensitive "Empty" item

The behaviour is chosen based on the "is_important" property of the action to which the submenu is associated.

Details

GtkUIManager

```
typedef struct _GtkUIManager GtkUIManager;
```

The GtkUIManager struct contains only private members and should not be accessed directly.

---

gtk_ui_manager_new ()

```
GtkUIManager* gtk_ui_manager_new            (void);
```

Creates a new ui manager object.

Returns : a new ui manager object.

Since 2.4

---

gtk_ui_manager_set_add_tearoffs ()

```
void        gtk_ui_manager_set_add_tearoffs (GtkUIManager *self,
                                             gboolean add_tearoffs);
```

Sets the "add_tearoffs" property, which controls whether menus generated by this GtkUIManager will have tearoff menu items.

Note that this only affects regular menus. Generated popup menus never have tearoff menu items.

self : a GtkUIManager
add_tearoffs : whether tearoff menu items are added

Since 2.4

---

gtk_ui_manager_get_add_tearoffs ()

```
gboolean    gtk_ui_manager_get_add_tearoffs (GtkUIManager *self);
```

Returns whether menus generated by this GtkUIManager will have tearoff menu items.

self : a GtkUIManager
Returns : whether tearoff menu items are added

Since 2.4

---

gtk_ui_manager_insert_action_group ()

```
void        gtk_ui_manager_insert_action_group
                                            (GtkUIManager *self,
                                             GtkActionGroup *action_group,
                                             gint pos);
```

Inserts an action group into the list of action groups associated with *self*. Actions in earlier groups hide actions with the same name in later groups.

self : a GtkUIManager object
action_group : the action group to be inserted
pos : the position at which the group will be inserted.

Since 2.4

---

gtk_ui_manager_remove_action_group ()

```
void        gtk_ui_manager_remove_action_group
                                            (GtkUIManager *self,
                                             GtkActionGroup *action_group);
```

Removes an action group from the list of action groups associated with *self*.

self : a GtkUIManager object
action_group : the action group to be removed

Since 2.4

---

gtk_ui_manager_get_action_groups ()

```
GList*      gtk_ui_manager_get_action_groups
                                            (GtkUIManager *self);
```

Returns the list of action groups associated with *self*.

self : a GtkUIManager object

Returns : a GList of action groups. The list is owned by GTK+ and should not be modified.

Since 2.4

---

gtk_ui_manager_get_accel_group ()

```
GtkAccelGroup* gtk_ui_manager_get_accel_group
                                            (GtkUIManager *self);
```

Returns the GtkAccelGroup associated with *self*.

self : a GtkUIManager object

Returns : the GtkAccelGroup.

Since 2.4

---

gtk_ui_manager_get_widget ()

```
GtkWidget*  gtk_ui_manager_get_widget       (GtkUIManager *self,
                                             const gchar *path);
```

Looks up a widget by following a path. The path consists of the names specified in the XML description of the UI. separated by '/'. Elements which don't have a name or action attribute in the XML (e.g. <popup>) can be addressed by their XML element name (e.g. "popup"). The root element ("/ui") can be omitted in the path.

Note that the widget found by following a path that ends in a <menu> element is the menuitem to which the menu

is attached, not the menu itself.

self : a GtkUIManager
path : a path
Returns : the widget found by following the path, or NULL if no widget was found.

Since 2.4

---

gtk_ui_manager_get_toplevels ()

```
GSList*     gtk_ui_manager_get_toplevels    (GtkUIManager *self,
                                             GtkUIManagerItemType types);
```

Obtains a list of all toplevel widgets of the requested types.

self : a GtkUIManager
types : specifies the types of toplevel widgets to include. Allowed types are GTK_UI_MANAGER_MENUBAR, GTK_UI_MANAGER_TOOLBAR and GTK_UI_MANAGER_POPUP.
Returns : a newly-allocated of all toplevel widgets of the requested types.

Since 2.4

---

gtk_ui_manager_get_action ()

```
GtkAction*  gtk_ui_manager_get_action       (GtkUIManager *self,
                                             const gchar *path);
```

Looks up an action by following a path. See gtk_ui_manager_get_widget() for more information about paths.

self : a GtkUIManager
path : a path

Returns : the action whose proxy widget is found by following the path, or `NULL` if no widget was found.

Since 2.4

---

gtk_ui_manager_add_ui_from_string ()

```
guint       gtk_ui_manager_add_ui_from_string
                                            (GtkUIManager *self,
                                             const gchar *buffer,
                                             gssize length,
                                             GError **error);
```

Parses a string containing a UI definition and merges it with the current contents of *self*. An enclosing <ui> element is added if it is missing.

self : a GtkUIManager object

buffer : the string to parse

length : the length of *buffer* (may be -1 if *buffer* is nul-terminated)

error : return location for an error

Returns : The merge id for the merged UI. The merge id can be used to unmerge the UI with `gtk_ui_manager_remove_ui()`. If an error occurred, the return value is 0.

Since 2.4

---

gtk_ui_manager_add_ui_from_file ()

```
guint       gtk_ui_manager_add_ui_from_file (GtkUIManager *self,
                                             const gchar *filename,
                                             GError **error);
```

Parses a file containing a UI definition and merges it with the current contents of *self*.

self : a GtkUIManager object

filename : the name of the file to parse

error : return location for an error

Returns : The merge id for the merged UI. The merge id can be used to unmerge the UI with `gtk_ui_manager_remove_ui()`. If an error occurred, the return value is 0.

Since 2.4

---

gtk_ui_manager_new_merge_id ()

```
guint       gtk_ui_manager_new_merge_id     (GtkUIManager *self);
```

Returns an unused merge id, suitable for use with `gtk_ui_manager_add_ui()`.

self : a GtkUIManager

Returns : an unused merge id.

Since 2.4

---

enum GtkUIManagerItemType

```
typedef enum {
  GTK_UI_MANAGER_AUTO         = 0,
  GTK_UI_MANAGER_MENUBAR      = 1 << 0,
  GTK_UI_MANAGER_MENU         = 1 << 1,
  GTK_UI_MANAGER_TOOLBAR      = 1 << 2,
  GTK_UI_MANAGER_PLACEHOLDER  = 1 << 3,
  GTK_UI_MANAGER_POPUP        = 1 << 4,
  GTK_UI_MANAGER_MENUITEM     = 1 << 5,
  GTK_UI_MANAGER_TOOLITEM     = 1 << 6,
  GTK_UI_MANAGER_SEPARATOR    = 1 << 7,
  GTK_UI_MANAGER_ACCELERATOR  = 1 << 8
} GtkUIManagerItemType;
```

These enumeration values are used by `gtk_ui_manager_add_ui()` to determine what UI element to create.

GTK_UI_MANAGER_AUTO	Pick the type of the UI element according to context.
GTK_UI_MANAGER_MENUBAR	Create a menubar.
GTK_UI_MANAGER_MENU	Create a menu.
GTK_UI_MANAGER_TOOLBAR	Create a toolbar.
GTK_UI_MANAGER_PLACEHOLDER	Insert a placeholder.
GTK_UI_MANAGER_POPUP	Create a popup menu.
GTK_UI_MANAGER_MENUITEM	Create a menuitem.
GTK_UI_MANAGER_TOOLITEM	Create a toolitem.
GTK_UI_MANAGER_SEPARATOR	Create a separator.
GTK_UI_MANAGER_ACCELERATOR	Install an accelerator.

---

gtk_ui_manager_add_ui ()

```
void        gtk_ui_manager_add_ui           (GtkUIManager *self,
                                             guint merge_id,
                                             const gchar *path,
                                             const gchar *name,
                                             const gchar *action,
                                             GtkUIManagerItemType type,
                                             gboolean top);
```

Adds a UI element to the current contents of *self*.

If *type* is GTK_UI_MANAGER_AUTO, GTK+ inserts a menuitem, toolitem or separator if such an element can be inserted at the place determined by *path*. Otherwise *type* must indicate an element that can be inserted at the place determined by *path*.

self :	a GtkUIManager
merge_id :	the merge id for the merged UI, see gtk_ui_manager_new_merge_id()
path :	a path
name :	the name for the added UI element
action :	the name of the action to be proxied, or NULL to add a separator
type :	the type of UI element to add.
top :	if TRUE, the UI element is added before its siblings, otherwise it is added after its siblings.

Since 2.4

---

gtk_ui_manager_remove_ui ()

```
void        gtk_ui_manager_remove_ui        (GtkUIManager *self,
                                             guint merge_id);
```

Unmerges the part of *self*s content identified by *merge_id*.

self : a GtkUIManager object

merge_id : a merge id as returned by gtk_ui_manager_add_ui_from_string()

Since 2.4

---

gtk_ui_manager_get_ui ()

```
gchar*      gtk_ui_manager_get_ui           (GtkUIManager *self);
```

Creates a UI definition of the merged UI.

self : a GtkUIManager

Returns : A newly allocated string containing an XML representation of the merged UI.

Since 2.4

---

gtk_ui_manager_ensure_update ()

```
void        gtk_ui_manager_ensure_update    (GtkUIManager *self);
```

Makes sure that all pending updates to the UI have been completed.

This may occasionally be necessary, since GtkUIManager updates the UI in an idle function. A typical example where this function is useful is to enforce that the menubar and toolbar have been added to the main window before showing it:

```
gtk_container_add (GTK_CONTAINER (window), vbox);
g_signal_connect (merge, "add_widget",
                  G_CALLBACK (add_widget), vbox);
gtk_ui_manager_add_ui_from_file (merge, "my-menus");
gtk_ui_manager_add_ui_from_file (merge, "my-toolbars");
gtk_ui_manager_ensure_update (merge);
gtk_widget_show (window);
```

self : a GtkUIManager

Since 2.4

Properties

The "add-tearoffs" property

```
  "add-tearoffs"         gboolean              : Read / Write
```

The "add-tearoffs" property controls whether generated menus have tearoff menu items.

Note that this only affects regular menus. Generated popup menus never have tearoff menu items.

Default value: FALSE

Since 2.4

The "ui" property

```
  "ui"                   gchararray            : Read
```

An XML string describing the merged UI.

Default value: NULL

Signals

The "actions-changed" signal

```
void        user_function                   (GtkUIManager *merge,
                                             gpointer user_data);
```

The "actions-changed" signal is emitted whenever the set of actions changes.

merge : a GtkUIManager
user_data : user data set when the signal handler was connected.

Since 2.4

The "add-widget" signal

```
void        user_function                   (GtkUIManager *merge,
                                             GtkWidget *widget,
                                             gpointer user_data);
```

The add_widget signal is emitted for each generated menubar and toolbar. It is not emitted for generated popup menus, which can be obtained by `gtk_ui_manager_get_widget()`.

merge : a GtkUIManager
widget : the added widget
user_data : user data set when the signal handler was connected.

Since 2.4

The "connect-proxy" signal

```
void        user_function                   (GtkUIManager *uimanager,
                                             GtkAction *action,
                                             GtkWidget *proxy,
                                             gpointer user_data);
```

The connect_proxy signal is emitted after connecting a proxy to an action in the group.

This is intended for simple customizations for which a custom action class would be too clumsy, e.g. showing tooltips for menuitems in the statusbar.

uimanager : the ui manager
action : the action
proxy : the proxy
user_data : user data set when the signal handler was connected.

Since 2.4

The "disconnect-proxy" signal

```
void        user_function                   (GtkUIManager *uimanager,
                                             GtkAction *action,
                                             GtkWidget *proxy,
                                             gpointer user_data);
```

The disconnect_proxy signal is emitted after disconnecting a proxy from an action in the group.

uimanager : the ui manager
action : the action
proxy : the proxy
user_data : user data set when the signal handler was connected.

Since 2.4

The "post-activate" signal

```
void        user_function                   (GtkUIManager *uimanager,
                                             GtkAction *action,
                                             gpointer user_data);
```

The post_activate signal is emitted just after the *action* is activated.

This is intended for applications to get notification just after any action is activated.

uimanager : the ui manager
action : the action
user_data : user data set when the signal handler was connected.

Since 2.4

The "pre-activate" signal

```
void        user_function                   (GtkUIManager *uimanager,
                                             GtkAction *action,
                                             gpointer user_data);
```

The pre_activate signal is emitted just before the *action* is activated.

This is intended for applications to get notification just before any action is activated.

uimanager : the ui manager
action : the action
user_data : user data set when the signal handler was connected.

Since 2.4

<< Action-based menus and toolbars **GtkActionGroup >>**

GtkActionGroup

GtkActionGroup — A group of actions

Synopsis

```
#include <gtk/gtk.h>

            GtkActionGroup;
GtkActionGroup* gtk_action_group_new        (const gchar *name);
const gchar* gtk_action_group_get_name      (GtkActionGroup *action_group);
gboolean    gtk_action_group_get_sensitive  (GtkActionGroup *action_group);
void        gtk_action_group_set_sensitive  (GtkActionGroup *action_group,
                                             gboolean sensitive);
gboolean    gtk_action_group_get_visible    (GtkActionGroup *action_group);
void        gtk_action_group_set_visible    (GtkActionGroup *action_group,
                                             gboolean visible);
GtkAction*  gtk_action_group_get_action     (GtkActionGroup *action_group,
                                             const gchar *action_name);
GList*      gtk_action_group_list_actions   (GtkActionGroup *action_group);
void        gtk_action_group_add_action     (GtkActionGroup *action_group,
                                             GtkAction *action);
void        gtk_action_group_add_action_with_accel
                                            (GtkActionGroup *action_group,
                                             GtkAction *action,
                                             const gchar *accelerator);
void        gtk_action_group_remove_action  (GtkActionGroup *action_group,
                                             GtkAction *action);
            GtkActionEntry;
void        gtk_action_group_add_actions    (GtkActionGroup *action_group,
                                             const GtkActionEntry *entries,
                                             guint n_entries,
                                             gpointer user_data);
void        gtk_action_group_add_actions_full
                                            (GtkActionGroup *action_group,
                                             const GtkActionEntry *entries,
                                             guint n_entries,
                                             gpointer user_data,
```

```
                                                GDestroyNotify destroy);
            GtkToggleActionEntry;
void        gtk_action_group_add_toggle_actions
                                               (GtkActionGroup *action_group,
                                                const GtkToggleActionEntry *entries,
                                                guint n_entries,
                                                gpointer user_data);
void        gtk_action_group_add_toggle_actions_full
                                               (GtkActionGroup *action_group,
                                                const GtkToggleActionEntry *entries,
                                                guint n_entries,
                                                gpointer user_data,
                                                GDestroyNotify destroy);
            GtkRadioActionEntry;
void        gtk_action_group_add_radio_actions
                                               (GtkActionGroup *action_group,
                                                const GtkRadioActionEntry *entries,
                                                guint n_entries,
                                                gint value,
                                                GCallback on_change,
                                                gpointer user_data);
void        gtk_action_group_add_radio_actions_full
                                               (GtkActionGroup *action_group,
                                                const GtkRadioActionEntry *entries,
                                                guint n_entries,
                                                gint value,
                                                GCallback on_change,
                                                gpointer user_data,
                                                GDestroyNotify destroy);
void        gtk_action_group_set_translate_func
                                               (GtkActionGroup *action_group,
                                                GtkTranslateFunc func,
                                                gpointer data,
                                                GtkDestroyNotify notify);
void        gtk_action_group_set_translation_domain
                                               (GtkActionGroup *action_group,
                                                const gchar *domain);
G_CONST_RETURN gchar* gtk_action_group_translate_string
                                               (GtkActionGroup *action_group,
                                                const gchar *string);
```

Object Hierarchy

```
GObject
 +----GtkActionGroup
```

Properties

```
"name"                gchararray            : Read / Write / Construct Only
"sensitive"           gboolean              : Read / Write
"visible"             gboolean              : Read / Write
```

Signal Prototypes

```
"connect-proxy"
            void        user_function      (GtkActionGroup *action_group,
                                            GtkAction *action,
                                            GtkWidget *proxy,
                                            gpointer user_data);
"disconnect-proxy"
            void        user_function      (GtkActionGroup *action_group,
                                            GtkAction *action,
                                            GtkWidget *proxy,
                                            gpointer user_data);
"post-activate"
            void        user_function      (GtkActionGroup *action_group,
                                            GtkAction *action,
                                            gpointer user_data);
"pre-activate"
            void        user_function      (GtkActionGroup *action_group,
                                            GtkAction *action,
                                            gpointer user_data);
```

Description

Actions are organised into groups. An action group is essentially a map from names to GtkAction objects.

All actions that would make sense to use in a particular context should be in a single group. Multiple action groups may be used for a particular user interface. In fact, it is expected that most nontrivial applications will make use of multiple groups. For example, in an application that can edit multiple documents, one group holding global actions (e.g. quit, about, new), and one group per document holding actions that act on that document (eg. save, cut/copy/paste, etc). Each window's

menus would be constructed from a combination of two action groups.

Accelerators are handled by the GTK+ accelerator map. All actions are assigned an accelerator path (which normally has the form `<Actions>/group-name/action-name`) and a shortcut is associated with this accelerator path. All menuitems and toolitems take on this accelerator path. The GTK+ accelerator map code makes sure that the correct shortcut is displayed next to the menu item.

Details

GtkActionGroup

```
typedef struct _GtkActionGroup GtkActionGroup;
```

The GtkActionGroup struct contains only private members and should not be accessed directly.

---

gtk_action_group_new ()

```
GtkActionGroup* gtk_action_group_new        (const gchar *name);
```

Creates a new GtkActionGroup object. The name of the action group is used when associating keybindings with the actions.

name : the name of the action group.
Returns : the new GtkActionGroup

Since 2.4

---

gtk_action_group_get_name ()

```
const gchar* gtk_action_group_get_name      (GtkActionGroup *action_group);
```

Gets the name of the action group.

action_group : the action group
Returns : the name of the action group.

Since 2.4

---

gtk_action_group_get_sensitive ()

```
gboolean    gtk_action_group_get_sensitive  (GtkActionGroup *action_group);
```

Returns TRUE if the group is sensitive. The constituent actions can only be logically sensitive (see `gtk_action_is_sensitive()`) if they are sensitive (see `gtk_action_get_sensitive()`) and their group is sensitive.

`action_group` : the action group
Returns : TRUE if the group is sensitive.

Since 2.4

---

gtk_action_group_set_sensitive ()

```
void        gtk_action_group_set_sensitive  (GtkActionGroup *action_group,
                                             gboolean sensitive);
```

Changes the sensitivity of *`action_group`*

`action_group` : the action group
`sensitive` : new sensitivity

Since 2.4

---

gtk_action_group_get_visible ()

```
gboolean    gtk_action_group_get_visible    (GtkActionGroup *action_group);
```

Returns TRUE if the group is visible. The constituent actions can only be logically visible (see `gtk_action_is_visible()`) if they are visible (see `gtk_action_get_visible()`) and their group is visible.

action_group : the action group
Returns : TRUE if the group is sensitive.

Since 2.4

---

gtk_action_group_set_visible ()

```
void        gtk_action_group_set_visible    (GtkActionGroup *action_group,
                                             gboolean visible);
```

Changes the visible of *action_group*.

action_group : the action group
visible : new visiblity

Since 2.4

---

gtk_action_group_get_action ()

```
GtkAction*  gtk_action_group_get_action     (GtkActionGroup *action_group,
                                             const gchar *action_name);
```

Looks up an action in the action group by name.

action_group : the action group
action_name : the name of the action
Returns : the action, or NULL if no action by that name exists

Since 2.4

---

gtk_action_group_list_actions ()

```
GList*      gtk_action_group_list_actions   (GtkActionGroup *action_group);
```

Lists the actions in the action group.

action_group : the action group
Returns : an allocated list of the action objects in the action group

Since 2.4

---

gtk_action_group_add_action ()

```
void        gtk_action_group_add_action     (GtkActionGroup *action_group,
                                             GtkAction *action);
```

Adds an action object to the action group. Note that this function does not set up the accel path of the action, which can lead to problems if a user tries to modify the accelerator of a menuitem associated with the action. Therefore you must either set the accel path yourself with `gtk_action_set_accel_path()`, or use `gtk_action_group_add_action_with_accel (..., NULL)`.

action_group : the action group
action : an action

Since 2.4

---

gtk_action_group_add_action_with_accel ()

```
void        gtk_action_group_add_action_with_accel
                                            (GtkActionGroup *action_group,
                                             GtkAction *action,
                                             const gchar *accelerator);
```

Adds an action object to the action group and sets up the accelerator.

If *accelerator* is `NULL`, attempts to use the accelerator associated with the stock_id of the action.

Accel paths are set to `<Actions>/`*group-name*`/`*action-name*.

action_group : the action group
action : the action to add

accelerator :	the accelerator for the action, in the format understood by gtk_accelerator_parse(), or "" for no accelerator, or NULL to use the stock accelerator

Since 2.4

---

gtk_action_group_remove_action ()

```
void        gtk_action_group_remove_action  (GtkActionGroup *action_group,
                                             GtkAction *action);
```

Removes an action object from the action group.

action_group :	the action group
action :	an action

Since 2.4

GtkActionEntry

```
typedef struct {
  const gchar     *name;
  const gchar     *stock_id;
  const gchar     *label;
  const gchar     *accelerator;
  const gchar     *tooltip;
  GCallback  callback;
} GtkActionEntry;
```

GtkActionEntry structs are used with gtk_action_group_add_actions() to construct actions.

const gchar *name;	The name of the action.
const gchar *stock_id;	The stock id for the action.
const gchar *label;	The label for the action. This field should typically be marked for translation, see gtk_action_group_set_translation_domain().
const gchar *accelerator;	The accelerator for the action, in the format understood by gtk_accelerator_parse().

const gchar *tooltip;	The tooltip for the action. This field should typically be marked for translation, see gtk_action_group_set_translation_domain().
GCallback callback;	The function to call when the action is activated.

---

gtk_action_group_add_actions ()

```
void        gtk_action_group_add_actions    (GtkActionGroup *action_group,
                                             const GtkActionEntry *entries,
                                             guint n_entries,
                                             gpointer user_data);
```

This is a convenience function to create a number of actions and add them to the action group.

The "activate" signals of the actions are connected to the callbacks and their accel paths are set to <Actions>/*group-name/action-name*.

action_group :	the action group
entries :	an array of action descriptions
n_entries :	the number of entries
user_data :	data to pass to the action callbacks

Since 2.4

---

gtk_action_group_add_actions_full ()

```
void        gtk_action_group_add_actions_full
                                            (GtkActionGroup *action_group,
                                             const GtkActionEntry *entries,
                                             guint n_entries,
                                             gpointer user_data,
                                             GDestroyNotify destroy);
```

This variant of gtk_action_group_add_actions() adds a GDestroyNotify callback for *user_data*.

action_group :	the action group
entries :	an array of action descriptions
n_entries :	the number of entries
user_data :	data to pass to the action callbacks

destroy : destroy notification callback for *user_data*

Since 2.4

GtkToggleActionEntry

```
typedef struct {
  const gchar     *name;
  const gchar     *stock_id;
  const gchar     *label;
  const gchar     *accelerator;
  const gchar     *tooltip;
  GCallback  callback;
  gboolean   is_active;
} GtkToggleActionEntry;
```

GtkToggleActionEntry structs are used with `gtk_action_group_add_toggle_actions()` to construct toggle actions.

const gchar **name*;	The name of the action.
const gchar **stock_id*;	The stock id for the action.
const gchar **label*;	The label for the action. This field should typically be marked for translation, see `gtk_action_group_set_translation_domain()`.
const gchar **accelerator*;	The accelerator for the action, in the format understood by `gtk_accelerator_parse()`.
const gchar **tooltip*;	The tooltip for the action. This field should typically be marked for translation, see `gtk_action_group_set_translation_domain()`.
GCallback *callback*;	The function to call when the action is activated.
gboolean *is_active*;	The initial state of the toggle action.

---

gtk_action_group_add_toggle_actions ()

```
void        gtk_action_group_add_toggle_actions
                                            (GtkActionGroup *action_group,
                                             const GtkToggleActionEntry *entries,
                                             guint n_entries,
                                             gpointer user_data);
```

This is a convenience function to create a number of toggle actions and add them to the action group.

The "activate" signals of the actions are connected to the callbacks and their accel paths are set to `<Actions>/group-name/action-name`.

action_group : the action group
entries : an array of toggle action descriptions
n_entries : the number of entries
user_data : data to pass to the action callbacks

Since 2.4

---

gtk_action_group_add_toggle_actions_full ()

```
void        gtk_action_group_add_toggle_actions_full
                                            (GtkActionGroup *action_group,
                                             const GtkToggleActionEntry *entries,
                                             guint n_entries,
                                             gpointer user_data,
                                             GDestroyNotify destroy);
```

This variant of `gtk_action_group_add_toggle_actions()` adds a GDestroyNotify callback for *user_data*.

action_group : the action group
entries : an array of toggle action descriptions
n_entries : the number of entries
user_data : data to pass to the action callbacks
destroy : destroy notification callback for *user_data*

Since 2.4

GtkRadioActionEntry

```
typedef struct {
  const gchar *name;
  const gchar *stock_id;
  const gchar *label;
  const gchar *accelerator;
  const gchar *tooltip;
```

```
  gint   value;
} GtkRadioActionEntry;
```

GtkRadioActionEntry structs are used with gtk_action_group_add_radio_actions() to construct groups of radio actions.

const gchar *name;	The name of the action.
const gchar *stock_id;	The stock id for the action.
const gchar *label;	The label for the action. This field should typically be marked for translation, see gtk_action_group_set_translation_domain().
const gchar *accelerator;	The accelerator for the action, in the format understood by gtk_accelerator_parse().
const gchar *tooltip;	The tooltip for the action. This field should typically be marked for translation, see gtk_action_group_set_translation_domain().
gint value;	The value to set on the radio action. See gtk_radio_action_get_current_value().

---

gtk_action_group_add_radio_actions ()

```
void        gtk_action_group_add_radio_actions
                                            (GtkActionGroup *action_group,
                                             const GtkRadioActionEntry *entries,
                                             guint n_entries,
                                             gint value,
                                             GCallback on_change,
                                             gpointer user_data);
```

This is a convenience routine to create a group of radio actions and add them to the action group.

The "changed" signal of the first radio action is connected to the *on_change* callback and the accel paths of the actions are set to <Actions>/*group-name*/*action-name*.

action_group :	the action group
entries :	an array of radio action descriptions
n_entries :	the number of entries
value :	the value of the action to activate initially, or -1 if no action should be activated
on_change :	the callback to connect to the changed signal
user_data :	data to pass to the action callbacks

Since 2.4

gtk_action_group_add_radio_actions_full ()

```
void        gtk_action_group_add_radio_actions_full
                                            (GtkActionGroup *action_group,
                                             const GtkRadioActionEntry *entries,
                                             guint n_entries,
                                             gint value,
                                             GCallback on_change,
                                             gpointer user_data,
                                             GDestroyNotify destroy);
```

This variant of gtk_action_group_add_radio_actions() adds a GDestroyNotify callback for *user_data*.

action_group :	the action group
entries :	an array of radio action descriptions
n_entries :	the number of entries
value :	the value of the action to activate initially, or -1 if no action should be activated
on_change :	the callback to connect to the changed signal
user_data :	data to pass to the action callbacks
destroy :	destroy notification callback for *user_data*

Since 2.4

gtk_action_group_set_translate_func ()

```
void        gtk_action_group_set_translate_func
                                            (GtkActionGroup *action_group,
                                             GtkTranslateFunc func,
                                             gpointer data,
                                             GtkDestroyNotify notify);
```

Sets a function to be used for translating the *label* and *tooltip* of GtkActionGroupEntrys added by gtk_action_group_add_actions().

If you're using gettext(), it is enough to set the translation domain with gtk_action_group_set_translation_domain().

action_group :	a GtkActionGroup
func :	a GtkTranslateFunc
data :	data to be passed to *func* and *notify*
notify :	a GtkDestroyNotify function to be called when *action_group* is destroyed and when the translation function is changed again

Since 2.4

---

gtk_action_group_set_translation_domain ()

```
void        gtk_action_group_set_translation_domain
                                            (GtkActionGroup *action_group,
                                             const gchar *domain);
```

Sets the translation domain and uses dgettext() for translating the *label* and *tooltip* of GtkActionEntrys added by gtk_action_group_add_actions().

If you're not using gettext() for localization, see gtk_action_group_set_translate_func().

action_group :	a GtkActionGroup
domain :	the translation domain to use for dgettext() calls

Since 2.4

---

gtk_action_group_translate_string ()

```
G_CONST_RETURN gchar* gtk_action_group_translate_string
                                            (GtkActionGroup *action_group,
                                             const gchar *string);
```

Translates a string using the specified translate_func(). This is mainly intended for language bindings.

action_group :	a GtkActionGroup
string :	a string
Returns :	the translation of *string*

Since 2.6

Properties

The "name" property

```
  "name"                gchararray            : Read / Write / Construct Only
```

A name for the action group.

Default value: NULL

The "sensitive" property

```
  "sensitive"           gboolean              : Read / Write
```

Whether the action group is enabled.

Default value: TRUE

The "visible" property

```
  "visible"             gboolean              : Read / Write
```

Whether the action group is visible.

Default value: TRUE

Signals

The "connect-proxy" signal

```
void        user_function                   (GtkActionGroup *action_group,
                                             GtkAction *action,
                                             GtkWidget *proxy,
                                             gpointer user_data);
```

The connect_proxy signal is emitted after connecting a proxy to an action in the group. Note that the proxy may have been connected to a different action before.

This is intended for simple customizations for which a custom action class would be too clumsy, e.g. showing tooltips for menuitems in the statusbar.

GtkUIManager proxies the signal and provides global notification just before any action is connected to a proxy, which is probably more convenient to use.

action_group : the group
action : the action
proxy : the proxy
user_data : user data set when the signal handler was connected.

Since 2.4

The "disconnect-proxy" signal

```
void        user_function                  (GtkActionGroup *action_group,
                                            GtkAction *action,
                                            GtkWidget *proxy,
                                            gpointer user_data);
```

The disconnect_proxy signal is emitted after disconnecting a proxy from an action in the group.

GtkUIManager proxies the signal and provides global notification just before any action is connected to a proxy, which is probably more convenient to use.

action_group : the group
action : the action
proxy : the proxy
user_data : user data set when the signal handler was connected.

Since 2.4

The "post-activate" signal

```
void        user_function                   (GtkActionGroup *action_group,
                                             GtkAction *action,
                                             gpointer user_data);
```

The post_activate signal is emitted just after the *action* in the *action_group* is activated

This is intended for GtkUIManager to proxy the signal and provide global notification just after any action is activated.

action_group : the group
action : the action
user_data : user data set when the signal handler was connected.

Since 2.4

The "pre-activate" signal

```
void        user_function                   (GtkActionGroup *action_group,
                                             GtkAction *action,
                                             gpointer user_data);
```

The pre_activate signal is emitted just before the *action* in the *action_group* is activated

This is intended for GtkUIManager to proxy the signal and provide global notification just before any action is activated.

action_group : the group
action : the action
user_data : user data set when the signal handler was connected.

Since 2.4

<< GtkUIManager **GtkAction >>**

GtkAction

GtkAction — An action which can be triggered by a menu or toolbar item

Synopsis

```
#include <gtk/gtk.h>

            GtkAction;
GtkAction*  gtk_action_new                  (const gchar *name,
                                             const gchar *label,
                                             const gchar *tooltip,
                                             const gchar *stock_id);
const gchar* gtk_action_get_name            (GtkAction *action);
gboolean    gtk_action_is_sensitive         (GtkAction *action);
gboolean    gtk_action_get_sensitive        (GtkAction *action);
void        gtk_action_set_sensitive        (GtkAction *action,
                                             gboolean sensitive);
gboolean    gtk_action_is_visible           (GtkAction *action);
gboolean    gtk_action_get_visible          (GtkAction *action);
void        gtk_action_set_visible          (GtkAction *action,
                                             gboolean visible);
void        gtk_action_activate             (GtkAction *action);
GtkWidget*  gtk_action_create_icon          (GtkAction *action,
                                             GtkIconSize icon_size);
GtkWidget*  gtk_action_create_menu_item     (GtkAction *action);
GtkWidget*  gtk_action_create_tool_item     (GtkAction *action);
void        gtk_action_connect_proxy        (GtkAction *action,
                                             GtkWidget *proxy);
void        gtk_action_disconnect_proxy     (GtkAction *action,
                                             GtkWidget *proxy);
GSList*     gtk_action_get_proxies          (GtkAction *action);
void        gtk_action_connect_accelerator  (GtkAction *action);
void        gtk_action_disconnect_accelerator
                                            (GtkAction *action);
void        gtk_action_block_activate_from  (GtkAction *action,
```

```
                                                GtkWidget *proxy);
void        gtk_action_unblock_activate_from
                                               (GtkAction *action,
                                                GtkWidget *proxy);
void        gtk_action_set_accel_path          (GtkAction *action,
                                                const gchar *accel_path);
void        gtk_action_set_accel_group         (GtkAction *action,
                                                GtkAccelGroup *accel_group);
```

Object Hierarchy

```
  GObject
   +----GtkAction
         +----GtkToggleAction
```

Properties

```
  "action-group"          GtkActionGroup        : Read / Write
  "hide-if-empty"         gboolean              : Read / Write
  "is-important"          gboolean              : Read / Write
  "label"                 gchararray            : Read / Write
  "name"                  gchararray            : Read / Write / Construct Only
  "sensitive"             gboolean              : Read / Write
  "short-label"           gchararray            : Read / Write
  "stock-id"              gchararray            : Read / Write
  "tooltip"               gchararray            : Read / Write
  "visible"               gboolean              : Read / Write
  "visible-horizontal"    gboolean              : Read / Write
  "visible-overflown"     gboolean              : Read / Write
  "visible-vertical"      gboolean              : Read / Write
```

Signal Prototypes

```
"activate"  void        user_function      (GtkAction *action,
```

```
                                                gpointer user_data);
```

Description

Actions represent operations that the user can be perform, along with some information how it should be presented in the interface. Each action provides methods to create icons, menu items and toolbar items representing itself.

As well as the callback that is called when the action gets activated, the following also gets associated with the action:

- a name (not translated, for path lookup)
- a label (translated, for display)
- an accelerator
- whether label indicates a stock id
- a tooltip (optional, translated)
- a toolbar label (optional, shorter than label)

The action will also have some state information:

- visible (shown/hidden)
- sensitive (enabled/disabled)

Apart from regular actions, there are toggle actions, which can be toggled between two states and radio actions, of which only one in a group can be in the "active" state. Other actions can be implemented as GtkAction subclasses.

Each action can have one or more proxy menu item, toolbar button or other proxy widgets. Proxies mirror the state of the action (text label, tooltip, icon, visible, sensitive, etc), and should change when the action's state changes. When the proxy is activated, it should activate its action.

Details

GtkAction

```
typedef struct _GtkAction GtkAction;
```

The GtkAction struct contains only private members and should not be accessed directly.

---

gtk_action_new ()

```
GtkAction*  gtk_action_new                  (const gchar *name,
```

```
                                                const gchar *label,
                                                const gchar *tooltip,
                                                const gchar *stock_id);
```

Creates a new GtkAction object. To add the action to a GtkActionGroup and set the accelerator for the action, call `gtk_action_group_add_action_with_accel()`. See the section called “UI Definitions” for information on allowed action names.

name : A unique name for the action
label : the label displayed in menu items and on buttons
tooltip : a tooltip for the action
stock_id : the stock icon to display in widgets representing the action
Returns : a new GtkAction

Since 2.4

---

gtk_action_get_name ()

```
const gchar* gtk_action_get_name            (GtkAction *action);
```

Returns the name of the action.

action : the action object
Returns : the name of the action. The string belongs to GTK+ and should not be freed.

Since 2.4

---

gtk_action_is_sensitive ()

```
gboolean    gtk_action_is_sensitive         (GtkAction *action);
```

Returns whether the action is effectively sensitive.

action : the action object

Returns : TRUE if the action and its associated action group are both sensitive.

Since 2.4

---

gtk_action_get_sensitive ()

```
gboolean    gtk_action_get_sensitive        (GtkAction *action);
```

Returns whether the action itself is sensitive. Note that this doesn't necessarily mean effective sensitivity. See `gtk_action_is_sensitive()` for that.

`action` : the action object

Returns : TRUE if the action itself is sensitive.

Since 2.4

---

gtk_action_set_sensitive ()

```
void        gtk_action_set_sensitive        (GtkAction *action,
                                             gboolean sensitive);
```

Sets the ::sensitive property of the action to *`sensitive`*. Note that this doesn't necessarily mean effective sensitivity. See `gtk_action_is_sensitive()` for that.

`action` : the action object

`sensitive` : TRUE to make the action sensitive

Since 2.6

---

gtk_action_is_visible ()

```
gboolean    gtk_action_is_visible           (GtkAction *action);
```

Returns whether the action is effectively visible.

action : the action object
Returns : TRUE if the action and its associated action group are both visible.

Since 2.4

---

gtk_action_get_visible ()

```
gboolean    gtk_action_get_visible          (GtkAction *action);
```

Returns whether the action itself is visible. Note that this doesn't necessarily mean effective visibility. See gtk_action_is_sensitive() for that.

action : the action object
Returns : TRUE if the action itself is visible.

Since 2.4

---

gtk_action_set_visible ()

```
void        gtk_action_set_visible          (GtkAction *action,
                                             gboolean visible);
```

Sets the ::visible property of the action to *visible*. Note that this doesn't necessarily mean effective visibility. See gtk_action_is_visible() for that.

action : the action object
visible : TRUE to make the action visible

Since 2.6

---

gtk_action_activate ()

```
void        gtk_action_activate             (GtkAction *action);
```

Emits the "activate" signal on the specified action, if it isn't insensitive. This gets called by the proxy widgets when they get activated.

It can also be used to manually activate an action.

action : the action object

Since 2.4

---

gtk_action_create_icon ()

```
GtkWidget*  gtk_action_create_icon          (GtkAction *action,
                                             GtkIconSize icon_size);
```

This function is intended for use by action implementations to create icons displayed in the proxy widgets.

action : the action object
icon_size : the size of the icon that should be created.
Returns : a widget that displays the icon for this action.

Since 2.4

---

gtk_action_create_menu_item ()

```
GtkWidget*  gtk_action_create_menu_item     (GtkAction *action);
```

Creates a menu item widget that proxies for the given action.

action : the action object
Returns : a menu item connected to the action.

Since 2.4

---

gtk_action_create_tool_item ()

```
GtkWidget*  gtk_action_create_tool_item     (GtkAction *action);
```

Creates a toolbar item widget that proxies for the given action.

action : the action object
Returns : a toolbar item connected to the action.

Since 2.4

---

gtk_action_connect_proxy ()

```
void        gtk_action_connect_proxy        (GtkAction *action,
                                             GtkWidget *proxy);
```

Connects a widget to an action object as a proxy. Synchronises various properties of the action with the widget (such as label text, icon, tooltip, etc), and attaches a callback so that the action gets activated when the proxy widget does.

If the widget is already connected to an action, it is disconnected first.

action : the action object
proxy : the proxy widget

Since 2.4

---

gtk_action_disconnect_proxy ()

```
void        gtk_action_disconnect_proxy     (GtkAction *action,
                                             GtkWidget *proxy);
```

Disconnects a proxy widget from an action. Does *not* destroy the widget, however.

action : the action object
proxy : the proxy widget

Since 2.4

---

gtk_action_get_proxies ()

```
GSList*     gtk_action_get_proxies          (GtkAction *action);
```

Returns the proxy widgets for an action.

action : the action object
Returns : a GSList of proxy widgets. The list is owned by the action and must not be modified.

Since 2.4

---

gtk_action_connect_accelerator ()

```
void        gtk_action_connect_accelerator  (GtkAction *action);
```

Installs the accelerator for *action* if *action* has an accel path and group. See `gtk_action_set_accel_path()` and `gtk_action_set_accel_group()`

Since multiple proxies may independently trigger the installation of the accelerator, the *action* counts the number of times this function has been called and doesn't remove the accelerator until `gtk_action_disconnect_accelerator()` has been called as many times.

action : a GtkAction

Since 2.4

---

gtk_action_disconnect_accelerator ()

```
void        gtk_action_disconnect_accelerator
                                            (GtkAction *action);
```

Undoes the effect of one call to gtk_action_connect_accelerator().

action : a GtkAction

Since 2.4

---

gtk_action_block_activate_from ()

```
void        gtk_action_block_activate_from  (GtkAction *action,
                                             GtkWidget *proxy);
```

Disables calls to the gtk_action_activate() function by signals on the given proxy widget. This is used to break notification loops for things like check or radio actions.

This function is intended for use by action implementations.

action : the action object
proxy : a proxy widget

Since 2.4

---

gtk_action_unblock_activate_from ()

```
void        gtk_action_unblock_activate_from
                                            (GtkAction *action,
                                             GtkWidget *proxy);
```

Re-enables calls to the gtk_action_activate() function by signals on the given proxy widget. This undoes the blocking done by gtk_action_block_activate_from().

This function is intended for use by action implementations.

action : the action object
proxy : a proxy widget

Since 2.4

---

gtk_action_set_accel_path ()

```
void        gtk_action_set_accel_path       (GtkAction *action,
                                             const gchar *accel_path);
```

Sets the accel path for this action. All proxy widgets associated with the action will have this accel path, so that their accelerators are consistent.

action : the action object
accel_path : the accelerator path

Since 2.4

---

gtk_action_set_accel_group ()

```
void        gtk_action_set_accel_group      (GtkAction *action,
                                             GtkAccelGroup *accel_group);
```

Sets the GtkAccelGroup in which the accelerator for this action will be installed.

action : the action object
accel_group : a GtkAccelGroup or NULL

Since 2.4

Properties

The "action-group" property

```
"action-group"          GtkActionGroup          : Read / Write
```

The GtkActionGroup this GtkAction is associated with, or NULL (for internal use).

The "hide-if-empty" property

```
"hide-if-empty"         gboolean                : Read / Write
```

When TRUE, empty menu proxies for this action are hidden.

Default value: TRUE

The "is-important" property

```
"is-important"          gboolean                : Read / Write
```

Whether the action is considered important. When TRUE, toolitem proxies for this action show text in GTK_TOOLBAR_BOTH_HORIZ mode.

Default value: FALSE

The "label" property

```
"label"                 gchararray              : Read / Write
```

The label used for menu items and buttons that activate this action.

Default value: NULL

The "name" property

```
  "name"                gchararray            : Read / Write / Construct Only
```

A unique name for the action.

Default value: NULL

The "sensitive" property

```
  "sensitive"           gboolean              : Read / Write
```

Whether the action is enabled.

Default value: TRUE

The "short-label" property

```
  "short-label"         gchararray            : Read / Write
```

A shorter label that may be used on toolbar buttons.

Default value: NULL

The "stock-id" property

```
  "stock-id"            gchararray            : Read / Write
```

The stock icon displayed in widgets representing this action.

Default value: NULL

The "tooltip" property

```
"tooltip"              gchararray            : Read / Write
```

A tooltip for this action.

Default value: NULL

The "visible" property

```
"visible"              gboolean              : Read / Write
```

Whether the action is visible.

Default value: TRUE

The "visible-horizontal" property

```
"visible-horizontal"   gboolean              : Read / Write
```

Whether the toolbar item is visible when the toolbar is in a horizontal orientation.

Default value: TRUE

The "visible-overflown" property

```
"visible-overflown"    gboolean              : Read / Write
```

When TRUE, toolitem proxies for this action are represented in the toolbar overflow menu.

Default value: TRUE

Since 2.6

The "visible-vertical" property

```
  "visible-vertical"     gboolean              : Read / Write
```

Whether the toolbar item is visible when the toolbar is in a vertical orientation.

Default value: TRUE

Signals

The "activate" signal

```
void        user_function                  (GtkAction *action,
                                            gpointer user_data);
```

The "activate" signal is emitted when the action is activated.

action : the GtkAction

user_data : user data set when the signal handler was connected.

Since 2.4

See Also

GtkActionGroup, GtkUIManager

GtkToggleAction

GtkToggleAction — An action which can be toggled between two states

Synopsis

```
#include <gtk/gtk.h>

            GtkToggleAction;
GtkToggleAction* gtk_toggle_action_new      (const gchar *name,
                                             const gchar *label,
                                             const gchar *tooltip,
                                             const gchar *stock_id);
void        gtk_toggle_action_toggled       (GtkToggleAction *action);
void        gtk_toggle_action_set_active    (GtkToggleAction *action,
                                             gboolean is_active);
gboolean    gtk_toggle_action_get_active    (GtkToggleAction *action);
void        gtk_toggle_action_set_draw_as_radio
                                            (GtkToggleAction *action,
                                             gboolean draw_as_radio);
gboolean    gtk_toggle_action_get_draw_as_radio
                                            (GtkToggleAction *action);
```

Object Hierarchy

```
  GObject
   +----GtkAction
         +----GtkToggleAction
               +----GtkRadioAction
```

Properties

```
  "draw-as-radio"       gboolean              : Read / Write
```

Signal Prototypes

```
"toggled"   void        user_function      (GtkToggleAction *toggleaction,
                                            gpointer user_data);
```

Description

A GtkToggleAction corresponds roughly to a GtkCheckMenuItem. It has an "active" state specifying whether the action has been checked or not.

Details

GtkToggleAction

```
typedef struct _GtkToggleAction GtkToggleAction;
```

The GtkToggleAction struct contains only private members and should not be accessed directly.

---

gtk_toggle_action_new ()

```
GtkToggleAction* gtk_toggle_action_new      (const gchar *name,
                                             const gchar *label,
                                             const gchar *tooltip,
                                             const gchar *stock_id);
```

Creates a new GtkToggleAction object. To add the action to a GtkActionGroup and set the accelerator for the action, call `gtk_action_group_add_action_with_accel()`.

name : A unique name for the action
label : The label displayed in menu items and on buttons
tooltip : A tooltip for the action
stock_id : The stock icon to display in widgets representing the action
Returns : a new GtkToggleAction

Since 2.4

---

gtk_toggle_action_toggled ()

```
void        gtk_toggle_action_toggled       (GtkToggleAction *action);
```

Emits the "toggled" signal on the toggle action.

action : the action object

Since 2.4

---

gtk_toggle_action_set_active ()

```
void        gtk_toggle_action_set_active    (GtkToggleAction *action,
                                             gboolean is_active);
```

Sets the checked state on the toggle action.

action : the action object
is_active : whether the action should be checked or not

Since 2.4

---

gtk_toggle_action_get_active ()

```
gboolean    gtk_toggle_action_get_active    (GtkToggleAction *action);
```

Returns the checked state of the toggle action.

action : the action object
Returns : the checked state of the toggle action

Since 2.4

---

gtk_toggle_action_set_draw_as_radio ()

```
void        gtk_toggle_action_set_draw_as_radio
                                            (GtkToggleAction *action,
                                             gboolean draw_as_radio);
```

Sets whether the action should have proxies like a radio action.

action : the action object
draw_as_radio : whether the action should have proxies like a radio action

Since 2.4

---

gtk_toggle_action_get_draw_as_radio ()

```
gboolean    gtk_toggle_action_get_draw_as_radio
                                            (GtkToggleAction *action);
```

Returns whether the action should have proxies like a radio action.

action : the action object

Returns : whether the action should have proxies like a radio action.

Since 2.4

Properties

The "draw-as-radio" property

```
  "draw-as-radio"        gboolean              : Read / Write
```

Whether the proxies for this action look like radio action proxies.

Default value: FALSE

Signals

The "toggled" signal

```
void        user_function                     (GtkToggleAction *toggleaction,
                                               gpointer user_data);
```

toggleaction : the object which received the signal.
user_data : user data set when the signal handler was connected.

<< GtkAction **GtkRadioAction >>**

GtkRadioAction

GtkRadioAction — An action of which only one in a group can be active

Synopsis

```
#include <gtk/gtk.h>

            GtkRadioAction;
GtkRadioAction* gtk_radio_action_new        (const gchar *name,
                                             const gchar *label,
                                             const gchar *tooltip,
                                             const gchar *stock_id,
                                             gint value);
GSList*     gtk_radio_action_get_group      (GtkRadioAction *action);
void        gtk_radio_action_set_group      (GtkRadioAction *action,
                                             GSList *group);
gint        gtk_radio_action_get_current_value
                                            (GtkRadioAction *action);
```

Object Hierarchy

```
  GObject
   +----GtkAction
         +----GtkToggleAction
               +----GtkRadioAction
```

Properties

```
  "group"                GtkRadioAction        : Write
  "value"                gint                  : Read / Write
```

Signal Prototypes

```
"changed"   void        user_function      (GtkRadioAction *action,
                                            GtkRadioAction *current,
                                            gpointer user_data);
```

Description

A GtkRadioAction is similar to GtkRadioMenuItem. A number of radio actions can be linked together so that only one may be active at any one time.

Details

GtkRadioAction

```
typedef struct _GtkRadioAction GtkRadioAction;
```

The GtkRadioAction struct contains only private members and should not be accessed directly.

---

gtk_radio_action_new ()

```
GtkRadioAction* gtk_radio_action_new        (const gchar *name,
                                             const gchar *label,
                                             const gchar *tooltip,
```

```
                                                const gchar *stock_id,
                                                gint value);
```

Creates a new GtkRadioAction object. To add the action to a GtkActionGroup and set the accelerator for the action, call `gtk_action_group_add_action_with_accel()`.

name :	A unique name for the action
label :	The label displayed in menu items and on buttons
tooltip :	A tooltip for this action
stock_id :	The stock icon to display in widgets representing this action
value :	The value which `gtk_radio_action_get_current_value()` should return if this action is selected.
Returns :	a new GtkRadioAction

Since 2.4

---

gtk_radio_action_get_group ()

```
GSList*     gtk_radio_action_get_group      (GtkRadioAction *action);
```

Returns the list representing the radio group for this object

action :	the action object
Returns :	the list representing the radio group for this object

Since 2.4

---

gtk_radio_action_set_group ()

```
void        gtk_radio_action_set_group      (GtkRadioAction *action,
```

```
                                                GSList *group);
```

Sets the radio group for the radio action object.

action : the action object
group : a list representing a radio group

Since 2.4

---

gtk_radio_action_get_current_value ()

```
gint        gtk_radio_action_get_current_value
                                            (GtkRadioAction *action);
```

Obtains the value property of the the currently active member of the group to which *action* belongs.

action : a GtkRadioAction
Returns : The value of the currently active group member

Since 2.4

Properties

The "group" property

```
  "group"                GtkRadioAction        : Write
```

Sets a new group for a radio action.

Since 2.4

The "value" property

```
  "value"                gint                : Read / Write
```

The value is an arbitrary integer which can be used as a convenient way to determine which action in the group is currently active in an ::activate or ::changed signal handler. See `gtk_radio_action_get_current_value()` and GtkRadioActionEntry for convenient ways to get and set this property.

Default value: 0

Since 2.4

Signals

The "changed" signal

```
void        user_function                   (GtkRadioAction *action,
                                             GtkRadioAction *current,
                                             gpointer user_data);
```

The ::changed signal is emitted on every member of a radio group when the active member is changed. The signal gets emitted after the ::activate signals for the previous and current active members.

action :	the action on which the signal is emitted
current :	the member of *action*s group which has just been activated
user_data :	user data set when the signal handler was connected.

Since 2.4

<< GtkToggleAction **Selectors (File/Font/Color/Input Devices) >>**

Selectors (File/Font/Color/Input Devices)

GtkColorButton - A button to launch a color selection dialog
GtkColorSelection - A widget used to select a color
GtkColorSelectionDialog - A standard dialog box for selecting a color
GtkFileSelection - Prompt the user for a file or directory name
GtkFileChooser - File chooser interface used by GtkFileChooserWidget and GtkFileChooserDialog
GtkFileChooserButton - A button to launch a file selection dialog
GtkFileChooserDialog - A file chooser dialog, suitable for "File/Open" or "File/Save" commands
GtkFileChooserWidget - File chooser widget that can be embedded in other widgets
GtkFileFilter - A filter for selecting a file subset
GtkFontButton - A button to launch a font selection dialog
GtkFontSelection - A widget for selecting fonts
GtkFontSelectionDialog - A dialog box for selecting fonts
GtkInputDialog - Configure devices for the XInput extension

GtkColorButton

GtkColorButton — A button to launch a color selection dialog

Synopsis

```
#include <gtk/gtk.h>

            GtkColorButton;
GtkWidget*  gtk_color_button_new            (void);
GtkWidget*  gtk_color_button_new_with_color (const GdkColor *color);
void        gtk_color_button_set_color      (GtkColorButton *color_button,
                                             const GdkColor *color);
void        gtk_color_button_get_color      (GtkColorButton *color_button,
                                             GdkColor *color);
void        gtk_color_button_set_alpha      (GtkColorButton *color_button,
                                             guint16 alpha);
guint16     gtk_color_button_get_alpha      (GtkColorButton *color_button);
void        gtk_color_button_set_use_alpha  (GtkColorButton *color_button,
                                             gboolean use_alpha);
gboolean    gtk_color_button_get_use_alpha  (GtkColorButton *color_button);
void        gtk_color_button_set_title      (GtkColorButton *color_button,
                                             const gchar *title);
G_CONST_RETURN gchar* gtk_color_button_get_title
                                            (GtkColorButton *color_button);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkWidget
```

```
                +----GtkContainer
                      +----GtkBin
                            +----GtkButton
                                  +----GtkColorButton
```

Implemented Interfaces

GtkColorButton implements AtkImplementorIface.

Properties

```
  "alpha"                guint                 : Read / Write
  "color"                GdkColor              : Read / Write
  "title"                gchararray            : Read / Write
  "use-alpha"            gboolean              : Read / Write
```

Signal Prototypes

```
"color-set" void        user_function      (GtkColorButton *widget,
                                            gpointer user_data);
```

Description

The GtkColorButton is a button which displays the currently selected color an allows to open a color selection dialog to change the color. It is suitable widget for selecting a color in a preference dialog.

Details

GtkColorButton

```
typedef struct _GtkColorButton GtkColorButton;
```

---

gtk_color_button_new ()

```
GtkWidget*  gtk_color_button_new            (void);
```

Creates a new color button. This returns a widget in the form of a small button containing a swatch representing the current selected color. When the button is clicked, a color-selection dialog will open, allowing the user to select a color. The swatch will be updated to reflect the new color when the user finishes.

Returns : a new color button.

Since 2.4

---

gtk_color_button_new_with_color ()

```
GtkWidget*  gtk_color_button_new_with_color (const GdkColor *color);
```

Creates a new color button.

color : A GdkColor to set the current color with.

Returns : a new color button.

Since 2.4

---

gtk_color_button_set_color ()

```
void        gtk_color_button_set_color      (GtkColorButton *color_button,
                                             const GdkColor *color);
```

Sets the current color to be *color*.

color_button : a GtkColorButton.

color : A GdkColor to set the current color with.

Since 2.4

---

gtk_color_button_get_color ()

```
void        gtk_color_button_get_color      (GtkColorButton *color_button,
                                             GdkColor *color);
```

Sets *color* to be the current color in the GtkColorButton widget.

color_button : a GtkColorButton.
color : a GdkColor to fill in with the current color.

Since 2.4

---

gtk_color_button_set_alpha ()

```
void        gtk_color_button_set_alpha      (GtkColorButton *color_button,
                                             guint16 alpha);
```

Sets the current opacity to be *alpha*.

color_button : a GtkColorButton.
alpha : an integer between 0 and 65535.

Since 2.4

---

gtk_color_button_get_alpha ()

```
guint16     gtk_color_button_get_alpha      (GtkColorButton *color_button);
```

Returns the current alpha value.

color_button : a GtkColorButton.
Returns : an integer between 0 and 65535.

Since 2.4

---

gtk_color_button_set_use_alpha ()

```
void        gtk_color_button_set_use_alpha  (GtkColorButton *color_button,
                                             gboolean use_alpha);
```

Sets whether or not the color button should use the alpha channel.

color_button : a GtkColorButton.
use_alpha : TRUE if color button should use alpha channel, FALSE if not.

Since 2.4

---

gtk_color_button_get_use_alpha ()

```
gboolean    gtk_color_button_get_use_alpha  (GtkColorButton *color_button);
```

Does the color selection dialog use the alpha channel?

color_button : a GtkColorButton.
Returns : TRUE if the color sample uses alpha channel, FALSE if not.

Since 2.4

---

gtk_color_button_set_title ()

```
void        gtk_color_button_set_title      (GtkColorButton *color_button,
                                             const gchar *title);
```

Sets the title for the color selection dialog.

color_button : a GtkColorButton
title : String containing new window title.

Since 2.4

---

gtk_color_button_get_title ()

```
G_CONST_RETURN gchar* gtk_color_button_get_title
                                            (GtkColorButton *color_button);
```

Gets the title of the color selection dialog.

color_button : a GtkColorButton
Returns : An internal string, do not free the return value

Since 2.4

Properties

The "alpha" property

```
  "alpha"                guint                 : Read / Write
```

The selected opacity value (0 fully transparent, 65535 fully opaque).

Allowed values: <= 65535

Default value: 65535

Since 2.4

The "color" property

"color"	GdkColor	: Read / Write

The selected color.

Since 2.4

The "title" property

"title"	gchararray	: Read / Write

The title of the color selection dialog

Default value: "Pick a Color"

Since 2.4

The "use-alpha" property

"use-alpha"	gboolean	: Read / Write

If this property is set to TRUE, the color swatch on the button is rendered against a checkerboard background to show its opacity and the opacity slider is displayed in the color selection dialog.

Default value: FALSE

Since 2.4

Signals

The "color-set" signal

```
void        user_function                   (GtkColorButton *widget,
                                             gpointer user_data);
```

The ::color-set signal is emitted when the user selects a color. When handling this signal, use `gtk_color_button_get_color()` and `gtk_color_button_get_alpha()` to find out which color was just selected.

widget : the object which received the signal.
user_data : user data set when the signal handler was connected.

Since 2.4

See Also

GtkColorSelectionDialog, GtkFontButton

<< Selectors (File/Font/Color/Input Devices) **GtkColorSelection >>**

GtkColorSelection

GtkColorSelection — A widget used to select a color

Synopsis

```
#include <gtk/gtk.h>

            GtkColorSelection;
GtkWidget*  gtk_color_selection_new         (void);
void        gtk_color_selection_set_update_policy
                                            (GtkColorSelection *colorsel,
                                             GtkUpdateType policy);
void        gtk_color_selection_set_has_opacity_control
                                            (GtkColorSelection *colorsel,
                                             gboolean has_opacity);
gboolean    gtk_color_selection_get_has_opacity_control
                                            (GtkColorSelection *colorsel);
void        gtk_color_selection_set_has_palette
                                            (GtkColorSelection *colorsel,
                                             gboolean has_palette);
gboolean    gtk_color_selection_get_has_palette
                                            (GtkColorSelection *colorsel);
guint16     gtk_color_selection_get_current_alpha
                                            (GtkColorSelection *colorsel);
void        gtk_color_selection_set_current_alpha
                                            (GtkColorSelection *colorsel,
                                             guint16 alpha);
void        gtk_color_selection_get_current_color
                                            (GtkColorSelection *colorsel,
                                             GdkColor *color);
void        gtk_color_selection_set_current_color
                                            (GtkColorSelection *colorsel,
                                             const GdkColor *color);
guint16     gtk_color_selection_get_previous_alpha
                                            (GtkColorSelection *colorsel);
void        gtk_color_selection_set_previous_alpha
```

```
                                                (GtkColorSelection *colorsel,
                                                 guint16 alpha);
void            gtk_color_selection_get_previous_color
                                                (GtkColorSelection *colorsel,
                                                 GdkColor *color);
void            gtk_color_selection_set_previous_color
                                                (GtkColorSelection *colorsel,
                                                 const GdkColor *color);
gboolean        gtk_color_selection_is_adjusting
                                                (GtkColorSelection *colorsel);
gboolean        gtk_color_selection_palette_from_string
                                                (const gchar *str,
                                                 GdkColor **colors,
                                                 gint *n_colors);
gchar*          gtk_color_selection_palette_to_string
                                                (const GdkColor *colors,
                                                 gint n_colors);
GtkColorSelectionChangePaletteFunc gtk_color_selection_set_change_palette_hook
                                                (GtkColorSelectionChangePaletteFunc
func);
void            (*GtkColorSelectionChangePaletteFunc)
                                                (const GdkColor *colors,
                                                 gint n_colors);
GtkColorSelectionChangePaletteWithScreenFunc
gtk_color_selection_set_change_palette_with_screen_hook

(GtkColorSelectionChangePaletteWithScreenFunc func);
void            (*GtkColorSelectionChangePaletteWithScreenFunc)
                                                (GdkScreen *screen,
                                                 const GdkColor *colors,
                                                 gint n_colors);
void            gtk_color_selection_set_color   (GtkColorSelection *colorsel,
                                                 gdouble *color);
void            gtk_color_selection_get_color   (GtkColorSelection *colorsel,
                                                 gdouble *color);
```

Object Hierarchy

```
  GObject
   +----GtkObject
```

```
          +----GtkWidget
                +----GtkContainer
                      +----GtkBox
                            +----GtkVBox
                                  +----GtkColorSelection
```

Implemented Interfaces

GtkColorSelection implements AtkImplementorIface.

Properties

```
  "current-alpha"         guint                : Read / Write
  "current-color"         GdkColor             : Read / Write
  "has-opacity-control"   gboolean             : Read / Write
  "has-palette"           gboolean             : Read / Write
```

Signal Prototypes

```
"color-changed"
            void        user_function      (GtkColorSelection *colorselection,
                                            gpointer user_data);
```

Description

The GtkColorSelection is a widget that is used to select a color. It consists of a color wheel and number of sliders and entry boxes for color parameters such as hue, saturation, value, red, green, blue, and opacity. It is found on the standard color selection dialog box GtkColorSelectionDialog.

Details

GtkColorSelection

```
typedef struct _GtkColorSelection GtkColorSelection;
```

The GtkColorSelection-struct struct contains private data only, and should be accessed using the functions below.

---

gtk_color_selection_new ()

```
GtkWidget*  gtk_color_selection_new         (void);
```

Creates a new GtkColorSelection.

Returns : a new GtkColorSelection

---

gtk_color_selection_set_update_policy ()

```
void        gtk_color_selection_set_update_policy
                                            (GtkColorSelection *colorsel,
                                             GtkUpdateType policy);
```

Warning

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

Sets the policy controlling when the color_changed signals are emitted. The available policies are:

- `GTK_UPDATE_CONTINUOUS` - signals are sent continuously as the color selection changes.
- `GTK_UPDATE_DISCONTINUOUS` - signals are sent only when the mouse button is released.
- `GTK_UPDATE_DELAYED` - signals are sent when the mouse button is released or when the mouse has been motionless for a period of time.

`colorsel` : a GtkColorSelection.

`policy` : a GtkUpdateType value indicating the desired policy.

---

gtk_color_selection_set_has_opacity_control ()

```
void        gtk_color_selection_set_has_opacity_control
                                            (GtkColorSelection *colorsel,
                                             gboolean has_opacity);
```

Sets the *colorsel* to use or not use opacity.

colorsel : a GtkColorSelection.
has_opacity : TRUE if *colorsel* can set the opacity, FALSE otherwise.

---

gtk_color_selection_get_has_opacity_control ()

```
gboolean    gtk_color_selection_get_has_opacity_control
                                            (GtkColorSelection *colorsel);
```

Determines whether the colorsel has an opacity control.

colorsel : a GtkColorSelection.
Returns : TRUE if the *colorsel* has an opacity control. FALSE if it does't.

---

gtk_color_selection_set_has_palette ()

```
void        gtk_color_selection_set_has_palette
                                            (GtkColorSelection *colorsel,
                                             gboolean has_palette);
```

Shows and hides the palette based upon the value of *has_palette*.

colorsel : a GtkColorSelection.
has_palette : TRUE if palette is to be visible, FALSE otherwise.

---

gtk_color_selection_get_has_palette ()

```
gboolean    gtk_color_selection_get_has_palette
                                            (GtkColorSelection *colorsel);
```

Determines whether the color selector has a color palette.

colorsel : a GtkColorSelection.

Returns : TRUE if the selector has a palette. FALSE if it hasn't.

---

gtk_color_selection_get_current_alpha ()

```
guint16     gtk_color_selection_get_current_alpha
                                            (GtkColorSelection *colorsel);
```

Returns the current alpha value.

colorsel : a GtkColorSelection.

Returns : an integer between 0 and 65535.

---

gtk_color_selection_set_current_alpha ()

```
void        gtk_color_selection_set_current_alpha
                                            (GtkColorSelection *colorsel,
                                             guint16 alpha);
```

Sets the current opacity to be *alpha*. The first time this is called, it will also set the original opacity to be *alpha* too.

colorsel : a GtkColorSelection.

alpha : an integer between 0 and 65535.

---

gtk_color_selection_get_current_color ()

```
void        gtk_color_selection_get_current_color
                                            (GtkColorSelection *colorsel,
                                             GdkColor *color);
```

Sets *color* to be the current color in the GtkColorSelection widget.

colorsel : a GtkColorSelection.

color : a GdkColor to fill in with the current color.

---

gtk_color_selection_set_current_color ()

```
void        gtk_color_selection_set_current_color
                                            (GtkColorSelection *colorsel,
                                             const GdkColor *color);
```

Sets the current color to be *color*. The first time this is called, it will also set the original color to be *color* too.

colorsel : a GtkColorSelection.

color : A GdkColor to set the current color with.

---

gtk_color_selection_get_previous_alpha ()

```
guint16     gtk_color_selection_get_previous_alpha
                                            (GtkColorSelection *colorsel);
```

Returns the previous alpha value.

colorsel : a GtkColorSelection.

Returns : an integer between 0 and 65535.

---

gtk_color_selection_set_previous_alpha ()

```
void        gtk_color_selection_set_previous_alpha
                                            (GtkColorSelection *colorsel,
                                             guint16 alpha);
```

Sets the 'previous' alpha to be *alpha*. This function should be called with some hesitations, as it might seem confusing to have that alpha change.

colorsel : a GtkColorSelection.

alpha : an integer between 0 and 65535.

---

gtk_color_selection_get_previous_color ()

```
void        gtk_color_selection_get_previous_color
                                            (GtkColorSelection *colorsel,
                                             GdkColor *color);
```

Fills *color* in with the original color value.

colorsel : a GtkColorSelection.

color : a GdkColor to fill in with the original color value.

---

gtk_color_selection_set_previous_color ()

```
void        gtk_color_selection_set_previous_color
                                            (GtkColorSelection *colorsel,
                                             const GdkColor *color);
```

Sets the 'previous' color to be *color*. This function should be called with some hesitations, as it might seem confusing to have that color change. Calling gtk_color_selection_set_current_color() will also set this color the first time it is called.

colorsel : a GtkColorSelection.

color : a GdkColor to set the previous color with.

---

gtk_color_selection_is_adjusting ()

```
gboolean    gtk_color_selection_is_adjusting
                                            (GtkColorSelection *colorsel);
```

Gets the current state of the *colorsel*.

colorsel : a GtkColorSelection.

Returns : TRUE if the user is currently dragging a color around, and FALSE if the selection has stopped.

---

gtk_color_selection_palette_from_string ()

```
gboolean    gtk_color_selection_palette_from_string
                                            (const gchar *str,
                                             GdkColor **colors,
                                             gint *n_colors);
```

Parses a color palette string; the string is a colon-separated list of color names readable by gdk_color_parse().

str :	a string encoding a color palette.
colors :	return location for allocated array of GdkColor.
n_colors :	return location for length of array.
Returns :	TRUE if a palette was successfully parsed.

---

gtk_color_selection_palette_to_string ()

```
gchar*      gtk_color_selection_palette_to_string
                                            (const GdkColor *colors,
                                             gint n_colors);
```

Encodes a palette as a string, useful for persistent storage.

colors :	an array of colors.
n_colors :	length of the array.
Returns :	allocated string encoding the palette.

---

gtk_color_selection_set_change_palette_hook ()

```
GtkColorSelectionChangePaletteFunc gtk_color_selection_set_change_palette_hook
                                            (GtkColorSelectionChangePaletteFunc
func);
```

Warning

gtk_color_selection_set_change_palette_hook is deprecated and should not be used in newly-written code. This function is deprecated in favor of gtk_color_selection_set_change_palette_with_screen_hook(), and does not work in multihead environments.

Installs a global function to be called whenever the user tries to modify the palette in a color selection. This function should save the new palette contents, and update the GtkSettings property "gtk-color-palette" so all GtkColorSelection widgets will be modified.

func : a function to call when the custom palette needs saving.

Returns : the previous change palette hook (that was replaced).

GtkColorSelectionChangePaletteFunc ()

```
void        (*GtkColorSelectionChangePaletteFunc)
                                            (const GdkColor *colors,
                                             gint n_colors);
```

colors :

n_colors :

---

gtk_color_selection_set_change_palette_with_screen_hook ()

```
GtkColorSelectionChangePaletteWithScreenFunc
gtk_color_selection_set_change_palette_with_screen_hook

(GtkColorSelectionChangePaletteWithScreenFunc func);
```

Installs a global function to be called whenever the user tries to modify the palette in a color selection. This function should save the new palette contents, and update the GtkSettings property "gtk-color-palette" so all GtkColorSelection widgets will be modified.

func : a function to call when the custom palette needs saving.

Returns : the previous change palette hook (that was replaced).

Since 2.2

GtkColorSelectionChangePaletteWithScreenFunc ()

```
void        (*GtkColorSelectionChangePaletteWithScreenFunc)
                                            (GdkScreen *screen,
```

```
                                                const GdkColor *colors,
                                                gint n_colors);
```

screen :
colors :
n_colors :

Since 2.2

---

gtk_color_selection_set_color ()

```
void        gtk_color_selection_set_color   (GtkColorSelection *colorsel,
                                             gdouble *color);
```

Warning

`gtk_color_selection_set_color` is deprecated and should not be used in newly-written code. Use `gtk_color_selection_set_current_color()` instead.

Sets the current color to be *color*. The first time this is called, it will also set the original color to be *color* too.

colorsel : a GtkColorSelection.
color : an array of 4 doubles specifying the red, green, blue and opacity to set the current color to.

---

gtk_color_selection_get_color ()

```
void        gtk_color_selection_get_color   (GtkColorSelection *colorsel,
                                             gdouble *color);
```

Warning

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

Sets *color* to be the current color in the GtkColorSelection widget.

This function is deprecated, use `gtk_color_selection_get_current_color()` instead.

colorsel : a GtkColorSelection.

color : an array of 4 gdouble to fill in with the current color.

Properties

The "current-alpha" property

```
"current-alpha"        guint                 : Read / Write
```

The current opacity value (0 fully transparent, 65535 fully opaque).

Allowed values: <= 65535

Default value: 65535

The "current-color" property

```
"current-color"        GdkColor              : Read / Write
```

The current color.

The "has-opacity-control" property

```
"has-opacity-control"  gboolean              : Read / Write
```

Whether the color selector should allow setting opacity.

Default value: FALSE

The "has-palette" property

```
"has-palette"          gboolean              : Read / Write
```

Whether a palette should be used.

Default value: FALSE

Signals

The "color-changed" signal

```
void        user_function                   (GtkColorSelection *colorselection,
                                             gpointer user_data);
```

This signal is emitted when the color changes in the GtkColorSelection according to its update policy.

colorselection : the object which received the signal.
user_data : user data set when the signal handler was connected.

<< GtkColorButton **GtkColorSelectionDialog >>**

GtkColorSelectionDialog

GtkColorSelectionDialog — A standard dialog box for selecting a color

Synopsis

```
#include <gtk/gtk.h>

            GtkColorSelectionDialog;
GtkWidget*  gtk_color_selection_dialog_new  (const gchar *title);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkWidget
               +----GtkContainer
                     +----GtkBin
                           +----GtkWindow
                                 +----GtkDialog
                                       +----GtkColorSelectionDialog
```

Implemented Interfaces

GtkColorSelectionDialog implements AtkImplementorIface.

Description

The GtkColorSelectionDialog provides a standard dialog which allows the user to select a color much like the GtkFileSelection provides a standard dialog for file selection.

Details

GtkColorSelectionDialog

```
typedef struct _GtkColorSelectionDialog GtkColorSelectionDialog;
```

The GtkColorSelectionDialog-struct struct contains the following fields. (These fields should be considered read-only. They should never be set by an application.)

GtkWidget *colorsel;	The GtkColorSelection widget contained within the dialog. Use this widget and its `gtk_color_selection_get_current_color()` function to gain access to the selected color. Connect a handler for this widget's color_changed signal to be notified when the color changes.
GtkWidget *ok_button;	The OK button widget contained within the dialog. Connect a handler for the clicked event.
GtkWidget *cancel_button;	The cancel button widget contained within the dialog. Connect a handler for the clicked event.
GtkWidget *help_button;	The help button widget contained within the dialog. Connect a handler for the clicked event.

---

gtk_color_selection_dialog_new ()

```
GtkWidget*  gtk_color_selection_dialog_new  (const gchar *title);
```

Creates a new GtkColorSelectionDialog.

title : a string containing the title text for the dialog.

Returns : a GtkColorSelectionDialog.

<< GtkColorSelection **GtkFileSelection >>**

GtkFileSelection

GtkFileSelection — Prompt the user for a file or directory name

Synopsis

```
#include <gtk/gtk.h>


            GtkFileSelection;
GtkWidget*  gtk_file_selection_new          (const gchar *title);
void        gtk_file_selection_set_filename (GtkFileSelection *filesel,
                                             const gchar *filename);
G_CONST_RETURN gchar* gtk_file_selection_get_filename
                                            (GtkFileSelection *filesel);
void        gtk_file_selection_complete     (GtkFileSelection *filesel,
                                             const gchar *pattern);
void        gtk_file_selection_show_fileop_buttons
                                            (GtkFileSelection *filesel);
void        gtk_file_selection_hide_fileop_buttons
                                            (GtkFileSelection *filesel);
gchar**     gtk_file_selection_get_selections
                                            (GtkFileSelection *filesel);
void        gtk_file_selection_set_select_multiple
                                            (GtkFileSelection *filesel,
                                             gboolean select_multiple);
gboolean    gtk_file_selection_get_select_multiple
                                            (GtkFileSelection *filesel);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkWidget
```

```
                +----GtkContainer
                      +----GtkBin
                            +----GtkWindow
                                  +----GtkDialog
                                        +----GtkFileSelection
```

Implemented Interfaces

GtkFileSelection implements AtkImplementorIface.

Properties

```
  "filename"              gchararray            : Read / Write
  "select-multiple"       gboolean              : Read / Write
  "show-fileops"          gboolean              : Read / Write
```

Description

GtkFileSelection should be used to retrieve file or directory names from the user. It will create a new dialog window containing a directory list, and a file list corresponding to the current working directory. The filesystem can be navigated using the directory list or the drop-down history menu. Alternatively, the TAB key can be used to navigate using filename completion - common in text based editors such as emacs and jed.

File selection dialogs are created with a call to `gtk_file_selection_new()`.

The default filename can be set using `gtk_file_selection_set_filename()` and the selected filename retrieved using `gtk_file_selection_get_filename()`.

Use `gtk_file_selection_complete()` to display files and directories that match a given pattern. This can be used for example, to show only *.txt files, or only files beginning with gtk*.

Simple file operations; create directory, delete file, and rename file, are available from buttons at the top of the dialog. These can be hidden using `gtk_file_selection_hide_fileop_buttons()` and shown again using `gtk_file_selection_show_fileop_buttons()`.

Example 1. Getting a filename from the user.

```
/* The file selection widget and the string to store the chosen filename */

void store_filename (GtkWidget *widget, gpointer user_data) {
```

```
   GtkWidget *file_selector = GTK_WIDGET (user_data);
   const gchar *selected_filename;

   selected_filename = gtk_file_selection_get_filename (GTK_FILE_SELECTION
(file_selector));
   g_print ("Selected filename: %s\n", selected_filename);
}

void create_file_selection (void) {

   GtkWidget *file_selector;

   /* Create the selector */

   file_selector = gtk_file_selection_new ("Please select a file for editing.");

   g_signal_connect (GTK_FILE_SELECTION (file_selector)->ok_button,
                     "clicked",
                     G_CALLBACK (store_filename),
                     file_selector);

   /* Ensure that the dialog box is destroyed when the user clicks a button. */

   g_signal_connect_swapped (GTK_FILE_SELECTION (file_selector)->ok_button,
                             "clicked",
                             G_CALLBACK (gtk_widget_destroy),
                             file_selector);

   g_signal_connect_swapped (GTK_FILE_SELECTION (file_selector)->cancel_button,
                             "clicked",
                             G_CALLBACK (gtk_widget_destroy),
                             file_selector);

   /* Display that dialog */

   gtk_widget_show (file_selector);
}
```

Details

GtkFileSelection

```
typedef struct {
  GtkWidget *dir_list;
  GtkWidget *file_list;
  GtkWidget *selection_entry;
  GtkWidget *selection_text;
  GtkWidget *main_vbox;
```

```
  GtkWidget *ok_button;
  GtkWidget *cancel_button;
  GtkWidget *help_button;
  GtkWidget *history_pulldown;
  GtkWidget *history_menu;
  GList     *history_list;
  GtkWidget *fileop_dialog;
  GtkWidget *fileop_entry;
  gchar     *fileop_file;
  gpointer   cmpl_state;

  GtkWidget *fileop_c_dir;
  GtkWidget *fileop_del_file;
  GtkWidget *fileop_ren_file;

  GtkWidget *button_area;
  GtkWidget *action_area;
} GtkFileSelection;
```

The GtkFileSelection struct contains the following GtkWidget fields:

*fileop_dialog;	the dialog box used to display the GtkFileSelection. It can be customized by adding/removing widgets from it using the standard GtkDialog functions.
*ok_button, *cancel_button;	the two main buttons that signals should be connected to in order to perform an action when the user hits either OK or Cancel.
*history_pulldown;	the GtkOptionMenu used to create the drop-down directory history.
*fileop_c_dir, *fileop_del_file, *fileop_ren_file;	the buttons that appear at the top of the file selection dialog. These "operation buttons" can be hidden and redisplayed with `gtk_file_selection_hide_fileop_buttons()` and `gtk_file_selection_show_fileop_buttons()` respectively.

---

gtk_file_selection_new ()

```
GtkWidget*  gtk_file_selection_new          (const gchar *title);
```

Creates a new file selection dialog box. By default it will contain a GtkTreeView of the application's current working directory, and a file listing. Operation buttons that allow the user to create a directory, delete files and rename files, are also present.

`title` : a message that will be placed in the file requestor's titlebar.

Returns : the new file selection.

---

gtk_file_selection_set_filename ()

```
void        gtk_file_selection_set_filename (GtkFileSelection *filesel,
                                             const gchar *filename);
```

Sets a default path for the file requestor. If *filename* includes a directory path, then the requestor will open with that path as its current working directory.

This has the consequence that in order to open the requestor with a working directory and an empty filename, *filename* must have a trailing directory separator.

The encoding of *filename* is the on-disk encoding, which may not be UTF-8. See g_filename_from_utf8().

filesel : a GtkFileSelection.

filename : a string to set as the default file name.

---

gtk_file_selection_get_filename ()

```
G_CONST_RETURN gchar* gtk_file_selection_get_filename
                                            (GtkFileSelection *filesel);
```

This function returns the selected filename in the on-disk encoding (see g_filename_from_utf8()), which may or may not be the same as that used by GTK+ (UTF-8). To convert to UTF-8, call g_filename_to_utf8(). The returned string points to a statically allocated buffer and should be copied if you plan to keep it around.

If no file is selected then the selected directory path is returned.

filesel : a GtkFileSelection

Returns : currently-selected filename in the on-disk encoding.

---

gtk_file_selection_complete ()

```
void        gtk_file_selection_complete     (GtkFileSelection *filesel,
                                             const gchar *pattern);
```

Will attempt to match *pattern* to a valid filenames or subdirectories in the current directory. If a match can be made, the matched filename will appear in the text entry field of the file selection dialog. If a partial match can be made, the

"Files" list will contain those file names which have been partially matched, and the "Folders" list those directories which have been partially matched.

filesel : a GtkFileSelection.

pattern : a string of characters which may or may not match any filenames in the current directory.

---

gtk_file_selection_show_fileop_buttons ()

```
void        gtk_file_selection_show_fileop_buttons
                                            (GtkFileSelection *filesel);
```

Shows the file operation buttons, if they have previously been hidden. The rest of the widgets in the dialog will be resized accordingly.

filesel : a GtkFileSelection.

---

gtk_file_selection_hide_fileop_buttons ()

```
void        gtk_file_selection_hide_fileop_buttons
                                            (GtkFileSelection *filesel);
```

Hides the file operation buttons that normally appear at the top of the dialog. Useful if you wish to create a custom file selector, based on GtkFileSelection.

filesel : a GtkFileSelection.

---

gtk_file_selection_get_selections ()

```
gchar**     gtk_file_selection_get_selections
                                            (GtkFileSelection *filesel);
```

Retrieves the list of file selections the user has made in the dialog box. This function is intended for use when the user can select multiple files in the file list. The first file in the list is equivalent to what `gtk_file_selection_get_filename()` would return.

The filenames are in the encoding of `g_filename_from_utf8()`, which may or may not be the same as that used by

GTK+ (UTF-8). To convert to UTF-8, call `g_filename_to_utf8()` on each string.

filesel : a GtkFileSelection

Returns : a newly-allocated NULL-terminated array of strings. Use `g_strfreev()` to free it.

---

gtk_file_selection_set_select_multiple ()

```
void        gtk_file_selection_set_select_multiple
                                            (GtkFileSelection *filesel,
                                             gboolean select_multiple);
```

Sets whether the user is allowed to select multiple files in the file list. Use `gtk_file_selection_get_selections()` to get the list of selected files.

filesel : a GtkFileSelection

select_multiple : whether or not the user is allowed to select multiple files in the file list.

---

gtk_file_selection_get_select_multiple ()

```
gboolean    gtk_file_selection_get_select_multiple
                                            (GtkFileSelection *filesel);
```

Determines whether or not the user is allowed to select multiple files in the file list. See `gtk_file_selection_set_select_multiple()`.

filesel : a GtkFileSelection

Returns : TRUE if the user is allowed to select multiple files in the file list

Properties

The "filename" property

```
  "filename"             gchararray            : Read / Write
```

The currently selected filename.

Default value: NULL

The "select-multiple" property

```
  "select-multiple"      gboolean              : Read / Write
```

Whether to allow multiple files to be selected.

Default value: FALSE

The "show-fileops" property

```
  "show-fileops"         gboolean              : Read / Write
```

Whether buttons for creating/manipulating files should be displayed.

Default value: FALSE

See Also

GtkDialog Add your own widgets into the GtkFileSelection.

<< GtkColorSelectionDialog **GtkFileChooser >>**

GtkFileChooser

GtkFileChooser — File chooser interface used by GtkFileChooserWidget and GtkFileChooserDialog

Synopsis

```
#include <gtk/gtk.h>


            GtkFileChooser;
enum        GtkFileChooserAction;
#define     GTK_FILE_CHOOSER_ERROR
enum        GtkFileChooserError;
GQuark      gtk_file_chooser_error_quark    (void);
void        gtk_file_chooser_set_action     (GtkFileChooser *chooser,
                                             GtkFileChooserAction action);
GtkFileChooserAction gtk_file_chooser_get_action
                                            (GtkFileChooser *chooser);
void        gtk_file_chooser_set_local_only (GtkFileChooser *chooser,
                                             gboolean local_only);
gboolean    gtk_file_chooser_get_local_only (GtkFileChooser *chooser);
void        gtk_file_chooser_set_select_multiple
                                            (GtkFileChooser *chooser,
                                             gboolean select_multiple);
gboolean    gtk_file_chooser_get_select_multiple
                                            (GtkFileChooser *chooser);
void        gtk_file_chooser_set_show_hidden
                                            (GtkFileChooser *chooser,
                                             gboolean show_hidden);
gboolean    gtk_file_chooser_get_show_hidden
                                            (GtkFileChooser *chooser);
void        gtk_file_chooser_set_current_name
                                            (GtkFileChooser *chooser,
                                             const gchar *name);
gchar*      gtk_file_chooser_get_filename   (GtkFileChooser *chooser);
```

```
gboolean    gtk_file_chooser_set_filename   (GtkFileChooser *chooser,
                                             const char *filename);
gboolean    gtk_file_chooser_select_filename
                                            (GtkFileChooser *chooser,
                                             const char *filename);
void        gtk_file_chooser_unselect_filename
                                            (GtkFileChooser *chooser,
                                             const char *filename);
void        gtk_file_chooser_select_all     (GtkFileChooser *chooser);
void        gtk_file_chooser_unselect_all   (GtkFileChooser *chooser);
GSList*     gtk_file_chooser_get_filenames  (GtkFileChooser *chooser);
gboolean    gtk_file_chooser_set_current_folder
                                            (GtkFileChooser *chooser,
                                             const gchar *filename);
gchar*      gtk_file_chooser_get_current_folder
                                            (GtkFileChooser *chooser);
gchar*      gtk_file_chooser_get_uri        (GtkFileChooser *chooser);
gboolean    gtk_file_chooser_set_uri        (GtkFileChooser *chooser,
                                             const char *uri);
gboolean    gtk_file_chooser_select_uri     (GtkFileChooser *chooser,
                                             const char *uri);
void        gtk_file_chooser_unselect_uri   (GtkFileChooser *chooser,
                                             const char *uri);
GSList*     gtk_file_chooser_get_uris       (GtkFileChooser *chooser);
gboolean    gtk_file_chooser_set_current_folder_uri
                                            (GtkFileChooser *chooser,
                                             const gchar *uri);
gchar*      gtk_file_chooser_get_current_folder_uri
                                            (GtkFileChooser *chooser);
void        gtk_file_chooser_set_preview_widget
                                            (GtkFileChooser *chooser,
                                             GtkWidget *preview_widget);
GtkWidget*  gtk_file_chooser_get_preview_widget
                                            (GtkFileChooser *chooser);
void        gtk_file_chooser_set_preview_widget_active
                                            (GtkFileChooser *chooser,
                                             gboolean active);
gboolean    gtk_file_chooser_get_preview_widget_active
                                            (GtkFileChooser *chooser);
void        gtk_file_chooser_set_use_preview_label
                                            (GtkFileChooser *chooser,
                                             gboolean use_label);
```

```
gboolean    gtk_file_chooser_get_use_preview_label
                                            (GtkFileChooser *chooser);
char*       gtk_file_chooser_get_preview_filename
                                            (GtkFileChooser *chooser);
char*       gtk_file_chooser_get_preview_uri
                                            (GtkFileChooser *chooser);
void        gtk_file_chooser_set_extra_widget
                                            (GtkFileChooser *chooser,
                                             GtkWidget *extra_widget);
GtkWidget*  gtk_file_chooser_get_extra_widget
                                            (GtkFileChooser *chooser);
void        gtk_file_chooser_add_filter     (GtkFileChooser *chooser,
                                             GtkFileFilter *filter);
void        gtk_file_chooser_remove_filter  (GtkFileChooser *chooser,
                                             GtkFileFilter *filter);
GSList*     gtk_file_chooser_list_filters   (GtkFileChooser *chooser);
void        gtk_file_chooser_set_filter     (GtkFileChooser *chooser,
                                             GtkFileFilter *filter);
GtkFileFilter* gtk_file_chooser_get_filter  (GtkFileChooser *chooser);
gboolean    gtk_file_chooser_add_shortcut_folder
                                            (GtkFileChooser *chooser,
                                             const char *folder,
                                             GError **error);
gboolean    gtk_file_chooser_remove_shortcut_folder
                                            (GtkFileChooser *chooser,
                                             const char *folder,
                                             GError **error);
GSList*     gtk_file_chooser_list_shortcut_folders
                                            (GtkFileChooser *chooser);
gboolean    gtk_file_chooser_add_shortcut_folder_uri
                                            (GtkFileChooser *chooser,
                                             const char *uri,
                                             GError **error);
gboolean    gtk_file_chooser_remove_shortcut_folder_uri
                                            (GtkFileChooser *chooser,
                                             const char *uri,
                                             GError **error);
GSList*     gtk_file_chooser_list_shortcut_folder_uris
                                            (GtkFileChooser *chooser);
```

Object Hierarchy

```
GInterface
 +----GtkFileChooser
```

Prerequisites

GtkFileChooser requires GtkWidget.

Known Implementations

GtkFileChooser is implemented by GtkFileChooserWidget, GtkFileChooserButton and GtkFileChooserDialog.

Properties

```
"action"               GtkFileChooserAction  : Read / Write
"extra-widget"         GtkWidget             : Read / Write
"file-system-backend"  gchararray            : Write / Construct Only
"filter"               GtkFileFilter         : Read / Write
"local-only"           gboolean              : Read / Write
"preview-widget"       GtkWidget             : Read / Write
"preview-widget-active" gboolean              : Read / Write
"select-multiple"      gboolean              : Read / Write
"show-hidden"          gboolean              : Read / Write
"use-preview-label"    gboolean              : Read / Write
```

Signal Prototypes

```
"current-folder-changed"
            void        user_function      (GtkFileChooser *chooser,
                                            gpointer user_data);
"file-activated"
            void        user_function      (GtkFileChooser *chooser,
```

```
                                                gpointer user_data);
"selection-changed"
            void        user_function      (GtkFileChooser *chooser,
                                                gpointer user_data);
"update-preview"
            void        user_function      (GtkFileChooser *chooser,
                                                gpointer user_data);
```

Description

GtkFileChooser is an interface that can be implemented by file selection widgets. In GTK+, the main objects that implement this interface are GtkFileChooserWidget, GtkFileChooserDialog, and GtkFileChooserButton. You do not need to write an object that implements the GtkFileChooser interface unless you are trying to adapt an existing file selector to expose a standard programming interface.

File Names and Encodings

When the user is finished selecting files in a GtkFileChooser, your program can get the selected names either as filenames or as URIs. For URIs, the normal escaping rules are applied if the URI contains non-ASCII characters. However, filenames are *always* returned in the character set specified by the `G_FILENAME_ENCODING` environment variable. Please see the Glib documentation for more details about this variable.

Important

This means that while you can pass the result of `gtk_file_chooser_get_filename()` to `open(2)` or `fopen(3)`, you may not be able to directly set it as the text of a GtkLabel widget unless you convert it first to UTF-8, which all GTK+ widgets expect. You should use `g_filename_to_utf8()` to convert filenames into strings that can be passed to GTK+ widgets.

Adding a Preview Widget

You can add a custom preview widget to a file chooser and then get notification about when the preview needs to be updated. To install a preview widget, use `gtk_file_chooser_set_preview_widget()`. Then, connect to the GtkFileChooser::update-preview signal to get notified when you need to update the contents of the preview.

Your callback should use `gtk_file_chooser_get_preview_filename()` to see what needs previewing. Once you have generated the preview for the corresponding file, you must call

gtk_file_chooser_set_preview_widget_active() with a boolean flag that indicates whether your callback could successfully generate a preview.

Example 2. Sample Usage

```
{
  GtkImage *preview;

  ...

  preview = gtk_image_new ();

  gtk_file_chooser_set_preview_widget (my_file_chooser, preview);
  g_signal_connect (my_file_chooser, "update-preview",
                    G_CALLBACK (update_preview_cb), preview);
}

static void
update_preview_cb (GtkFileChooser *file_chooser, gpointer data)
{
  GtkWidget *preview;
  char *filename;
  GdkPixbuf *pixbuf;
  gboolean have_preview;

  preview = GTK_WIDGET (data);
  filename = gtk_file_chooser_get_preview_filename (file_chooser);

  pixbuf = gdk_pixbuf_new_from_file_at_size (filename, 128, 128, NULL);
  have_preview = (pixbuf != NULL);
  g_free (filename);

  gtk_image_set_from_pixbuf (GTK_IMAGE (preview), pixbuf);
  if (pixbuf)
    gdk_pixbuf_unref (pixbuf);

  gtk_file_chooser_set_preview_widget_active (file_chooser, have_preview);
}
```

Adding Extra Widgets

You can add extra widgets to a file chooser to provide options that are not present in the default design. For

example, you can add a toggle button to give the user the option to open a file in read-only mode. You can use `gtk_file_chooser_set_extra_widget()` to insert additional widgets in a file chooser.

Example 3. Sample Usage

```
{
  GtkWidget *toggle;

  ...

  toggle = gtk_check_button_new_with_label ("Open file read-only");
  gtk_widget_show (toggle);
  gtk_file_chooser_set_extra_widget (my_file_chooser, toggle);
}
```

Note

If you want to set more than one extra widget in the file chooser, you can a container such as a GtkVBox or a GtkTable and include your widgets in it. Then, set the container as the whole extra widget.

Key Bindings

Internally, GTK+ implements a file chooser's graphical user interface with the private `GtkFileChooserDefaultClass`. This widget has several key bindings and their associated signals. This section describes the available key binding signals.

Example 4. GtkFileChooser key binding example

The default keys that activate the key-binding signals in `GtkFileChooserDefaultClass` are as follows:

Signal name	Key
location-popup	**Control-L**
up-folder	**Alt-Up**
down-folder	**Alt-Down**
home-folder	**Alt-Home**

To change these defaults to something else, you could include the following fragment in your `.gtkrc-2.0`

file:

```
binding "my-own-gtkfilechooser-bindings" {
        bind "<Alt><Shift>l" {
                "location-popup" ()
        }
        bind "<Alt><Shift>Up" {
                "up-folder" ()
        }
        bind "<Alt><Shift>Down" {
                "down-folder" ()
        }
        bind "<Alt><Shift>Home" {
                "home-folder-folder" ()
        }
}

class "GtkFileChooserDefault" binding "my-own-gtkfilechooser-bindings"
```

The "GtkFileChooserDefault::location-popup" signal

```
          void user_function (GtkFileChooserDefault *chooser,
                              gpointer user_data);
```

This is used to make the file chooser show a "Location" dialog which the user can use to manually type the name of the file he wishes to select. By default this is bound to **Control-L**.

chooser : the object which received the signal.

user_data : user data set when the signal handler was connected.

The "GtkFileChooserDefault::up-folder" signal

```
          void user_function (GtkFileChooserDefault *chooser,
                              gpointer user_data);
```

This is used to make the file chooser go to the parent of the current folder in the file hierarchy. By default this is bound to **Alt-Up**.

chooser : the object which received the signal.
user_data : user data set when the signal handler was connected.

The "GtkFileChooserDefault::down-folder" signal

```
void user_function (GtkFileChooserDefault *chooser,
                    gpointer user_data);
```

This is used to make the file chooser go to a child of the current folder in the file hierarchy. The subfolder that will be used is displayed in the path bar widget of the file chooser. For example, if the path bar is showing "/foo/*bar*/baz", then this will cause the file chooser to switch to the "baz" subfolder. By default this is bound to **Alt-Down**.

chooser : the object which received the signal.
user_data : user data set when the signal handler was connected.

The "GtkFileChooserDefault::home-folder" signal

```
void user_function (GtkFileChooserDefault *chooser,
                    gpointer user_data);
```

This is used to make the file chooser show the user's home folder in the file list. By default this is bound to **Alt-Home**.

chooser : the object which received the signal.
user_data : user data set when the signal handler was connected.

Details

GtkFileChooser

```
typedef struct _GtkFileChooser GtkFileChooser;
```

---

enum GtkFileChooserAction

```
typedef enum
{
  GTK_FILE_CHOOSER_ACTION_OPEN,
  GTK_FILE_CHOOSER_ACTION_SAVE,
  GTK_FILE_CHOOSER_ACTION_SELECT_FOLDER,
  GTK_FILE_CHOOSER_ACTION_CREATE_FOLDER
} GtkFileChooserAction;
```

Describes whether a GtkFileChooser is being used to open existing files or to save to a possibly new file.

GTK_FILE_CHOOSER_ACTION_OPEN	Indicates open mode. The file chooser will only let the user pick an existing file.
GTK_FILE_CHOOSER_ACTION_SAVE	Indicates save mode. The file chooser will let the user pick an existing file, or type in a new filename.
GTK_FILE_CHOOSER_ACTION_SELECT_FOLDER	Indicates an Open mode for selecting folders. The file chooser will let the user pick an existing folder.
GTK_FILE_CHOOSER_ACTION_CREATE_FOLDER	Indicates a mode for creating a new folder. The file chooser will let the user name an existing or new folder.

GTK_FILE_CHOOSER_ERROR

```
#define GTK_FILE_CHOOSER_ERROR (gtk_file_chooser_error_quark ())
```

Used to get the GError quark for GtkFileChooser errors.

---

enum GtkFileChooserError

```
typedef enum {
  GTK_FILE_CHOOSER_ERROR_NONEXISTENT,
  GTK_FILE_CHOOSER_ERROR_BAD_FILENAME
```

```
} GtkFileChooserError;
```

These identify the various errors that can occur while calling GtkFileChooser functions.

`GTK_FILE_CHOOSER_ERROR_NONEXISTENT`	Indicates that a file does not exist.
`GTK_FILE_CHOOSER_ERROR_BAD_FILENAME`	Indicates a malformed filename.

---

gtk_file_chooser_error_quark ()

```
GQuark      gtk_file_chooser_error_quark    (void);
```

Registers an error quark for GtkFileChooser if necessary.

Returns : The error quark used for GtkFileChooser errors.

Since 2.4

---

gtk_file_chooser_set_action ()

```
void        gtk_file_chooser_set_action     (GtkFileChooser *chooser,
                                             GtkFileChooserAction action);
```

Sets the type of operation that that the chooser is performing; the user interface is adapted to suit the selected action. For example, an option to create a new folder might be shown if the action is `GTK_FILE_CHOOSER_ACTION_SAVE` but not if the action is `GTK_FILE_CHOOSER_ACTION_OPEN`.

`chooser` : a GtkFileChooser

`action` : the action that the file selector is performing

Since 2.4

---

gtk_file_chooser_get_action ()

```
GtkFileChooserAction gtk_file_chooser_get_action
                                                (GtkFileChooser *chooser);
```

Gets the type of operation that the file chooser is performing; see gtk_file_chooser_set_action().

chooser : a GtkFileChooser

Returns : the action that the file selector is performing

Since 2.4

---

gtk_file_chooser_set_local_only ()

```
void        gtk_file_chooser_set_local_only (GtkFileChooser *chooser,
                                             gboolean local_only);
```

Sets whether only local files can be selected in the file selector. If *local_only* is TRUE (the default), then the selected file are files are guaranteed to be accessible through the operating systems native file file system and therefore the application only needs to worry about the filename functions in GtkFileChooser, like gtk_file_chooser_get_filename(), rather than the URI functions like gtk_file_chooser_get_uri(),

chooser : a GtkFileChooser

local_only : TRUE if only local files can be selected

Since 2.4

---

gtk_file_chooser_get_local_only ()

```
gboolean    gtk_file_chooser_get_local_only (GtkFileChooser *chooser);
```

Gets whether only local files can be selected in the file selector. See gtk_file_chooser_set_local_only()

chooser : a GtkFileChoosre

Returns : TRUE if only local files can be selected.

Since 2.4

---

gtk_file_chooser_set_select_multiple ()

```
void        gtk_file_chooser_set_select_multiple
                                            (GtkFileChooser *chooser,
                                             gboolean select_multiple);
```

Sets whether multiple files can be selected in the file selector. This is only relevant if the action is set to be GTK_FILE_CHOOSER_ACTION_OPEN or GTK_FILE_CHOOSER_ACTION_SAVE. It cannot be set with either of the folder actions.

chooser : a GtkFileChooser

select_multiple : TRUE if multiple files can be selected.

Since 2.4

---

gtk_file_chooser_get_select_multiple ()

```
gboolean    gtk_file_chooser_get_select_multiple
                                            (GtkFileChooser *chooser);
```

Gets whether multiple files can be selected in the file selector. See gtk_file_chooser_set_select_multiple().

chooser : a GtkFileChooser

Returns : TRUE if multiple files can be selected.

Since 2.4

---

gtk_file_chooser_set_show_hidden ()

```
void        gtk_file_chooser_set_show_hidden
                                            (GtkFileChooser *chooser,
                                             gboolean show_hidden);
```

Sets whether hidden files and folders are displayed in the file selector.

chooser : a GtkFileChooser

show_hidden : TRUE if hidden files and folders should be displayed.

Since 2.6

---

gtk_file_chooser_get_show_hidden ()

```
gboolean    gtk_file_chooser_get_show_hidden
                                            (GtkFileChooser *chooser);
```

Gets whether hidden files and folders are displayed in the file selector. See gtk_file_chooser_set_show_hidden().

chooser : a GtkFileChooser

Returns : TRUE if hidden files and folders are displayed.

Since 2.6

---

gtk_file_chooser_set_current_name ()

```
void        gtk_file_chooser_set_current_name
                                            (GtkFileChooser *chooser,
                                             const gchar *name);
```

Sets the current name in the file selector, as if entered by the user. Note that the name passed in here is a UTF-8 string rather than a filename. This function is meant for such uses as a suggested name in a "Save As..." dialog.

If you want to preselect a particular existing file, you should use `gtk_file_chooser_set_filename()` instead.

chooser : a GtkFileChooser

name : the filename to use, as a UTF-8 string

Since 2.4

---

gtk_file_chooser_get_filename ()

```
gchar*      gtk_file_chooser_get_filename   (GtkFileChooser *chooser);
```

Gets the filename for the currently selected file in the file selector. If multiple files are selected, one of the filenames will be returned at random.

If the file chooser is in folder mode, this function returns the selected folder.

chooser : a GtkFileChooser

Returns : The currently selected filename, or `NULL` if no file is selected, or the selected file can't be represented with a local filename. Free with `g_free()`.

Since 2.4

---

gtk_file_chooser_set_filename ()

```
gboolean    gtk_file_chooser_set_filename   (GtkFileChooser *chooser,
                                             const char *filename);
```

Sets *filename* as the current filename for the the file chooser; If the file name isn't in the current folder of *chooser*, then the current folder of *chooser* will be changed to the folder containing *filename*. This is equivalent to a sequence of gtk_file_chooser_unselect_all() followed by gtk_file_chooser_select_filename().

Note that the file must exist, or nothing will be done except for the directory change. To pre-enter a filename for the user, as in a save-as dialog, use gtk_file_chooser_set_current_name()

chooser :	a GtkFileChooser
filename :	the filename to set as current
Returns :	TRUE if both the folder could be changed and the file was selected successfully, FALSE otherwise.

Since 2.4

---

gtk_file_chooser_select_filename ()

```
gboolean    gtk_file_chooser_select_filename
                                        (GtkFileChooser *chooser,
                                         const char *filename);
```

Selects a filename. If the file name isn't in the current folder of *chooser*, then the current folder of *chooser* will be changed to the folder containing *filename*.

chooser :	a GtkFileChooser
filename :	the filename to select
Returns :	TRUE if both the folder could be changed and the file was selected successfully, FALSE otherwise.

Since 2.4

---

gtk_file_chooser_unselect_filename ()

```
void        gtk_file_chooser_unselect_filename
                                            (GtkFileChooser *chooser,
                                             const char *filename);
```

Unselects a currently selected filename. If the filename is not in the current directory, does not exist, or is otherwise not currently selected, does nothing.

chooser : a GtkFileChooser

filename : the filename to unselect

Since 2.4

---

gtk_file_chooser_select_all ()

```
void        gtk_file_chooser_select_all     (GtkFileChooser *chooser);
```

Selects all the files in the current folder of a file chooser.

chooser : a GtkFileChooser

Since 2.4

---

gtk_file_chooser_unselect_all ()

```
void        gtk_file_chooser_unselect_all   (GtkFileChooser *chooser);
```

Unselects all the files in the current folder of a file chooser.

chooser : a GtkFileChooser

Since 2.4

---

gtk_file_chooser_get_filenames ()

```
GSList*     gtk_file_chooser_get_filenames  (GtkFileChooser *chooser);
```

Lists all the selected files and subfolders in the current folder of *chooser*. The returned names are full absolute paths. If files in the current folder cannot be represented as local filenames they will be ignored. (See gtk_file_chooser_get_uris())

chooser : a GtkFileChooser

Returns : a GSList containing the filenames of all selected files and subfolders in the current folder. Free the returned list with g_slist_free(), and the filenames with g_free().

Since 2.4

---

gtk_file_chooser_set_current_folder ()

```
gboolean    gtk_file_chooser_set_current_folder
                                            (GtkFileChooser *chooser,
                                             const gchar *filename);
```

Sets the current folder for *chooser* from a local filename. The user will be shown the full contents of the current folder, plus user interface elements for navigating to other folders.

chooser : a GtkFileChooser

filename : the full path of the new current folder

Returns : TRUE if the folder could be changed successfully, FALSE otherwise.

Since 2.4

---

gtk_file_chooser_get_current_folder ()

```
gchar*      gtk_file_chooser_get_current_folder
                                            (GtkFileChooser *chooser);
```

Gets the current folder of *chooser* as a local filename. See gtk_file_chooser_set_current_folder().

chooser : a GtkFileChooser

Returns : the full path of the current folder, or NULL if the current path cannot be represented as a local filename. Free with g_free().

Since 2.4

---

gtk_file_chooser_get_uri ()

```
gchar*      gtk_file_chooser_get_uri        (GtkFileChooser *chooser);
```

Gets the URI for the currently selected file in the file selector. If multiple files are selected, one of the filenames will be returned at random.

If the file chooser is in folder mode, this function returns the selected folder.

chooser : a GtkFileChooser

Returns : The currently selected URI, or NULL if no file is selected. Free with g_free()

Since 2.4

---

gtk_file_chooser_set_uri ()

```
gboolean    gtk_file_chooser_set_uri        (GtkFileChooser *chooser,
                                             const char *uri);
```

Sets the file referred to by *uri* as the current file for the the file chooser; If the file name isn't in the current folder of *chooser*, then the current folder of *chooser* will be changed to the folder containing *uri*. This is

equivalent to a sequence of `gtk_file_chooser_unselect_all()` followed by `gtk_file_chooser_select_uri()`.

Note that the file must exist, or nothing will be done except for the directory change. To pre-enter a filename for the user, as in a save-as dialog, use `gtk_file_chooser_set_current_name()`

chooser :	a GtkFileChooser
uri :	the URI to set as current
Returns :	TRUE if both the folder could be changed and the URI was selected successfully, FALSE otherwise.

Since 2.4

---

gtk_file_chooser_select_uri ()

```
gboolean    gtk_file_chooser_select_uri     (GtkFileChooser *chooser,
                                             const char *uri);
```

Selects the file to by *uri*. If the URI doesn't refer to a file in the current folder of *chooser*, then the current folder of *chooser* will be changed to the folder containing *filename*.

chooser :	a GtkFileChooser
uri :	the URI to select
Returns :	TRUE if both the folder could be changed and the URI was selected successfully, FALSE otherwise.

Since 2.4

---

gtk_file_chooser_unselect_uri ()

```
void        gtk_file_chooser_unselect_uri   (GtkFileChooser *chooser,
                                             const char *uri);
```

Unselects the file referred to by *uri*. If the file is not in the current directory, does not exist, or is otherwise not

currently selected, does nothing.

chooser : a GtkFileChooser
uri : the URI to unselect

Since 2.4

---

gtk_file_chooser_get_uris ()

```
GSList*     gtk_file_chooser_get_uris       (GtkFileChooser *chooser);
```

Lists all the selected files and subfolders in the current folder of *chooser*. The returned names are full absolute URIs.

chooser : a GtkFileChooser
Returns : a GSList containing the URIs of all selected files and subfolders in the current folder. Free the returned list with `g_slist_free()`, and the filenames with `g_free()`.

Since 2.4

---

gtk_file_chooser_set_current_folder_uri ()

```
gboolean    gtk_file_chooser_set_current_folder_uri
                                        (GtkFileChooser *chooser,
                                         const gchar *uri);
```

Sets the current folder for *chooser* from an URI. The user will be shown the full contents of the current folder, plus user interface elements for navigating to other folders.

chooser : a GtkFileChooser
uri : the URI for the new current folder
Returns : TRUE if the folder could be changed successfully, FALSE otherwise.

Since 2.4

---

gtk_file_chooser_get_current_folder_uri ()

```
gchar*      gtk_file_chooser_get_current_folder_uri
                                            (GtkFileChooser *chooser);
```

Gets the current folder of *chooser* as an URI. See gtk_file_chooser_set_current_folder_uri().

chooser : a GtkFileChooser

Returns : the URI for the current folder. Free with g_free().

Since 2.4

---

gtk_file_chooser_set_preview_widget ()

```
void        gtk_file_chooser_set_preview_widget
                                            (GtkFileChooser *chooser,
                                             GtkWidget *preview_widget);
```

Sets an application-supplied widget to use to display a custom preview of the currently selected file. To implement a preview, after setting the preview widget, you connect to the ::update-preview signal, and call gtk_file_chooser_get_preview_filename() or gtk_file_chooser_get_preview_uri() on each change. If you can display a preview of the new file, update your widget and set the preview active using gtk_file_chooser_set_preview_widget_active(). Otherwise, set the preview inactive.

When there is no application-supplied preview widget, or the application-supplied preview widget is not active, the file chooser may display an internally generated preview of the current file or it may display no preview at all.

chooser : a GtkFileChooser

preview_widget : widget for displaying preview.

Since 2.4

---

gtk_file_chooser_get_preview_widget ()

```
GtkWidget*  gtk_file_chooser_get_preview_widget
                                        (GtkFileChooser *chooser);
```

Gets the current preview widget; see gtk_file_chooser_set_preview_widget().

chooser : a GtkFileChooser

Returns : the current preview widget, or NULL

Since 2.4

---

gtk_file_chooser_set_preview_widget_active ()

```
void        gtk_file_chooser_set_preview_widget_active
                                        (GtkFileChooser *chooser,
                                         gboolean active);
```

Sets whether the preview widget set by gtk_file_chooser_set_preview_widget_active() should be shown for the current filename. When *active* is set to false, the file chooser may display an internally generated preview of the current file or it may display no preview at all. See gtk_file_chooser_set_preview_widget() for more details.

chooser : a GtkFileChooser

active : whether to display the user-specified preview widget

Since 2.4

---

gtk_file_chooser_get_preview_widget_active ()

```
gboolean    gtk_file_chooser_get_preview_widget_active
                                            (GtkFileChooser *chooser);
```

Gets whether the preview widget set by `gtk_file_chooser_set_preview_widget()` should be shown for the current filename. See `gtk_file_chooser_set_preview_widget_active()`.

chooser : a GtkFileChooser

Returns : TRUE if the preview widget is active for the current filename.

Since 2.4

---

gtk_file_chooser_set_use_preview_label ()

```
void        gtk_file_chooser_set_use_preview_label
                                            (GtkFileChooser *chooser,
                                             gboolean use_label);
```

Sets whether the file chooser should display a stock label with the name of the file that is being previewed; the default is TRUE. Applications that want to draw the whole preview area themselves should set this to FALSE and display the name themselves in their preview widget.

See also: `gtk_file_chooser_set_preview_widget()`

chooser : a GtkFileChooser

use_label : whether to display a stock label with the name of the previewed file

Since 2.4

---

gtk_file_chooser_get_use_preview_label ()

```
gboolean    gtk_file_chooser_get_use_preview_label
                                            (GtkFileChooser *chooser);
```

Gets whether a stock label should be drawn with the name of the previewed file. See `gtk_file_chooser_set_use_preview_label()`.

chooser : a GtkFileChooser

Returns : TRUE if the file chooser is set to display a label with the name of the previewed file, FALSE otherwise.

---

gtk_file_chooser_get_preview_filename ()

```
char*       gtk_file_chooser_get_preview_filename
                                            (GtkFileChooser *chooser);
```

Gets the filename that should be previewed in a custom preview widget. See `gtk_file_chooser_set_preview_widget()`.

chooser : a GtkFileChooser

Returns : the filename to preview, or NULL if no file is selected, or if the selected file cannot be represented as a local filename. Free with `g_free()`

Since 2.4

---

gtk_file_chooser_get_preview_uri ()

```
char*       gtk_file_chooser_get_preview_uri
                                            (GtkFileChooser *chooser);
```

Gets the URI that should be previewed in a custom preview widget. See `gtk_file_chooser_set_preview_widget()`.

chooser : a GtkFileChooser

Returns : the URI for the file to preview, or NULL if no file is selected. Free with `g_free()`.

Since 2.4

gtk_file_chooser_set_extra_widget ()

```
void        gtk_file_chooser_set_extra_widget
                                            (GtkFileChooser *chooser,
                                             GtkWidget *extra_widget);
```

Sets an application-supplied widget to provide extra options to the user.

chooser : a GtkFileChooser
extra_widget : widget for extra options

Since 2.4

gtk_file_chooser_get_extra_widget ()

```
GtkWidget*  gtk_file_chooser_get_extra_widget
                                            (GtkFileChooser *chooser);
```

Gets the current preview widget; see gtk_file_chooser_set_extra_widget().

chooser : a GtkFileChooser
Returns : the current extra widget, or NULL

Since 2.4

gtk_file_chooser_add_filter ()

```
void        gtk_file_chooser_add_filter     (GtkFileChooser *chooser,
                                             GtkFileFilter *filter);
```

Adds *filter* to the list of filters that the user can select between. When a filter is selected, only files that are passed by that filter are displayed.

chooser : a GtkFileChooser
filter : a GtkFileFilter

Since 2.4

---

gtk_file_chooser_remove_filter ()

```
void        gtk_file_chooser_remove_filter  (GtkFileChooser *chooser,
                                             GtkFileFilter *filter);
```

Removes *filter* from the list of filters that the user can select between.

chooser : a GtkFileChooser
filter : a GtkFileFilter

Since 2.4

---

gtk_file_chooser_list_filters ()

```
GSList*     gtk_file_chooser_list_filters   (GtkFileChooser *chooser);
```

Lists the current set of user-selectable filters; see gtk_file_chooser_add_filter(), gtk_file_chooser_remove_filter().

chooser : a GtkFileChooser

Returns : a GSList containing the current set of user selectable filters. The contents of the list are owned by GTK+, but you must free the list itself with g_slist_free() when you are done with it.

Since 2.4

---

gtk_file_chooser_set_filter ()

```
void        gtk_file_chooser_set_filter     (GtkFileChooser *chooser,
                                             GtkFileFilter *filter);
```

Sets the current filter; only the files that pass the filter will be displayed. If the user-selectable list of filters is non-empty, then the filter should be one of the filters in that list. Setting the current filter when the list of filters is empty is useful if you want to restrict the displayed set of files without letting the user change it.

chooser : a GtkFileChooser
filter : a GtkFileFilter

Since 2.4

---

gtk_file_chooser_get_filter ()

```
GtkFileFilter* gtk_file_chooser_get_filter  (GtkFileChooser *chooser);
```

Gets the current filter; see gtk_file_chooser_set_filter().

chooser : a GtkFileChooser
Returns : the current filter, or NULL

Since 2.4

---

gtk_file_chooser_add_shortcut_folder ()

```
gboolean    gtk_file_chooser_add_shortcut_folder
                                            (GtkFileChooser *chooser,
```

```
                                                const char *folder,
                                                GError **error);
```

Adds a folder to be displayed with the shortcut folders in a file chooser. Note that shortcut folders do not get saved, as they are provided by the application. For example, you can use this to add a "/usr/share/mydrawprogram/Clipart" folder to the volume list.

chooser :	a GtkFileChooser
folder :	filename of the folder to add
error :	location to store error, or NULL
Returns :	TRUE if the folder could be added successfully, FALSE otherwise. In the latter case, the *error* will be set as appropriate.

Since 2.4

---

gtk_file_chooser_remove_shortcut_folder ()

```
gboolean    gtk_file_chooser_remove_shortcut_folder
                                                (GtkFileChooser *chooser,
                                                 const char *folder,
                                                 GError **error);
```

Removes a folder from a file chooser's list of shortcut folders.

chooser :	a GtkFileChooser
folder :	filename of the folder to remove
error :	location to store error, or NULL
Returns :	TRUE if the operation succeeds, FALSE otherwise. In the latter case, the *error* will be set as appropriate. See also: gtk_file_chooser_add_shortcut_folder()

Since 2.4

---

gtk_file_chooser_list_shortcut_folders ()

```
GSList*     gtk_file_chooser_list_shortcut_folders
                                                (GtkFileChooser *chooser);
```

Queries the list of shortcut folders in the file chooser, as set by `gtk_file_chooser_add_shortcut_folder()`.

chooser : a GtkFileChooser

Returns : A list of folder filenames, or NULL if there are no shortcut folders. Free the returned list with `g_slist_free()`, and the filenames with `g_free()`.

Since 2.4

---

gtk_file_chooser_add_shortcut_folder_uri ()

```
gboolean    gtk_file_chooser_add_shortcut_folder_uri
                                                (GtkFileChooser *chooser,
                                                 const char *uri,
                                                 GError **error);
```

Adds a folder URI to be displayed with the shortcut folders in a file chooser. Note that shortcut folders do not get saved, as they are provided by the application. For example, you can use this to add a "file:///usr/share/mydrawprogram/Clipart" folder to the volume list.

chooser : a GtkFileChooser

uri : URI of the folder to add

error : location to store error, or NULL

Returns : TRUE if the folder could be added successfully, FALSE otherwise. In the latter case, the *error* will be set as appropriate.

Since 2.4

---

gtk_file_chooser_remove_shortcut_folder_uri ()

```
gboolean    gtk_file_chooser_remove_shortcut_folder_uri
                                            (GtkFileChooser *chooser,
                                             const char *uri,
                                             GError **error);
```

Removes a folder URI from a file chooser's list of shortcut folders.

chooser :	a GtkFileChooser
uri :	URI of the folder to remove
error :	location to store error, or NULL
Returns :	TRUE if the operation succeeds, FALSE otherwise. In the latter case, the *error* will be set as appropriate. See also: gtk_file_chooser_add_shortcut_folder_uri()

Since 2.4

---

gtk_file_chooser_list_shortcut_folder_uris ()

```
GSList*     gtk_file_chooser_list_shortcut_folder_uris
                                            (GtkFileChooser *chooser);
```

Queries the list of shortcut folders in the file chooser, as set by gtk_file_chooser_add_shortcut_folder_uri().

chooser :	a GtkFileChooser
Returns :	A list of folder URIs, or NULL if there are no shortcut folders. Free the returned list with g_slist_free(), and the URIs with g_free().

Since 2.4

Properties

The "action" property

```
```

```
  "action"                GtkFileChooserAction  : Read / Write
```

The type of operation that the file selector is performing.

Default value: GTK_FILE_CHOOSER_ACTION_OPEN

The "extra-widget" property

```
  "extra-widget"          GtkWidget             : Read / Write
```

Application supplied widget for extra options.

The "file-system-backend" property

```
  "file-system-backend"   gchararray            : Write / Construct Only
```

Name of file system backend to use.

Default value: NULL

The "filter" property

```
  "filter"                GtkFileFilter         : Read / Write
```

The current filter for selecting which files are displayed.

The "local-only" property

```
  "local-only"            gboolean              : Read / Write
```

Whether the selected file(s) should be limited to local file: URLs.

Default value: TRUE

The "preview-widget" property

```
  "preview-widget"       GtkWidget             : Read / Write
```

Application supplied widget for custom previews.

The "preview-widget-active" property

```
  "preview-widget-active" gboolean              : Read / Write
```

Whether the application supplied widget for custom previews should be shown.

Default value: TRUE

The "select-multiple" property

```
  "select-multiple"      gboolean              : Read / Write
```

Whether to allow multiple files to be selected.

Default value: FALSE

The "show-hidden" property

```
  "show-hidden"          gboolean              : Read / Write
```

Whether the hidden files and folders should be displayed.

Default value: FALSE

The "use-preview-label" property

```
  "use-preview-label"    gboolean              : Read / Write
```

Whether to display a stock label with the name of the previewed file.

Default value: TRUE

Signals

The "current-folder-changed" signal

```
void        user_function                    (GtkFileChooser *chooser,
                                              gpointer user_data);
```

This signal is emitted when the current folder in a GtkFileChooser changes. This can happen due to the user performing some action that changes folders, such as selecting a bookmark or visiting a folder on the file list. It can also happen as a result of calling a function to explicitly change the current folder in a file chooser.

Normally you do not need to connect to this signal, unless you need to keep track of which folder a file chooser is showing.

See also: gtk_file_chooser_set_current_folder(), gtk_file_chooser_get_current_folder(), gtk_file_chooser_set_current_folder_uri(), gtk_file_chooser_get_current_folder_uri().

chooser : the object which received the signal.
user_data : user data set when the signal handler was connected.

The "file-activated" signal

```
void        user_function                   (GtkFileChooser *chooser,
                                             gpointer user_data);
```

This signal is emitted when the user "activates" a file in the file chooser. This can happen by double-clicking on a file in the file list, or by pressing **Enter**.

Normally you do not need to connect to this signal. It is used internally by GtkFileChooserDialog to know when to activate the default button in the dialog.

See also: `gtk_file_chooser_get_filename()`, `gtk_file_chooser_get_filenames()`, `gtk_file_chooser_get_uri()`, `gtk_file_chooser_get_uris()`.

chooser : the object which received the signal.
user_data : user data set when the signal handler was connected.

The "selection-changed" signal

```
void        user_function                   (GtkFileChooser *chooser,
                                             gpointer user_data);
```

This signal is emitted when there is a change in the set of selected files in a GtkFileChooser. This can happen when the user modifies the selection with the mouse or the keyboard, or when explicitly calling functions to change the selection.

Normally you do not need to connect to this signal, as it is easier to wait for the file chooser to finish running, and then to get the list of selected files using the functions mentioned below.

See also: `gtk_file_chooser_select_filename()`, `gtk_file_chooser_unselect_filename()`, `gtk_file_chooser_get_filename()`, `gtk_file_chooser_get_filenames()`, `gtk_file_chooser_select_uri()`, `gtk_file_chooser_unselect_uri()`, `gtk_file_chooser_get_uri()`, `gtk_file_chooser_get_uris()`.

chooser : the object which received the signal.

user_data : user data set when the signal handler was connected.

The "update-preview" signal

```
void        user_function                   (GtkFileChooser *chooser,
                                             gpointer user_data);
```

This signal is emitted when the preview in a file chooser should be regenerated. For example, this can happen when the currently selected file changes. You should use this signal if you want your file chooser to have a preview widget.

Once you have installed a preview widget with `gtk_file_chooser_set_preview_widget()`, you should update it when this signal is emitted. You can use the functions `gtk_file_chooser_get_preview_filename()` or `gtk_file_chooser_get_preview_uri()` to get the name of the file to preview. Your widget may not be able to preview all kinds of files; your callback must call `gtk_file_chooser_set_preview_wiget_active()` to inform the file chooser about whether the preview was generated successfully or not.

Please see the example code in the section called “Adding a Preview Widget”.

See also: `gtk_file_chooser_set_preview_widget()`, `gtk_file_chooser_set_preview_widget_active()`, `gtk_file_chooser_set_use_preview_label()`, `gtk_file_chooser_get_preview_filename()`, `gtk_file_chooser_get_preview_uri()`.

chooser : the object which received the signal.
user_data : user data set when the signal handler was connected.

See Also

GtkFileChooserDialog, GtkFileChooserWidget, GtkFileChooserButton

GtkFileChooserButton

GtkFileChooserButton — A button to launch a file selection dialog

Synopsis

```
#include <gtk/gtk.h>

            GtkFileChooserButton;
GtkWidget*  gtk_file_chooser_button_new     (const gchar *title);
GtkWidget*  gtk_file_chooser_button_new_with_backend
                                            (const gchar *title,
                                             const gchar *backend);
GtkWidget*  gtk_file_chooser_button_new_with_dialog
                                            (GtkWidget *dialog);
G_CONST_RETURN gchar* gtk_file_chooser_button_get_title
                                            (GtkFileChooserButton *button);
void        gtk_file_chooser_button_set_title
                                            (GtkFileChooserButton *button,
                                             const gchar *title);
gboolean    gtk_file_chooser_button_get_active
                                            (GtkFileChooserButton *button);
void        gtk_file_chooser_button_set_active
                                            (GtkFileChooserButton *button,
                                             gboolean is_active);
gint        gtk_file_chooser_button_get_width_chars
                                            (GtkFileChooserButton *button);
void        gtk_file_chooser_button_set_width_chars
                                            (GtkFileChooserButton *button,
                                             gint n_chars);
```

Object Hierarchy

```
GObject
 +----GtkObject
       +----GtkWidget
             +----GtkContainer
                   +----GtkBox
                         +----GtkHBox
                               +----GtkFileChooserButton
```

Implemented Interfaces

GtkFileChooserButton implements AtkImplementorIface and GtkFileChooser.

Properties

```
"active"                gboolean              : Read / Write
"dialog"                GtkFileChooserDialog  : Write / Construct Only
"title"                 gchararray            : Read / Write
"width-chars"           gint                  : Read / Write
```

Description

The GtkFileChooserButton is a widget that lets the user select a file. It implements the GtkFileChooser interface. Visually, it is a file name with a button to bring up a GtkFileChooserDialog. The user can then use that dialog to change the file associated with that button. This widget does not support setting the "select-multiple" property to TRUE.

Example 5. Create a button to let the user select a file in /etc

```
{
  GtkWidget *button;

  button = gtk_file_chooser_button_new (_("Select a file"));
  gtk_file_chooser_set_current_folder (GTK_FILE_CHOOSER (button),
                                       "/etc");
}
```

The GtkFileChooserButton supports all four GtkFileChooserActions that the GtkFileChooser supports. Two of the actions, GTK_FILE_CHOOSER_ACTION_SAVE and GTK_FILE_CHOOSER_ACTION_CREATE_FOLDER, give the button the appearance of an entry next to a button. The user can type the name of a file in this entry, and it will complete as it types. The other two actions, GTK_FILE_CHOOSER_ACTION_OPEN and GTK_FILE_CHOOSER_ACTION_SELECT_FOLDER, make the GtkFileChooserAction look like a GtkButton.

Example 6. Using GtkFileChooserButton in save mode

```
{
  GtkWidget *button;

  button = gtk_file_chooser_button_new (_("Save as..."));
  gtk_file_chooser_set_action (GTK_FILE_CHOOSER (button),
                               GTK_FILE_CHOOSER_ACTION_SAVE);
  gtk_file_chooser_set_current_folder (GTK_FILE_CHOOSER_BUTTON (button),
                                       DEFAULT_SAVE_DIRECTORY);
}
```

Important

The GtkFileChooserButton will ellipsize the label while in Open mode, and thus will thus request little horizontal space. To give the button more space, you should call gtk_widget_size_request(), gtk_file_chooser_button_set_width_chars(), or pack the button in such a way that other interface elements give space to the widget.

Details

GtkFileChooserButton

```
typedef struct _GtkFileChooserButton GtkFileChooserButton;
```

This should not be accessed directly. Use the accessor functions below.

---

gtk_file_chooser_button_new ()

```
GtkWidget*  gtk_file_chooser_button_new     (const gchar *title);
```

Creates a new file-selecting button widget.

title : the title of the browse dialog.
Returns : a new button widget.

Since 2.6

---

gtk_file_chooser_button_new_with_backend ()

```
GtkWidget*  gtk_file_chooser_button_new_with_backend
                                                (const gchar *title,
                                                 const gchar *backend);
```

Creates a new file-selecting button widget using *backend*.

title : the title of the browse dialog.
backend : the name of the GtkFileSystem backend to use.
Returns : a new button widget.

Since 2.6

---

gtk_file_chooser_button_new_with_dialog ()

```
GtkWidget*  gtk_file_chooser_button_new_with_dialog
                                                (GtkWidget *dialog);
```

Creates a GtkFileChooserButton widget which uses *dialog* as it's file-picking window. Note that *dialog* must be a GtkFileChooserDialog (or subclass).

dialog : the GtkDialog widget to use.
Returns : a new button widget.

Since 2.6

---

gtk_file_chooser_button_get_title ()

```
G_CONST_RETURN gchar* gtk_file_chooser_button_get_title
                                                (GtkFileChooserButton *button);
```

Retrieves the title of the browse dialog used by *button*. The returned value should not be modified or freed.

button : the button widget to examine.
Returns : a pointer to the browse dialog's title.

Since 2.6

---

gtk_file_chooser_button_set_title ()

```
void        gtk_file_chooser_button_set_title
                                                (GtkFileChooserButton *button,
                                                 const gchar *title);
```

Modifies the *title* of the browse dialog used by *button*.

button : the button widget to modify.
title : the new browse dialog title.

Since 2.6

---

gtk_file_chooser_button_get_active ()

```
gboolean    gtk_file_chooser_button_get_active
                                                (GtkFileChooserButton *button);
```

Retrieves whether or not the dialog attached to *button* is visible.

button : the button widget to examine.
Returns : a boolean whether the dialog is visible or not.

Since 2.6

---

gtk_file_chooser_button_set_active ()

```
void        gtk_file_chooser_button_set_active
                                            (GtkFileChooserButton *button,
                                             gboolean is_active);
```

Modifies whether or not the dialog attached to *button* is visible or not.

button : the button widget to modify.
is_active : whether or not the dialog is visible.

Since 2.6

---

gtk_file_chooser_button_get_width_chars ()

```
gint        gtk_file_chooser_button_get_width_chars
                                            (GtkFileChooserButton *button);
```

Retrieves the width in characters of the *button* widget's entry and/or label.

button : the button widget to examine.
Returns : an integer width (in characters) that the button will use to size itself.

Since 2.6

---

gtk_file_chooser_button_set_width_chars ()

```
void        gtk_file_chooser_button_set_width_chars
                                            (GtkFileChooserButton *button,
                                             gint n_chars);
```

Sets the width (in characters) that *button* will use to *n_chars*.

button : the button widget to examine.
n_chars : the new width, in chracters.

Since 2.6

Properties

The "active" property

```
  "active"               gboolean              : Read / Write
```

TRUE, if the GtkFileChooserDialog associated with the button has been made visible. This can also be set by the application, though it is rarely useful to do so.

Default value: FALSE

The "dialog" property

```
  "dialog"               GtkFileChooserDialog  : Write / Construct Only
```

Instance of the GtkFileChooserDialog associated with the button.

The "title" property

"title"	gchararray	: Read / Write

Title to put on the GtkFileChooserDialog associated with the button.

Default value: "Select a File"

The "width-chars" property

"width-chars"	gint	: Read / Write

The width of the entry and label inside the button, in characters.

Allowed values: >= -1

Default value: -1

See Also

GtkFileChooserDialog

<< GtkFileChooser **GtkFileChooserDialog >>**

GtkFileChooserDialog

GtkFileChooserDialog — A file chooser dialog, suitable for "File/Open" or "File/Save" commands

Synopsis

```
#include <gtk/gtk.h>

            GtkFileChooserDialog;
GtkWidget*  gtk_file_chooser_dialog_new     (const gchar *title,
                                             GtkWindow *parent,
                                             GtkFileChooserAction action,
                                             const gchar *first_button_text,
                                             ...);
GtkWidget*  gtk_file_chooser_dialog_new_with_backend
                                            (const gchar *title,
                                             GtkWindow *parent,
                                             GtkFileChooserAction action,
                                             const gchar *backend,
                                             const gchar *first_button_text,
                                             ...);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkWidget
               +----GtkContainer
                     +----GtkBin
                           +----GtkWindow
                                 +----GtkDialog
                                       +----GtkFileChooserDialog
```

Implemented Interfaces

GtkFileChooserDialog implements AtkImplementorIface and GtkFileChooser.

Description

GtkFileChooserDialog is a dialog box suitable for use with "File/Open" or "File/Save as" commands. This widget works by putting a GtkFileChooserWidget inside a GtkDialog. It exposes the GtkFileChooserIface interface, so you can use all of the GtkFileChooser functions on the file chooser dialog as well as those for GtkDialog.

Note that GtkFileChooserDialog does not have any methods of its own. Instead, you should use the functions that work on a GtkFileChooser.

Example 7. Typical usage

In the simplest of cases, you can use GtkFileChooserDialog as in the following code:

```
GtkWidget *dialog;

dialog = gtk_file_chooser_dialog_new ("Open File",
                                      parent_window,
                                      GTK_FILE_CHOOSER_ACTION_OPEN,
                                      GTK_STOCK_CANCEL, GTK_RESPONSE_CANCEL,
                                      GTK_STOCK_OPEN, GTK_RESPONSE_ACCEPT,
                                      NULL);

if (gtk_dialog_run (GTK_DIALOG (dialog)) == GTK_RESPONSE_ACCEPT)
  {
    char *filename;

    filename = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dialog));
    open_file (filename);
    g_free (filename);
  }

gtk_widget_destroy (dialog);
```

Response Codes

GtkFileChooserDialog inherits from GtkDialog, so buttons that go in its action area have response codes such as

GTK_RESPONSE_ACCEPT and GTK_RESPONSE_CANCEL. For example, you could call `gtk_file_chooser_dialog_new()` as follows:

```
GtkWidget *dialog;

dialog = gtk_file_chooser_dialog_new ("Open File",
                                      parent_window,
                                      GTK_FILE_CHOOSER_ACTION_OPEN,
                                      GTK_STOCK_CANCEL, GTK_RESPONSE_CANCEL,
                                      GTK_STOCK_OPEN, GTK_RESPONSE_ACCEPT,
                                      NULL);
```

This will create buttons for "Cancel" and "Open" that use stock response identifiers from GtkResponseType. For most dialog boxes you can use your own custom response codes rather than the ones in GtkResponseType, but GtkFileChooserDialog assumes that its "accept"-type action, e.g. an "Open" or "Save" button, *will* have one of the following response codes:

GTK_RESPONSE_ACCEPT
GTK_RESPONSE_OK
GTK_RESPONSE_YES
GTK_RESPONSE_APPLY

This is because GtkFileChooserDialog must intercept responses and switch to folders if appropriate, rather than letting the dialog terminate — the implementation uses these known response codes to know which responses can be blocked if appropriate.

Note

To summarize, make sure you use a stock response code when you use GtkFileChooserDialog to ensure proper operation.

Details

GtkFileChooserDialog

```
typedef struct _GtkFileChooserDialog GtkFileChooserDialog;
```

---

gtk_file_chooser_dialog_new ()

```
GtkWidget*  gtk_file_chooser_dialog_new     (const gchar *title,
                                             GtkWindow *parent,
                                             GtkFileChooserAction action,
                                             const gchar *first_button_text,
                                             ...);
```

Creates a new GtkFileChooserDialog. This function is analogous to `gtk_dialog_new_with_buttons()`.

title :	Title of the dialog, or NULL
parent :	Transient parent of the dialog, or NULL
action :	Open or save mode for the dialog
first_button_text :	stock ID or text to go in the first button, or NULL
... :	response ID for the first button, then additional (button, id) pairs, ending with NULL
Returns :	a new GtkFileChooserDialog

Since 2.4

---

gtk_file_chooser_dialog_new_with_backend ()

```
GtkWidget*  gtk_file_chooser_dialog_new_with_backend
                                            (const gchar *title,
                                             GtkWindow *parent,
                                             GtkFileChooserAction action,
                                             const gchar *backend,
                                             const gchar *first_button_text,
                                             ...);
```

Creates a new GtkFileChooserDialog with a specified backend. This is especially useful if you use `gtk_file_chooser_set_local_only()` to allow non-local files and you use a more expressive vfs, such as gnome-vfs, to load files.

title :	Title of the dialog, or NULL
parent :	Transient parent of the dialog, or NULL

action :	Open or save mode for the dialog
backend :	The name of the specific filesystem backend to use.
first_button_text :	stock ID or text to go in the first button, or NULL
... :	response ID for the first button, then additional (button, id) pairs, ending with NULL
Returns :	a new GtkFileChooserDialog

Since 2.4

See Also

GtkFileChooser, GtkDialog

<< GtkFileChooserButton **GtkFileChooserWidget >>**

GtkFileChooserWidget

GtkFileChooserWidget — File chooser widget that can be embedded in other widgets

Synopsis

```
#include <gtk/gtk.h>


            GtkFileChooserWidget;
GtkWidget*  gtk_file_chooser_widget_new     (GtkFileChooserAction action);
GtkWidget*  gtk_file_chooser_widget_new_with_backend
                                            (GtkFileChooserAction action,
                                             const gchar *backend);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkWidget
               +----GtkContainer
                     +----GtkBox
                           +----GtkVBox
                                 +----GtkFileChooserWidget
```

Implemented Interfaces

GtkFileChooserWidget implements AtkImplementorIface and GtkFileChooser.

Description

GtkFileChooserWidget is a widget suitable for selecting files. It is the main building block of a GtkFileChooserDialog. Most applications will only need to use the latter; you can use GtkFileChooserWidget as part of a larger window if you have special needs.

Note that GtkFileChooserWidget does not have any methods of its own. Instead, you should use the functions that work on a GtkFileChooser.

Details

GtkFileChooserWidget

```
typedef struct _GtkFileChooserWidget GtkFileChooserWidget;
```

---

gtk_file_chooser_widget_new ()

```
GtkWidget*  gtk_file_chooser_widget_new     (GtkFileChooserAction action);
```

Creates a new GtkFileChooserWidget. This is a file chooser widget that can be embedded in custom windows, and it is the same widget that is used by GtkFileChooserDialog.

action : Open or save mode for the widget

Returns : a new GtkFileChooserWidget

Since 2.4

---

gtk_file_chooser_widget_new_with_backend ()

```
GtkWidget*  gtk_file_chooser_widget_new_with_backend
                                        (GtkFileChooserAction action,
                                         const gchar *backend);
```

Creates a new GtkFileChooserWidget with a specified backend. This is especially useful if you use `gtk_file_chooser_set_local_only()` to allow non-local files. This is a file chooser widget that can be embedded in custom windows and it is the same widget that is used by GtkFileChooserDialog.

action : Open or save mode for the widget
backend : The name of the specific filesystem backend to use.
Returns : a new GtkFileChooserWidget

Since 2.4

See Also

GtkFileChooser, GtkFileChooserDialog

<< GtkFileChooserDialog **GtkFileFilter >>**

GtkFileFilter

GtkFileFilter — A filter for selecting a file subset

Synopsis

```
#include <gtk/gtk.h>

            GtkFileFilter;
            GtkFileFilterInfo;
enum        GtkFileFilterFlags;
gboolean    (*GtkFileFilterFunc)            (const GtkFileFilterInfo *filter_info,
                                             gpointer data);
GtkFileFilter* gtk_file_filter_new          (void);
void        gtk_file_filter_set_name        (GtkFileFilter *filter,
                                             const gchar *name);
G_CONST_RETURN gchar* gtk_file_filter_get_name
                                            (GtkFileFilter *filter);
void        gtk_file_filter_add_mime_type   (GtkFileFilter *filter,
                                             const gchar *mime_type);
void        gtk_file_filter_add_pattern     (GtkFileFilter *filter,
                                             const gchar *pattern);
void        gtk_file_filter_add_custom      (GtkFileFilter *filter,
                                             GtkFileFilterFlags needed,
                                             GtkFileFilterFunc func,
                                             gpointer data,
                                             GDestroyNotify notify);
GtkFileFilterFlags gtk_file_filter_get_needed
                                            (GtkFileFilter *filter);
gboolean    gtk_file_filter_filter          (GtkFileFilter *filter,
                                             const GtkFileFilterInfo *filter_info);
```

Object Hierarchy

```
  GObject
```

```
   +----GtkObject
         +----GtkFileFilter
```

Description

Details

GtkFileFilter

```
typedef struct _GtkFileFilter GtkFileFilter;
```

GtkFileFilterInfo

```
typedef struct {
  GtkFileFilterFlags contains;

  const gchar *filename;
  const gchar *uri;
  const gchar *display_name;
  const gchar *mime_type;
} GtkFileFilterInfo;
```

---

enum GtkFileFilterFlags

```
typedef enum {
  GTK_FILE_FILTER_FILENAME     = 1 << 0,
  GTK_FILE_FILTER_URI          = 1 << 1,
  GTK_FILE_FILTER_DISPLAY_NAME = 1 << 2,
  GTK_FILE_FILTER_MIME_TYPE    = 1 << 3
} GtkFileFilterFlags;
```

GtkFileFilterFunc ()

```
gboolean    (*GtkFileFilterFunc)           (const GtkFileFilterInfo *filter_info,
                                            gpointer data);
```

filter_info :
data :
Returns :

---

gtk_file_filter_new ()

```
GtkFileFilter* gtk_file_filter_new          (void);
```

Creates a new GtkFileFilter with no rules added to it. Such a filter doesn't accept any files, so is not particularly useful until you add rules with `gtk_file_filter_add_mime_type()`, `gtk_file_filter_add_pattern()`, or `gtk_file_filter_add_custom()`. To create a filter that accepts any file, use:

```
GtkFileFilter *filter = gtk_file_filter_new ();
gtk_file_filter_add_pattern (filter, "*");
```

Returns : a new GtkFileFilter

Since 2.4

---

gtk_file_filter_set_name ()

```
void        gtk_file_filter_set_name        (GtkFileFilter *filter,
                                             const gchar *name);
```

Sets the human-readable name of the filter; this is the string that will be displayed in the file selector user interface if there is a selectable list of filters.

filter : a GtkFileFilter
name : the human-readable-name for the filter, or `NULL` to remove any existing name.

Since 2.4

---

gtk_file_filter_get_name ()

```
G_CONST_RETURN gchar* gtk_file_filter_get_name
                                                (GtkFileFilter *filter);
```

Gets the human-readable name for the filter. See `gtk_file_filter_set_name()`.

filter : a GtkFileFilter

Returns : The human-readable name of the filter, or NULL. This value is owned by GTK+ and must not be modified or freed.

Since 2.4

---

gtk_file_filter_add_mime_type ()

```
void        gtk_file_filter_add_mime_type   (GtkFileFilter *filter,
                                             const gchar *mime_type);
```

Adds a rule allowing a given mime type to *filter*.

filter : A GtkFileFilter

mime_type : name of a MIME type

Since 2.4

---

gtk_file_filter_add_pattern ()

```
void        gtk_file_filter_add_pattern     (GtkFileFilter *filter,
                                             const gchar *pattern);
```

Adds a rule allowing a shell style glob to a filter.

filter : a GtkFileFilter

pattern : a shell style glob

Since 2.4

---

gtk_file_filter_add_custom ()

```
void        gtk_file_filter_add_custom      (GtkFileFilter *filter,
                                             GtkFileFilterFlags needed,
                                             GtkFileFilterFunc func,
                                             gpointer data,
                                             GDestroyNotify notify);
```

Adds rule to a filter that allows files based on a custom callback function. The bitfield *needed* which is passed in provides information about what sorts of information that the filter function needs; this allows GTK+ to avoid retrieving expensive information when it isn't needed by the filter.

filter : a GtkFileFilter
needed : bitfield of flags indicating the information that the custom filter function needs.
func : callback function; if the function returns TRUE, then the file will be displayed.
data : data to pass to *func*
notify : function to call to free *data* when it is no longer needed.

Since 2.4

---

gtk_file_filter_get_needed ()

```
GtkFileFilterFlags gtk_file_filter_get_needed
                                            (GtkFileFilter *filter);
```

Gets the fields that need to be filled in for the structure passed to gtk_file_filter_filter()

This function will not typically be used by applications; it is intended principally for use in the implementation of GtkFileChooser.

filter : a GtkFileFilter
Returns : bitfield of flags indicating needed fields when calling gtk_file_filter_filter()

Since 2.4

---

gtk_file_filter_filter ()

```
gboolean    gtk_file_filter_filter          (GtkFileFilter *filter,
                                             const GtkFileFilterInfo *filter_info);
```

Tests whether a file should be displayed according to *filter*. The GtkFileFilterInfo structure *filter_info* should include the fields returned feom gtk_file_filter_get_needed().

This function will not typically be used by applications; it is intended principally for use in the implementation of GtkFileChooser.

filter :	a GtkFileFilter
filter_info :	a GtkFileFilterInfo structure containing information about a file.
Returns :	TRUE if the file should be displayed

Since 2.4

<< GtkFileChooserWidget **GtkFontButton >>**

GtkFontButton

GtkFontButton — A button to launch a font selection dialog

Synopsis

```
#include <gtk/gtk.h>


            GtkFontButton;
GtkWidget*  gtk_font_button_new             (void);
GtkWidget*  gtk_font_button_new_with_font   (const gchar *fontname);
gboolean    gtk_font_button_set_font_name   (GtkFontButton *font_button,
                                             const gchar *fontname);
G_CONST_RETURN gchar* gtk_font_button_get_font_name
                                            (GtkFontButton *font_button);
void        gtk_font_button_set_show_style  (GtkFontButton *font_button,
                                             gboolean show_style);
gboolean    gtk_font_button_get_show_style  (GtkFontButton *font_button);
void        gtk_font_button_set_show_size   (GtkFontButton *font_button,
                                             gboolean show_size);
gboolean    gtk_font_button_get_show_size   (GtkFontButton *font_button);
void        gtk_font_button_set_use_font    (GtkFontButton *font_button,
                                             gboolean use_font);
gboolean    gtk_font_button_get_use_font    (GtkFontButton *font_button);
void        gtk_font_button_set_use_size    (GtkFontButton *font_button,
                                             gboolean use_size);
gboolean    gtk_font_button_get_use_size    (GtkFontButton *font_button);
void        gtk_font_button_set_title       (GtkFontButton *font_button,
                                             const gchar *title);
G_CONST_RETURN gchar* gtk_font_button_get_title
                                            (GtkFontButton *font_button);
```

Object Hierarchy

```
GObject
 +----GtkObject
       +----GtkWidget
             +----GtkContainer
                   +----GtkBin
                         +----GtkButton
                               +----GtkFontButton
```

Implemented Interfaces

GtkFontButton implements AtkImplementorIface.

Properties

```
"font-name"           gchararray           : Read / Write
"show-size"           gboolean             : Read / Write
"show-style"          gboolean             : Read / Write
"title"               gchararray           : Read / Write
"use-font"            gboolean             : Read / Write
"use-size"            gboolean             : Read / Write
```

Signal Prototypes

```
"font-set"  void        user_function      (GtkFontButton *widget,
                                            gpointer user_data);
```

Description

The GtkFontButton is a button which displays the currently selected font an allows to open a font selection dialog to change the font. It is suitable widget for selecting a font in a preference dialog.

Details

GtkFontButton

```
typedef struct _GtkFontButton GtkFontButton;
```

---

gtk_font_button_new ()

```
GtkWidget*  gtk_font_button_new             (void);
```

Creates a new font picker widget.

Returns : a new font picker widget.

Since 2.4

---

gtk_font_button_new_with_font ()

```
GtkWidget*  gtk_font_button_new_with_font   (const gchar *fontname);
```

Creates a new font picker widget.

fontname : Name of font to display in font selection dialog
Returns : a new font picker widget.

Since 2.4

---

gtk_font_button_set_font_name ()

```
gboolean    gtk_font_button_set_font_name   (GtkFontButton *font_button,
                                             const gchar *fontname);
```

Sets or updates the currently-displayed font in font picker dialog.

font_button :	a GtkFontButton
fontname :	Name of font to display in font selection dialog
Returns :	Return value of gtk_font_selection_dialog_set_font_name() if the font selection dialog exists, otherwise FALSE.

Since 2.4

---

gtk_font_button_get_font_name ()

```
G_CONST_RETURN gchar* gtk_font_button_get_font_name
                                            (GtkFontButton *font_button);
```

Retrieves the name of the currently selected font.

font_button :	a GtkFontButton
Returns :	an internal copy of the font name which must not be freed.

Since 2.4

---

gtk_font_button_set_show_style ()

```
void        gtk_font_button_set_show_style  (GtkFontButton *font_button,
                                             gboolean show_style);
```

If *show_style* is TRUE, the font style will be displayed along with name of the selected font.

font_button : a GtkFontButton
show_style : TRUE if font style should be displayed in label.

Since 2.4

---

gtk_font_button_get_show_style ()

```
gboolean    gtk_font_button_get_show_style  (GtkFontButton *font_button);
```

Returns whether the name of the font style will be shown in the label.

font_button : a GtkFontButton
Returns : whether the font style will be shown in the label.

Since 2.4

---

gtk_font_button_set_show_size ()

```
void        gtk_font_button_set_show_size   (GtkFontButton *font_button,
                                             gboolean show_size);
```

If *show_size* is TRUE, the font size will be displayed along with the name of the selected font.

font_button : a GtkFontButton
show_size : TRUE if font size should be displayed in dialog.

Since 2.4

---

gtk_font_button_get_show_size ()

```
gboolean    gtk_font_button_get_show_size   (GtkFontButton *font_button);
```

Returns whether the font size will be shown in the label.

font_button : a GtkFontButton
Returns : whether the font size will be shown in the label.

Since 2.4

---

gtk_font_button_set_use_font ()

```
void        gtk_font_button_set_use_font    (GtkFontButton *font_button,
                                             gboolean use_font);
```

If *use_font* is TRUE, the font name will be written using the selected font.

font_button : a GtkFontButton
use_font : If TRUE, font name will be written using font chosen.

Since 2.4

---

gtk_font_button_get_use_font ()

```
gboolean    gtk_font_button_get_use_font    (GtkFontButton *font_button);
```

Returns whether the selected font is used in the label.

font_button : a GtkFontButton
Returns : whether the selected font is used in the label.

Since 2.4

---

gtk_font_button_set_use_size ()

```
void        gtk_font_button_set_use_size    (GtkFontButton *font_button,
                                             gboolean use_size);
```

If *use_size* is TRUE, the font name will be written using the selected size.

font_button : a GtkFontButton
use_size : If TRUE, font name will be written using the selected size.

Since 2.4

---

gtk_font_button_get_use_size ()

```
gboolean    gtk_font_button_get_use_size    (GtkFontButton *font_button);
```

Returns whether the selected size is used in the label.

font_button : a GtkFontButton
Returns : whether the selected size is used in the label.

Since 2.4

---

gtk_font_button_set_title ()

```
void        gtk_font_button_set_title       (GtkFontButton *font_button,
                                             const gchar *title);
```

Sets the title for the font selection dialog.

font_button : a GtkFontButton

title : a string containing the font selection dialog title

Since 2.4

---

gtk_font_button_get_title ()

```
G_CONST_RETURN gchar* gtk_font_button_get_title
                                                (GtkFontButton *font_button);
```

Retrieves the title of the font selection dialog.

font_button : a GtkFontButton

Returns : an internal copy of the title string which must not be freed.

Since 2.4

Properties

The "font-name" property

```
  "font-name"           gchararray           : Read / Write
```

The name of the currently selected font.

Default value: "Sans 12"

Since 2.4

The "show-size" property

```

```

```
"show-size"             gboolean              : Read / Write
```

If this property is set to TRUE, the selected font size will be shown in the label. For a more WYSIWIG way to show the selected size, see the ::use-size property.

Default value: TRUE

Since 2.4

The "show-style" property

```
"show-style"            gboolean              : Read / Write
```

If this property is set to TRUE, the name of the selected font style will be shown in the label. For a more WYSIWIG way to show the selected style, see the ::use-font property.

Default value: TRUE

Since 2.4

The "title" property

```
"title"                 gchararray            : Read / Write
```

The title of the font selection dialog.

Default value: "Pick a Font"

Since 2.4

The "use-font" property

```
  "use-font"                gboolean                : Read / Write
```

If this property is set to TRUE, the label will be drawn in the selected font.

Default value: FALSE

Since 2.4

The "use-size" property

```
  "use-size"                gboolean                : Read / Write
```

If this property is set to TRUE, the label will be drawn with the selected font size.

Default value: FALSE

Since 2.4

Signals

The "font-set" signal

```
void        user_function                   (GtkFontButton *widget,
                                             gpointer user_data);
```

The ::font-set signal is emitted when the user selects a font. When handling this signal, use `gtk_font_button_get_font_name()` to find out which font was just selected.

widget : the object which received the signal.
user_data : user data set when the signal handler was connected.

Since 2.4

See Also

GtkFontSelectionDialog, GtkColorButton.

<< GtkFileFilter **GtkFontSelection >>**

GtkFontSelection

GtkFontSelection — A widget for selecting fonts

Synopsis

```
#include <gtk/gtk.h>


            GtkFontSelection;
GtkWidget*  gtk_font_selection_new          (void);
GdkFont*    gtk_font_selection_get_font     (GtkFontSelection *fontsel);
gchar*      gtk_font_selection_get_font_name
                                            (GtkFontSelection *fontsel);
gboolean    gtk_font_selection_set_font_name
                                            (GtkFontSelection *fontsel,
                                             const gchar *fontname);
G_CONST_RETURN gchar* gtk_font_selection_get_preview_text
                                            (GtkFontSelection *fontsel);
void        gtk_font_selection_set_preview_text
                                            (GtkFontSelection *fontsel,
                                             const gchar *text);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkWidget
               +----GtkContainer
                     +----GtkBox
```

```
                    +----GtkVBox
                          +----GtkFontSelection
```

Implemented Interfaces

GtkFontSelection implements AtkImplementorIface.

Properties

```
  "font"                GdkFont               : Read
  "font-name"           gchararray            : Read / Write
  "preview-text"        gchararray            : Read / Write
```

Description

The GtkFontSelection widget lists the available fonts, styles and sizes, allowing the user to select a font. It is used in the GtkFontSelectionDialog widget to provide a dialog box for selecting fonts.

To set the font which is initially selected, use `gtk_font_selection_set_font_name()`.

To get the selected font use `gtk_font_selection_get_font()` or `gtk_font_selection_get_font_name()`.

To change the text which is shown in the preview area, use `gtk_font_selection_set_preview_text()`.

Details

GtkFontSelection

```
typedef struct _GtkFontSelection GtkFontSelection;
```

The GtkFontSelection struct contains private data only, and should only be accessed using the functions below.

---

gtk_font_selection_new ()

```
GtkWidget*  gtk_font_selection_new          (void);
```

Creates a new GtkFontSelection.

Returns : a new GtkFontSelection.

---

gtk_font_selection_get_font ()

```
GdkFont*    gtk_font_selection_get_font     (GtkFontSelection *fontsel);
```

Warning

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

Gets the currently-selected font.

`fontsel` : a GtkFontSelection.

Returns : the currently-selected font, or NULL if no font is selected.

---

gtk_font_selection_get_font_name ()

```
gchar*      gtk_font_selection_get_font_name
                                            (GtkFontSelection *fontsel);
```

Gets the currently-selected font name.

`fontsel` : a GtkFontSelection.

Returns :

---

gtk_font_selection_set_font_name ()

```
gboolean    gtk_font_selection_set_font_name
                                            (GtkFontSelection *fontsel,
                                             const gchar *fontname);
```

Sets the currently-selected font.

fontsel : a GtkFontSelection.

fontname : a fontname.

Returns : TRUE if the font was found.

---

gtk_font_selection_get_preview_text ()

```
G_CONST_RETURN gchar* gtk_font_selection_get_preview_text
                                            (GtkFontSelection *fontsel);
```

Gets the text displayed in the preview area.

fontsel : a GtkFontSelection.

Returns : the text displayed in the preview area. This string is owned by the widget and should not be modified or freed.

---

gtk_font_selection_set_preview_text ()

```
void        gtk_font_selection_set_preview_text
                                            (GtkFontSelection *fontsel,
                                             const gchar *text);
```

Sets the text displayed in the preview area.

fontsel : a GtkFontSelection.

text : the text to display in the preview area.

Properties

The "font" property

"font"	GdkFont	: Read

The GdkFont that is currently selected.

The "font-name" property

"font-name"	gchararray	: Read / Write

The X string that represents this font.

Default value: NULL

The "preview-text" property

"preview-text"	gchararray	: Read / Write

The text to display in order to demonstrate the selected font.

Default value: "abcdefghijk ABCDEFGHIJK"

See Also

GtkFontSelectionDialog a dialog box which uses GtkFontSelection.

<< GtkFontButton **GtkFontSelectionDialog >>**

GtkFontSelectionDialog

GtkFontSelectionDialog — A dialog box for selecting fonts

Synopsis

```
#include <gtk/gtk.h>


            GtkFontSelectionDialog;
GtkWidget*  gtk_font_selection_dialog_new   (const gchar *title);
GdkFont*    gtk_font_selection_dialog_get_font
                                            (GtkFontSelectionDialog *fsd);
gchar*      gtk_font_selection_dialog_get_font_name
                                            (GtkFontSelectionDialog *fsd);
gboolean    gtk_font_selection_dialog_set_font_name
                                            (GtkFontSelectionDialog *fsd,
                                             const gchar *fontname);
G_CONST_RETURN gchar* gtk_font_selection_dialog_get_preview_text
                                            (GtkFontSelectionDialog *fsd);
void        gtk_font_selection_dialog_set_preview_text
                                            (GtkFontSelectionDialog *fsd,
                                             const gchar *text);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkWidget
               +----GtkContainer
                     +----GtkBin
```

```
                    +----GtkWindow
                          +----GtkDialog
                                +----GtkFontSelectionDialog
```

Implemented Interfaces

GtkFontSelectionDialog implements AtkImplementorIface.

Description

The GtkFontSelectionDialog widget is a dialog box for selecting a font.

To set the font which is initially selected, use `gtk_font_selection_dialog_set_font_name()`.

To get the selected font use `gtk_font_selection_dialog_get_font()` or `gtk_font_selection_dialog_get_font_name()`.

To change the text which is shown in the preview area, use `gtk_font_selection_dialog_set_preview_text()`.

Details

GtkFontSelectionDialog

```
typedef struct {
  GtkWidget *ok_button;
  GtkWidget *apply_button;
  GtkWidget *cancel_button;
} GtkFontSelectionDialog;
```

GtkWidget *`ok_button`;	The OK button of the dialog
GtkWidget *`apply_button`;	The Apply button of the dialog. This button is hidden by default but you can show/hide it
GtkWidget *`cancel_button`;	The Cancel button of the dialog

---

gtk_font_selection_dialog_new ()

```
GtkWidget*  gtk_font_selection_dialog_new   (const gchar *title);
```

Creates a new GtkFontSelectionDialog.

title : the title of the dialog box.

Returns : a new GtkFontSelectionDialog.

---

gtk_font_selection_dialog_get_font ()

```
GdkFont*    gtk_font_selection_dialog_get_font
                                            (GtkFontSelectionDialog *fsd);
```

Warning

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

Gets the currently-selected font.

fsd : a GtkFontSelectionDialog.

Returns : the currently-selected font, or `NULL` if no font is selected.

---

gtk_font_selection_dialog_get_font_name ()

```
gchar*      gtk_font_selection_dialog_get_font_name
                                            (GtkFontSelectionDialog *fsd);
```

Gets the currently-selected font name.

fsd : a GtkFontSelectionDialog.

Returns : the currently-selected font name, or `NULL` if no font is selected.

---

gtk_font_selection_dialog_set_font_name ()

```
gboolean    gtk_font_selection_dialog_set_font_name
                                            (GtkFontSelectionDialog *fsd,
                                             const gchar *fontname);
```

Sets the currently-selected font.

fsd : a GtkFontSelectionDialog.

fontname : a fontname.

Returns : TRUE if the font was found.

---

gtk_font_selection_dialog_get_preview_text ()

```
G_CONST_RETURN gchar* gtk_font_selection_dialog_get_preview_text
                                            (GtkFontSelectionDialog *fsd);
```

Gets the text displayed in the preview area.

fsd : a GtkFontSelectionDialog.

Returns : the text displayed in the preview area. This string is owned by the widget and should not be modified or freed.

---

gtk_font_selection_dialog_set_preview_text ()

```
void        gtk_font_selection_dialog_set_preview_text
                                            (GtkFontSelectionDialog *fsd,
                                             const gchar *text);
```

Sets the text displayed in the preview area.

fsd : a GtkFontSelectionDialog.

text : the text to display in the preview area.

See Also

	the underlying widget for selecting fonts.
GtkFontSelection, GtkDialog,	
	the parent class of GtkFontSelectionDialog

<< GtkFontSelection **GtkInputDialog >>**

GtkInputDialog

GtkInputDialog — Configure devices for the XInput extension

Synopsis

```
#include <gtk/gtk.h>

            GtkInputDialog;
GtkWidget*  gtk_input_dialog_new            (void);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkWidget
               +----GtkContainer
                     +----GtkBin
                           +----GtkWindow
                                 +----GtkDialog
                                       +----GtkInputDialog
```

Implemented Interfaces

GtkInputDialog implements AtkImplementorIface.

Signal Prototypes

```
"disable-device"
            void        user_function      (GtkInputDialog *inputdialog,
                                            GdkDevice *deviceid,
                                            gpointer user_data);
"enable-device"
            void        user_function      (GtkInputDialog *inputdialog,
                                            GdkDevice *deviceid,
                                            gpointer user_data);
```

Description

NOTE this widget is considered too specialized/little-used for GTK+, and will in the future be moved to some other package. If your application needs this widget, feel free to use it, as the widget does work and is useful in some applications; it's just not of general interest. However, we are not accepting new features for the widget, and it will eventually move out of the GTK+ distribution.

GtkInputDialog displays a dialog which allows the user to configure XInput extension devices. For each device, they can control the mode of the device (disabled, screen-relative, or window-relative), the mapping of axes to coordinates, and the mapping of the devices macro keys to key press events.

GtkInputDialog contains two buttons to which the application can connect; one for closing the dialog, and one for saving the changes. No actions are bound to these by default. The changes that the user makes take effect immediately.

Details

GtkInputDialog

```
typedef struct _GtkInputDialog GtkInputDialog;
```

---

gtk_input_dialog_new ()

```
GtkWidget*  gtk_input_dialog_new           (void);
```

Creates a new GtkInputDialog.

Returns : the new GtkInputDialog.

Signals

The "disable-device" signal

```
void        user_function                   (GtkInputDialog *inputdialog,
                                             GdkDevice *deviceid,
                                             gpointer user_data);
```

This signal is emitted when the user changes the mode of a device from a GDK_MODE_SCREEN or GDK_MODE_WINDOW to GDK_MODE_ENABLED.

inputdialog : the object which received the signal.
deviceid : The ID of the newly disabled device.
user_data : user data set when the signal handler was connected.

The "enable-device" signal

```
void        user_function                   (GtkInputDialog *inputdialog,
                                             GdkDevice *deviceid,
                                             gpointer user_data);
```

This signal is emitted when the user changes the mode of a device from GDK_MODE_DISABLED to a GDK_MODE_SCREEN or GDK_MODE_WINDOW.

inputdialog : the object which received the signal.
deviceid : The ID of the newly enabled device.
user_data : user data set when the signal handler was connected.

<< GtkFontSelectionDialog **Layout Containers >>**

Layout Containers

GtkAlignment - A widget which controls the alignment and size of its child
GtkAspectFrame - A frame that constrains its child to a particular aspect ratio
GtkHBox - A horizontal container box
GtkVBox - A vertical container box
GtkHButtonBox - A container for arranging buttons horizontally
GtkVButtonBox - A container for arranging buttons vertically
GtkFixed - A container which allows you to position widgets at fixed coordinates
GtkHPaned - A container with two panes arranged horizontally
GtkVPaned - A container with two panes arranged vertically
GtkLayout - Infinite scrollable area containing child widgets and/or custom drawing
GtkNotebook - A tabbed notebook container
GtkTable - Pack widgets in regular patterns
GtkExpander - A container which can hide its child

<< GtkInputDialog **GtkAlignment >>**

GtkAlignment

GtkAlignment — A widget which controls the alignment and size of its child

Synopsis

```
#include <gtk/gtk.h>

            GtkAlignment;
GtkWidget*  gtk_alignment_new               (gfloat xalign,
                                             gfloat yalign,
                                             gfloat xscale,
                                             gfloat yscale);
void        gtk_alignment_set               (GtkAlignment *alignment,
                                             gfloat xalign,
                                             gfloat yalign,
                                             gfloat xscale,
                                             gfloat yscale);
void        gtk_alignment_get_padding       (GtkAlignment *alignment,
                                             guint *padding_top,
                                             guint *padding_bottom,
                                             guint *padding_left,
                                             guint *padding_right);
void        gtk_alignment_set_padding       (GtkAlignment *alignment,
                                             guint padding_top,
                                             guint padding_bottom,
                                             guint padding_left,
                                             guint padding_right);
```

Object Hierarchy

```
GObject
 +----GtkObject
       +----GtkWidget
             +----GtkContainer
                   +----GtkBin
                         +----GtkAlignment
```

Implemented Interfaces

GtkAlignment implements AtkImplementorIface.

Properties

```
"bottom-padding"       guint                 : Read / Write
"left-padding"         guint                 : Read / Write
"right-padding"        guint                 : Read / Write
"top-padding"          guint                 : Read / Write
"xalign"               gfloat                : Read / Write
"xscale"               gfloat                : Read / Write
"yalign"               gfloat                : Read / Write
"yscale"               gfloat                : Read / Write
```

Description

The GtkAlignment widget controls the alignment and size of its child widget. It has four settings: xscale, yscale, xalign, and yalign.

The scale settings are used to specify how much the child widget should expand to fill the space allocated to the GtkAlignment. The values can range from 0 (meaning the child doesn't expand at all) to 1 (meaning the child expands to fill all of the available space).

The align settings are used to place the child widget within the available area. The values range from 0 (top or left) to 1 (bottom or right). Of course, if the scale settings are both set to 1, the alignment settings have no effect.

Details

GtkAlignment

```
typedef struct _GtkAlignment GtkAlignment;
```

The GtkAlignment-struct struct contains private data only, and should be accessed using the functions below.

---

gtk_alignment_new ()

```
GtkWidget*  gtk_alignment_new                (gfloat xalign,
                                              gfloat yalign,
                                              gfloat xscale,
                                              gfloat yscale);
```

Creates a new GtkAlignment.

xalign : the horizontal alignment of the child widget, from 0 (left) to 1 (right).

yalign : the vertical alignment of the child widget, from 0 (top) to 1 (bottom).

xscale : the amount that the child widget expands horizontally to fill up unused space, from 0 to 1. A value of 0 indicates that the child widget should never expand. A value of 1 indicates that the child widget will expand to fill all of the space allocated for the GtkAlignment.

yscale : the amount that the child widget expands vertically to fill up unused space, from 0 to 1. The values are similar to *xscale*.

Returns : the new GtkAlignment.

---

gtk_alignment_set ()

```
void        gtk_alignment_set               (GtkAlignment *alignment,
                                             gfloat xalign,
                                             gfloat yalign,
                                             gfloat xscale,
                                             gfloat yscale);
```

Sets the GtkAlignment values.

alignment :	a GtkAlignment.
xalign :	the horizontal alignment of the child widget, from 0 (left) to 1 (right).
yalign :	the vertical alignment of the child widget, from 0 (top) to 1 (bottom).
xscale :	the amount that the child widget expands horizontally to fill up unused space, from 0 to 1. A value of 0 indicates that the child widget should never expand. A value of 1 indicates that the child widget will expand to fill all of the space allocated for the GtkAlignment.
yscale :	the amount that the child widget expands vertically to fill up unused space, from 0 to 1. The values are similar to *xscale*.

---

gtk_alignment_get_padding ()

```
void        gtk_alignment_get_padding       (GtkAlignment *alignment,
                                             guint *padding_top,
                                             guint *padding_bottom,
                                             guint *padding_left,
                                             guint *padding_right);
```

Gets the padding on the different sides of the widget. See gtk_alignment_set_padding().

alignment :	a GtkAlignment
padding_top :	location to store the padding for the top of the widget, or NULL
padding_bottom :	location to store the padding for the bottom of the widget, or NULL

padding_left : location to store the padding for the left of the widget, or NULL
padding_right : location to store the padding for the right of the widget, or NULL

Since 2.4

---

gtk_alignment_set_padding ()

```
void        gtk_alignment_set_padding       (GtkAlignment *alignment,
                                             guint padding_top,
                                             guint padding_bottom,
                                             guint padding_left,
                                             guint padding_right);
```

Sets the padding on the different sides of the widget. The padding adds blank space to the sides of the widget. For instance, this can be used to indent the child widget towards the right by adding padding on the left.

alignment : a GtkAlignment
padding_top : the padding at the top of the widget
padding_bottom : the padding at the bottom of the widget
padding_left : the padding at the left of the widget
padding_right : the padding at the right of the widget.

Since 2.4

Properties

The "bottom-padding" property

```
  "bottom-padding"       guint                 : Read / Write
```

The padding to insert at the bottom of the widget.

Allowed values: <= G_MAXINT

Default value: 0

The "left-padding" property

```
"left-padding"         guint                 : Read / Write
```

The padding to insert at the left of the widget.

Allowed values: <= G_MAXINT

Default value: 0

The "right-padding" property

```
"right-padding"        guint                 : Read / Write
```

The padding to insert at the right of the widget.

Allowed values: <= G_MAXINT

Default value: 0

The "top-padding" property

```
"top-padding"          guint                 : Read / Write
```

The padding to insert at the right of the widget.

Allowed values: <= G_MAXINT

Default value: 0

Since 2.4

The "xalign" property

```
  "xalign"                gfloat                : Read / Write
```

Horizontal position of child in available space. 0.0 is left aligned, 1.0 is right aligned.

Allowed values: [0,1]

Default value: 0.5

The "xscale" property

```
  "xscale"                gfloat                : Read / Write
```

If available horizontal space is bigger than needed for the child, how much of it to use for the child. 0.0 means none, 1.0 means all.

Allowed values: [0,1]

Default value: 1

The "yalign" property

```
  "yalign"                gfloat                : Read / Write
```

Vertical position of child in available space. 0.0 is top aligned, 1.0 is bottom aligned.

Allowed values: [0,1]

Default value: 0.5

The "yscale" property

```
"yscale"                gfloat                : Read / Write
```

If available vertical space is bigger than needed for the child, how much of it to use for the child. 0.0 means none, 1.0 means all.

Allowed values: [0,1]

Default value: 1

<< Layout Containers **GtkAspectFrame >>**

GtkAspectFrame

GtkAspectFrame — A frame that constrains its child to a particular aspect ratio

Synopsis

```
#include <gtk/gtk.h>

            GtkAspectFrame;
GtkWidget*  gtk_aspect_frame_new            (const gchar *label,
                                             gfloat xalign,
                                             gfloat yalign,
                                             gfloat ratio,
                                             gboolean obey_child);
void        gtk_aspect_frame_set            (GtkAspectFrame *aspect_frame,
                                             gfloat xalign,
                                             gfloat yalign,
                                             gfloat ratio,
                                             gboolean obey_child);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkWidget
               +----GtkContainer
                     +----GtkBin
                           +----GtkFrame
                                 +----GtkAspectFrame
```

Implemented Interfaces

GtkAspectFrame implements AtkImplementorIface.

Properties

```
  "obey-child"          gboolean              : Read / Write
  "ratio"               gfloat                : Read / Write
  "xalign"              gfloat                : Read / Write
  "yalign"              gfloat                : Read / Write
```

Description

The GtkAspectFrame is useful when you want pack a widget so that it can resize but always retains the same aspect ratio. For instance, one might be drawing a small preview of a larger image. GtkAspectFrame derives from GtkFrame, so it can draw a label and a frame around the child. The frame will be "shrink-wrapped" to the size of the child.

Details

GtkAspectFrame

```
typedef struct _GtkAspectFrame GtkAspectFrame;
```

---

gtk_aspect_frame_new ()

```
GtkWidget*  gtk_aspect_frame_new            (const gchar *label,
                                             gfloat xalign,
                                             gfloat yalign,
                                             gfloat ratio,
                                             gboolean obey_child);
```

Create a new GtkAspectFrame.

label :	Label text.
xalign :	Horizontal alignment of the child within the allocation of the GtkAspectFrame. This ranges from 0.0 (left aligned) to 1.0 (right aligned)
yalign :	Vertical alignment of the child within the allocation of the GtkAspectFrame. This ranges from 0.0 (left aligned) to 1.0 (right aligned)
ratio :	The desired aspect ratio.
obey_child :	If TRUE, *ratio* is ignored, and the aspect ratio is taken from the requistion of the child.
Returns :	the new GtkAspectFrame.

---

gtk_aspect_frame_set ()

```
void        gtk_aspect_frame_set            (GtkAspectFrame *aspect_frame,
                                             gfloat xalign,
                                             gfloat yalign,
                                             gfloat ratio,
                                             gboolean obey_child);
```

Set parameters for an existing GtkAspectFrame.

aspect_frame :	a GtkAspectFrame
xalign :	Horizontal alignment of the child within the allocation of the GtkAspectFrame. This ranges from 0.0 (left aligned) to 1.0 (right aligned)
yalign :	Vertical alignment of the child within the allocation of the GtkAspectFrame. This ranges from 0.0 (left aligned) to 1.0 (right aligned)
ratio :	The desired aspect ratio.
obey_child :	If TRUE, *ratio* is ignored, and the aspect ratio is taken from the requistion of the child.

Properties

The "obey-child" property

```
  "obey-child"           gboolean              : Read / Write
```

Force aspect ratio to match that of the frame's child.

Default value: TRUE

The "ratio" property

```
  "ratio"                gfloat                : Read / Write
```

Aspect ratio if obey_child is FALSE.

Allowed values: [1e-04,10000]

Default value: 0.5

The "xalign" property

```
  "xalign"               gfloat                : Read / Write
```

X alignment of the child.

Allowed values: [0,1]

Default value: 0.5

The "yalign" property

```
  "yalign"               gfloat                : Read / Write
```

Y alignment of the child.

Allowed values: [0,1]

Default value: 0.5

<< GtkAlignment **GtkHBox >>**

GtkHBox

GtkHBox — A horizontal container box

Synopsis

```
#include <gtk/gtk.h>


            GtkHBox;
GtkWidget*  gtk_hbox_new                    (gboolean homogeneous,
                                             gint spacing);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkWidget
               +----GtkContainer
                     +----GtkBox
                           +----GtkHBox
                                 +----GtkCombo
                                 +----GtkFileChooserButton
                                 +----GtkStatusbar
```

Implemented Interfaces

GtkHBox implements AtkImplementorIface.

Description

GtkHBox is a container that organizes child widgets into a single row.

Use the GtkBox packing interface to determine the arrangement, spacing, width, and alignment of GtkHBox children.

All children are allocated the same height.

Details

GtkHBox

```
typedef struct _GtkHBox GtkHBox;
```

---

gtk_hbox_new ()

```
GtkWidget*  gtk_hbox_new                    (gboolean homogeneous,
                                             gint spacing);
```

Creates a new GtkHBox.

homogeneous : TRUE if all children are to be given equal space allotments.
spacing : the number of pixels to place by default between children.
Returns : a new GtkHBox.

See Also

GtkVBox a sister class that organizes widgets into a column.

<< GtkAspectFrame **GtkVBox >>**

GtkVBox

GtkVBox — A vertical container box

Synopsis

```
#include <gtk/gtk.h>

            GtkVBox;
GtkWidget*  gtk_vbox_new                    (gboolean homogeneous,
                                             gint spacing);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkWidget
               +----GtkContainer
                     +----GtkBox
                           +----GtkVBox
                                 +----GtkColorSelection
                                 +----GtkFileChooserWidget
                                 +----GtkFontSelection
                                 +----GtkGammaCurve
```

Implemented Interfaces

GtkVBox implements AtkImplementorIface.

Description

GtkVBox is a container that organizes child widgets into a single column.

Use the GtkBox packing interface to determine the arrangement, spacing, height, and alignment of GtkVBox children.

All children are allocated the same width.

Details

GtkVBox

```
typedef struct _GtkVBox GtkVBox;
```

---

gtk_vbox_new ()

```
GtkWidget*  gtk_vbox_new                            (gboolean homogeneous,
                                                     gint spacing);
```

Creates a new GtkVBox.

homogeneous : TRUE if all children are to be given equal space allotments.
spacing : the number of pixels to place by default between children.
Returns : a new GtkVBox.

See Also

GtkHBox a sister class that organizes widgets into a row.

<< GtkHBox **GtkHButtonBox >>**

GtkHButtonBox

GtkHButtonBox — A container for arranging buttons horizontally

Synopsis

```
#include <gtk/gtk.h>


            GtkHButtonBox;
GtkWidget*  gtk_hbutton_box_new             (void);
gint        gtk_hbutton_box_get_spacing_default
                                            (void);
GtkButtonBoxStyle gtk_hbutton_box_get_layout_default
                                            (void);
void        gtk_hbutton_box_set_spacing_default
                                            (gint spacing);
void        gtk_hbutton_box_set_layout_default
                                            (GtkButtonBoxStyle layout);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkWidget
               +----GtkContainer
                     +----GtkBox
                           +----GtkButtonBox
                                 +----GtkHButtonBox
```

Implemented Interfaces

GtkHButtonBox implements AtkImplementorIface.

Description

A button box should be used to provide a consistent layout of buttons throughout your application. The layout/spacing can be altered by the programmer, or if desired, by the user to alter the 'feel' of a program to a small degree.

A GtkHButtonBox is created with `gtk_hbutton_box_new()`. Buttons are packed into a button box the same way widgets are added to any other container, using `gtk_container_add()`. You can also use `gtk_box_pack_start()` or `gtk_box_pack_end()`, but for button boxes both these functions work just like `gtk_container_add()`, ie., they pack the button in a way that depends on the current layout style and on whether the button has had `gtk_button_box_set_child_secondary()` called on it.

The spacing between buttons can be set with `gtk_box_set_spacing()`. The arrangement and layout of the buttons can be changed with `gtk_button_box_set_layout()`.

Details

GtkHButtonBox

```
typedef struct _GtkHButtonBox GtkHButtonBox;
```

GtkHButtonBox does not contain any public fields.

---

gtk_hbutton_box_new ()

```
GtkWidget*  gtk_hbutton_box_new             (void);
```

Creates a new horizontal button box.

Returns : a new button box GtkWidget.

---

gtk_hbutton_box_get_spacing_default ()

```
gint        gtk_hbutton_box_get_spacing_default
                                                (void);
```

Warning

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

Retrieves the current default spacing for horizontal button boxes. This is the number of pixels to be placed between the buttons when they are arranged.

Returns : the default number of pixels between buttons.

---

gtk_hbutton_box_get_layout_default ()

```
GtkButtonBoxStyle gtk_hbutton_box_get_layout_default
                                                (void);
```

Warning

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

Retrieves the current layout used to arrange buttons in button box widgets.

Returns : the current GtkButtonBoxStyle.

---

gtk_hbutton_box_set_spacing_default ()

```
void        gtk_hbutton_box_set_spacing_default
                                            (gint spacing);
```

Warning

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

Changes the default spacing that is placed between widgets in an horizontal button box.

`spacing` : an integer value.

---

gtk_hbutton_box_set_layout_default ()

```
void        gtk_hbutton_box_set_layout_default
                                            (GtkButtonBoxStyle layout);
```

Warning

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

Sets a new layout mode that will be used by all button boxes.

`layout` : a new GtkButtonBoxStyle.

See Also

GtkBox	Used to pack widgets into button boxes.
GtkButtonBox	Provides functions for controlling button boxes.
GtkVButtonBox	Pack buttons vertically

<< GtkVBox **GtkVButtonBox >>**

GtkVButtonBox

GtkVButtonBox — A container for arranging buttons vertically

Synopsis

```
#include <gtk/gtk.h>


            GtkVButtonBox;
GtkWidget*  gtk_vbutton_box_new             (void);
gint        gtk_vbutton_box_get_spacing_default
                                            (void);
void        gtk_vbutton_box_set_spacing_default
                                            (gint spacing);
GtkButtonBoxStyle gtk_vbutton_box_get_layout_default
                                            (void);
void        gtk_vbutton_box_set_layout_default
                                            (GtkButtonBoxStyle layout);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkWidget
               +----GtkContainer
                     +----GtkBox
                           +----GtkButtonBox
                                 +----GtkVButtonBox
```

Implemented Interfaces

GtkVButtonBox implements AtkImplementorIface.

Description

A button box should be used to provide a consistent layout of buttons throughout your application. The layout/spacing can be altered by the programmer, or if desired, by the user to alter the 'feel' of a program to a small degree.

A GtkVButtonBox is created with `gtk_vbutton_box_new()`. Buttons are packed into a button box the same way widgets are added to any other container, using `gtk_container_add()`. You can also use `gtk_box_pack_start()` or `gtk_box_pack_end()`, but for button boxes both these functions work just like `gtk_container_add()`, ie., they pack the button in a way that depends on the current layout style and on whether the button has had `gtk_button_box_set_child_secondary()` called on it.

The spacing between buttons can be set with `gtk_box_set_spacing()`. The arrangement and layout of the buttons can be changed with `gtk_button_box_set_layout()`.

Details

GtkVButtonBox

```
typedef struct _GtkVButtonBox GtkVButtonBox;
```

GtkVButtonBox does not contain any public fields.

---

gtk_vbutton_box_new ()

```
GtkWidget*  gtk_vbutton_box_new             (void);
```

Creates a new vertical button box.

Returns : a new button box GtkWidget.

---

gtk_vbutton_box_get_spacing_default ()

```
gint        gtk_vbutton_box_get_spacing_default
                                                (void);
```

Warning

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

Retrieves the current default spacing for vertical button boxes. This is the number of pixels to be placed between the buttons when they are arranged.

Returns : the default number of pixels between buttons.

---

gtk_vbutton_box_set_spacing_default ()

```
void        gtk_vbutton_box_set_spacing_default
                                                (gint spacing);
```

Warning

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

Changes the default spacing that is placed between widgets in an vertical button box.

`spacing` : an integer value.

---

gtk_vbutton_box_get_layout_default ()

```
GtkButtonBoxStyle gtk_vbutton_box_get_layout_default
                                                    (void);
```

Warning

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

Retrieves the current layout used to arrange buttons in button box widgets.

Returns : the current GtkButtonBoxStyle.

---

gtk_vbutton_box_set_layout_default ()

```
void        gtk_vbutton_box_set_layout_default
                                                    (GtkButtonBoxStyle layout);
```

Warning

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

Sets a new layout mode that will be used by all button boxes.

`layout` : a new GtkButtonBoxStyle.

See Also

GtkBox	Used to pack widgets into button boxes.
GtkButtonBox	Provides functions for controlling button boxes.
GtkHButtonBox	Pack buttons horizontally

<< GtkHButtonBox **GtkFixed >>**

GtkFixed

GtkFixed — A container which allows you to position widgets at fixed coordinates

Synopsis

```
#include <gtk/gtk.h>

            GtkFixed;
            GtkFixedChild;
GtkWidget*  gtk_fixed_new                   (void);
void        gtk_fixed_put                   (GtkFixed *fixed,
                                             GtkWidget *widget,
                                             gint x,
                                             gint y);
void        gtk_fixed_move                  (GtkFixed *fixed,
                                             GtkWidget *widget,
                                             gint x,
                                             gint y);
gboolean    gtk_fixed_get_has_window        (GtkFixed *fixed);
void        gtk_fixed_set_has_window        (GtkFixed *fixed,
                                             gboolean has_window);
```

Object Hierarchy

```
  GObject
   +----GtkObject
```

```
          +----GtkWidget
                +----GtkContainer
                      +----GtkFixed
```

Implemented Interfaces

GtkFixed implements AtkImplementorIface.

Child Properties

```
  "x"                    gint                    : Read / Write
  "y"                    gint                    : Read / Write
```

Description

The GtkFixed widget is a container which can place child widgets at fixed positions and with fixed sizes, given in pixels. GtkFixed performs no automatic layout management.

For most applications, you should not use this container! It keeps you from having to learn about the other GTK+ containers, but it results in broken applications. With GtkFixed, the following things will result in truncated text, overlapping widgets, and other display bugs:

- Themes, which may change widget sizes.
- Fonts other than the one you used to write the app will of course change the size of widgets containing text; keep in mind that users may use a larger font because of difficulty reading the default, or they may be using Windows or the framebuffer port of GTK+, where different fonts are available.
- Translation of text into other languages changes its size. Also, display of non-English text will use a different font in many cases.

In addition, the fixed widget can't properly be mirrored in right-to-left languages such as Hebrew and Arabic. i.e. normally GTK+ will flip the interface to put labels to the right of the thing they label, but it can't do that with GtkFixed. So your application will not be usable in right-to-left languages.

Finally, fixed positioning makes it kind of annoying to add/remove GUI elements, since you have to reposition all the other elements. This is a long-term maintenance problem for your application.

If you know none of these things are an issue for your application, and prefer the simplicity of GtkFixed, by all means use the widget. But you should be aware of the tradeoffs.

Details

GtkFixed

```
typedef struct _GtkFixed GtkFixed;
```

The GtkFixed-struct struct contains the following fields. (These fields should be considered read-only. They should never be set by an application.)

GList *children; a list of GtkFixedChild elements, containing the child widgets and their positions.

GtkFixedChild

```
typedef struct {
  GtkWidget *widget;
  gint x;
  gint y;
} GtkFixedChild;
```

The GtkFixedChild-struct struct contains the following fields. (These fields should be considered read-only. They should never be set by an application.)

GtkWidget *widget;	the child GtkWidget.
gint x;	the horizontal position of the widget within the GtkFixed container.
gint y;	the vertical position of the widget within the GtkFixed container.

---

gtk_fixed_new ()

```
GtkWidget*  gtk_fixed_new                   (void);
```

Creates a new GtkFixed.

Returns : a new GtkFixed.

---

gtk_fixed_put ()

```
void        gtk_fixed_put                   (GtkFixed *fixed,
                                             GtkWidget *widget,
                                             gint x,
                                             gint y);
```

Adds a widget to a GtkFixed container at the given position.

fixed : a GtkFixed.
widget : the widget to add.
x : the horizontal position to place the widget at.
y : the vertical position to place the widget at.

---

gtk_fixed_move ()

```
void        gtk_fixed_move                  (GtkFixed *fixed,
                                             GtkWidget *widget,
                                             gint x,
                                             gint y);
```

Moves a child of a GtkFixed container to the given position.

fixed : a GtkFixed.

widget : the child widget.
x : the horizontal position to move the widget to.
y : the vertical position to move the widget to.

---

gtk_fixed_get_has_window ()

```
gboolean    gtk_fixed_get_has_window        (GtkFixed *fixed);
```

Gets whether the GtkFixed has its own GdkWindow. See `gdk_fixed_set_has_window()`.

fixed : a GtkWidget
Returns : TRUE if *fixed* has its own window.

---

gtk_fixed_set_has_window ()

```
void        gtk_fixed_set_has_window        (GtkFixed *fixed,
                                             gboolean has_window);
```

Sets whether a GtkFixed widget is created with a separate GdkWindow for widget->window or not. (By default, it will be created with no separate GdkWindow). This function must be called while the GtkFixed is not realized, for instance, immediately after the window is created.

fixed : a GtkFixed
has_window : TRUE if a separate window should be created

Child Properties

The "x" child property

```
  "x"                    gint                  : Read / Write
```

X position of child widget.

Default value: 0

The "y" child property

"y"	gint	: Read / Write

Y position of child widget.

Default value: 0

<< GtkVButtonBox **GtkHPaned >>**

GtkHPaned

GtkHPaned — A container with two panes arranged horizontally

Synopsis

```
#include <gtk/gtk.h>

            GtkHPaned;
GtkWidget*  gtk_hpaned_new                  (void);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkWidget
               +----GtkContainer
                     +----GtkPaned
                           +----GtkHPaned
```

Implemented Interfaces

GtkHPaned implements AtkImplementorIface.

Description

The HPaned widget is a container widget with two children arranged horizontally. The division between the two panes is adjustable by the user by dragging a handle. See GtkPaned for details.

Details

GtkHPaned

```
typedef struct _GtkHPaned GtkHPaned;
```

---

gtk_hpaned_new ()

```
GtkWidget*  gtk_hpaned_new                    (void);
```

Create a new GtkHPaned

Returns : the new GtkHPaned

<< GtkFixed **GtkVPaned >>**

GtkVPaned

GtkVPaned — A container with two panes arranged vertically

Synopsis

```
#include <gtk/gtk.h>

            GtkVPaned;
GtkWidget*  gtk_vpaned_new                  (void);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkWidget
               +----GtkContainer
                     +----GtkPaned
                           +----GtkVPaned
```

Implemented Interfaces

GtkVPaned implements AtkImplementorIface.

Description

The VPaned widget is a container widget with two children arranged vertically. The division between the two panes is adjustable by the user by dragging a handle. See GtkPaned for details.

Details

GtkVPaned

```
typedef struct _GtkVPaned GtkVPaned;
```

---

gtk_vpaned_new ()

```
GtkWidget*  gtk_vpaned_new                          (void);
```

Create a new GtkVPaned

Returns : the new GtkVPaned

<< GtkHPaned **GtkLayout >>**

GtkLayout

GtkLayout — Infinite scrollable area containing child widgets and/or custom drawing

Synopsis

```
#include <gtk/gtk.h>

            GtkLayout;
GtkWidget*  gtk_layout_new                  (GtkAdjustment *hadjustment,
                                             GtkAdjustment *vadjustment);
void        gtk_layout_put                  (GtkLayout *layout,
                                             GtkWidget *child_widget,
                                             gint x,
                                             gint y);
void        gtk_layout_move                 (GtkLayout *layout,
                                             GtkWidget *child_widget,
                                             gint x,
                                             gint y);
void        gtk_layout_set_size             (GtkLayout *layout,
                                             guint width,
                                             guint height);
void        gtk_layout_get_size             (GtkLayout *layout,
                                             guint *width,
                                             guint *height);
void        gtk_layout_freeze               (GtkLayout *layout);
void        gtk_layout_thaw                 (GtkLayout *layout);
GtkAdjustment* gtk_layout_get_hadjustment   (GtkLayout *layout);
GtkAdjustment* gtk_layout_get_vadjustment   (GtkLayout *layout);
void        gtk_layout_set_hadjustment      (GtkLayout *layout,
                                             GtkAdjustment *adjustment);
void        gtk_layout_set_vadjustment      (GtkLayout *layout,
                                             GtkAdjustment *adjustment);
```

Object Hierarchy

```
GObject
 +----GtkObject
       +----GtkWidget
             +----GtkContainer
                   +----GtkLayout
```

Implemented Interfaces

GtkLayout implements AtkImplementorIface.

Properties

```
"hadjustment"          GtkAdjustment         : Read / Write
"height"               guint                 : Read / Write
"vadjustment"          GtkAdjustment         : Read / Write
"width"                guint                 : Read / Write
```

Child Properties

```
"x"                    gint                  : Read / Write
"y"                    gint                  : Read / Write
```

Signal Prototypes

```
"set-scroll-adjustments"
            void        user_function      (GtkLayout *layout,
```

```
                                                GtkAdjustment *arg1,
                                                GtkAdjustment *arg2,
                                                gpointer user_data);
```

Description

GtkLayout is similar to GtkDrawingArea in that it's a "blank slate" and doesn't do anything but paint a blank background by default. It's different in that it supports scrolling natively (you can add it to a GtkScrolledWindow), and it can contain child widgets, since it's a GtkContainer. However if you're just going to draw, a GtkDrawingArea is a better choice since it has lower overhead.

When handling expose events on a GtkLayout, you must draw to GTK_LAYOUT (layout)->bin_window, rather than to GTK_WIDGET (layout)->window, as you would for a drawing area.

Details

GtkLayout

```
typedef struct {
  GdkWindow *bin_window;
} GtkLayout;
```

---

gtk_layout_new ()

```
GtkWidget*  gtk_layout_new                  (GtkAdjustment *hadjustment,
                                             GtkAdjustment *vadjustment);
```

Creates a new GtkLayout. Unless you have a specific adjustment you'd like the layout to use for scrolling, pass `NULL` for *`hadjustment`* and *`vadjustment`*.

`hadjustment` : horizontal scroll adjustment, or `NULL`

`vadjustment` : vertical scroll adjustment, or `NULL`

Returns : a new GtkLayout

---

gtk_layout_put ()

```
void        gtk_layout_put                  (GtkLayout *layout,
                                             GtkWidget *child_widget,
                                             gint x,
                                             gint y);
```

Adds *child_widget* to *layout*, at position (*x*,*y*). *layout* becomes the new parent container of *child_widget*.

layout :	a GtkLayout
child_widget :	child widget
x :	X position of child widget
y :	Y position of child widget

---

gtk_layout_move ()

```
void        gtk_layout_move                 (GtkLayout *layout,
                                             GtkWidget *child_widget,
                                             gint x,
                                             gint y);
```

Moves a current child of *layout* to a new position.

layout :	a GtkLayout
child_widget :	a current child of *layout*
x :	X position to move to
y :	Y position to move to

---

gtk_layout_set_size ()

```
void        gtk_layout_set_size             (GtkLayout *layout,
                                             guint width,
```

```
                                                guint height);
```

Sets the size of the scrollable area of the layout.

layout : a GtkLayout
width : width of entire scrollable area
height : height of entire scrollable area

---

gtk_layout_get_size ()

```
void        gtk_layout_get_size             (GtkLayout *layout,
                                             guint *width,
                                             guint *height);
```

Gets the size that has been set on the layout, and that determines the total extents of the layout's scrollbar area. See `gtk_layout_set_size()`.

layout : a GtkLayout
width : location to store the width set on *layout*, or `NULL`
height : location to store the height set on *layout*, or `NULL`

---

gtk_layout_freeze ()

```
void        gtk_layout_freeze               (GtkLayout *layout);
```

Warning

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

This is a deprecated function, it doesn't do anything useful.

layout : a GtkLayout

---

gtk_layout_thaw ()

```
void        gtk_layout_thaw                 (GtkLayout *layout);
```

Warning

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

This is a deprecated function, it doesn't do anything useful.

layout : a GtkLayout

---

gtk_layout_get_hadjustment ()

```
GtkAdjustment* gtk_layout_get_hadjustment   (GtkLayout *layout);
```

This function should only be called after the layout has been placed in a GtkScrolledWindow or otherwise configured for scrolling. It returns the GtkAdjustment used for communication between the horizontal scrollbar and *layout*.

See GtkScrolledWindow, GtkScrollbar, GtkAdjustment for details.

layout : a GtkLayout

Returns : horizontal scroll adjustment

---

gtk_layout_get_vadjustment ()

```
GtkAdjustment* gtk_layout_get_vadjustment   (GtkLayout *layout);
```

This function should only be called after the layout has been placed in a GtkScrolledWindow or otherwise configured for scrolling. It returns the GtkAdjustment used for communication between the vertical scrollbar and *layout*.

See GtkScrolledWindow, GtkScrollbar, GtkAdjustment for details.

layout : a GtkLayout
Returns : vertical scroll adjustment

---

gtk_layout_set_hadjustment ()

```
void        gtk_layout_set_hadjustment      (GtkLayout *layout,
                                             GtkAdjustment *adjustment);
```

Sets the horizontal scroll adjustment for the layout.

See GtkScrolledWindow, GtkScrollbar, GtkAdjustment for details.

layout : a GtkLayout
adjustment : new scroll adjustment

---

gtk_layout_set_vadjustment ()

```
void        gtk_layout_set_vadjustment      (GtkLayout *layout,
                                             GtkAdjustment *adjustment);
```

Sets the vertical scroll adjustment for the layout.

See GtkScrolledWindow, GtkScrollbar, GtkAdjustment for details.

layout : a GtkLayout
adjustment : new scroll adjustment

Properties

The "hadjustment" property

```
"hadjustment"          GtkAdjustment          : Read / Write
```

The GtkAdjustment for the horizontal position.

The "height" property

```
"height"               guint                  : Read / Write
```

The height of the layout.

Allowed values: <= G_MAXINT

Default value: 100

The "vadjustment" property

```
"vadjustment"          GtkAdjustment          : Read / Write
```

The GtkAdjustment for the vertical position.

The "width" property

```
"width"                guint                  : Read / Write
```

The width of the layout.

Allowed values: <= G_MAXINT

Default value: 100

Child Properties

The "x" child property

```
  "x"                   gint                  : Read / Write
```

X position of child widget.

Default value: 0

The "y" child property

```
  "y"                   gint                  : Read / Write
```

Y position of child widget.

Default value: 0

Signals

The "set-scroll-adjustments" signal

```
void        user_function                   (GtkLayout *layout,
                                             GtkAdjustment *arg1,
                                             GtkAdjustment *arg2,
                                             gpointer user_data);
```

layout : the object which received the signal.
arg1 :
arg2 :
user_data : user data set when the signal handler was connected.

See Also

GtkDrawingArea, GtkScrolledWindow

<< GtkVPaned **GtkNotebook >>**

GtkNotebook

GtkNotebook — A tabbed notebook container

Synopsis

```
#include <gtk/gtk.h>

            GtkNotebook;
            GtkNotebookPage;
GtkWidget*  gtk_notebook_new                (void);
gint        gtk_notebook_append_page        (GtkNotebook *notebook,
                                             GtkWidget *child,
                                             GtkWidget *tab_label);
gint        gtk_notebook_append_page_menu   (GtkNotebook *notebook,
                                             GtkWidget *child,
                                             GtkWidget *tab_label,
                                             GtkWidget *menu_label);
gint        gtk_notebook_prepend_page       (GtkNotebook *notebook,
                                             GtkWidget *child,
                                             GtkWidget *tab_label);
gint        gtk_notebook_prepend_page_menu  (GtkNotebook *notebook,
                                             GtkWidget *child,
                                             GtkWidget *tab_label,
                                             GtkWidget *menu_label);
gint        gtk_notebook_insert_page        (GtkNotebook *notebook,
                                             GtkWidget *child,
                                             GtkWidget *tab_label,
                                             gint position);
gint        gtk_notebook_insert_page_menu   (GtkNotebook *notebook,
                                             GtkWidget *child,
                                             GtkWidget *tab_label,
                                             GtkWidget *menu_label,
                                             gint position);
void        gtk_notebook_remove_page        (GtkNotebook *notebook,
                                             gint page_num);
```

```
#define     gtk_notebook_current_page
gint        gtk_notebook_page_num           (GtkNotebook *notebook,
                                             GtkWidget *child);
#define     gtk_notebook_set_page
void        gtk_notebook_next_page          (GtkNotebook *notebook);
void        gtk_notebook_prev_page          (GtkNotebook *notebook);
void        gtk_notebook_reorder_child      (GtkNotebook *notebook,
                                             GtkWidget *child,
                                             gint position);
void        gtk_notebook_set_tab_pos        (GtkNotebook *notebook,
                                             GtkPositionType pos);
void        gtk_notebook_set_show_tabs      (GtkNotebook *notebook,
                                             gboolean show_tabs);
void        gtk_notebook_set_show_border    (GtkNotebook *notebook,
                                             gboolean show_border);
void        gtk_notebook_set_scrollable     (GtkNotebook *notebook,
                                             gboolean scrollable);
void        gtk_notebook_set_tab_border     (GtkNotebook *notebook,
                                             guint border_width);
void        gtk_notebook_popup_enable       (GtkNotebook *notebook);
void        gtk_notebook_popup_disable      (GtkNotebook *notebook);
gint        gtk_notebook_get_current_page   (GtkNotebook *notebook);
GtkWidget*  gtk_notebook_get_menu_label     (GtkNotebook *notebook,
                                             GtkWidget *child);
GtkWidget*  gtk_notebook_get_nth_page       (GtkNotebook *notebook,
                                             gint page_num);
gint        gtk_notebook_get_n_pages        (GtkNotebook *notebook);
GtkWidget*  gtk_notebook_get_tab_label      (GtkNotebook *notebook,
                                             GtkWidget *child);
void        gtk_notebook_query_tab_label_packing
                                            (GtkNotebook *notebook,
                                             GtkWidget *child,
                                             gboolean *expand,
                                             gboolean *fill,
                                             GtkPackType *pack_type);
void        gtk_notebook_set_homogeneous_tabs
                                            (GtkNotebook *notebook,
                                             gboolean homogeneous);
void        gtk_notebook_set_menu_label     (GtkNotebook *notebook,
                                             GtkWidget *child,
                                             GtkWidget *menu_label);
void        gtk_notebook_set_menu_label_text
                                            (GtkNotebook *notebook,
```

```
                                                GtkWidget *child,
                                                const gchar *menu_text);
void        gtk_notebook_set_tab_hborder    (GtkNotebook *notebook,
                                                guint tab_hborder);
void        gtk_notebook_set_tab_label      (GtkNotebook *notebook,
                                                GtkWidget *child,
                                                GtkWidget *tab_label);
void        gtk_notebook_set_tab_label_packing
                                            (GtkNotebook *notebook,
                                                GtkWidget *child,
                                                gboolean expand,
                                                gboolean fill,
                                                GtkPackType pack_type);
void        gtk_notebook_set_tab_label_text (GtkNotebook *notebook,
                                                GtkWidget *child,
                                                const gchar *tab_text);
void        gtk_notebook_set_tab_vborder    (GtkNotebook *notebook,
                                                guint tab_vborder);
G_CONST_RETURN gchar* gtk_notebook_get_menu_label_text
                                            (GtkNotebook *notebook,
                                                GtkWidget *child);
gboolean    gtk_notebook_get_scrollable     (GtkNotebook *notebook);
gboolean    gtk_notebook_get_show_border    (GtkNotebook *notebook);
gboolean    gtk_notebook_get_show_tabs      (GtkNotebook *notebook);
G_CONST_RETURN gchar* gtk_notebook_get_tab_label_text
                                            (GtkNotebook *notebook,
                                                GtkWidget *child);
GtkPositionType gtk_notebook_get_tab_pos    (GtkNotebook *notebook);
void        gtk_notebook_set_current_page   (GtkNotebook *notebook,
                                                gint page_num);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkWidget
               +----GtkContainer
                     +----GtkNotebook
```

Implemented Interfaces

GtkNotebook implements AtkImplementorIface.

Properties

```
"enable-popup"          gboolean              : Read / Write
"homogeneous"           gboolean              : Read / Write
"page"                  gint                  : Read / Write
"scrollable"            gboolean              : Read / Write
"show-border"           gboolean              : Read / Write
"show-tabs"             gboolean              : Read / Write
"tab-border"            guint                 : Write
"tab-hborder"           guint                 : Read / Write
"tab-pos"               GtkPositionType       : Read / Write
"tab-vborder"           guint                 : Read / Write
```

Child Properties

```
"menu-label"            gchararray            : Read / Write
"position"              gint                  : Read / Write
"tab-expand"            gboolean              : Read / Write
"tab-fill"              gboolean              : Read / Write
"tab-label"             gchararray            : Read / Write
"tab-pack"              GtkPackType           : Read / Write
```

Style Properties

```
"has-backward-stepper" gboolean               : Read
"has-forward-stepper"  gboolean               : Read
"has-secondary-backward-stepper" gboolean              : Read
"has-secondary-forward-stepper" gboolean              : Read
```

Signal Prototypes

```
"change-current-page"
            void        user_function      (GtkNotebook *notebook,
                                            gint arg1,
                                            gpointer user_data);
"focus-tab" gboolean    user_function      (GtkNotebook *notebook,
                                            GtkNotebookTab arg1,
                                            gpointer user_data);
"move-focus-out"
            void        user_function      (GtkNotebook *notebook,
                                            GtkDirectionType arg1,
                                            gpointer user_data);
"select-page"
            gboolean    user_function      (GtkNotebook *notebook,
                                            gboolean arg1,
                                            gpointer user_data);
"switch-page"
            void        user_function      (GtkNotebook *notebook,
                                            GtkNotebookPage *page,
                                            guint page_num,
                                            gpointer user_data);
```

Description

The GtkNotebook widget is a GtkContainer whose children are pages that can be switched between using tab labels along one edge.

There are many configuration options for GtkNotebook. Among other things, you can choose on which edge the tabs appear (see `gtk_notebook_set_tab_pos()`), whether, if there are too many tabs to fit the noteobook should be made bigger or scrolling arrows added (see gtk_notebook_set_scrollable), and whether there will be a popup menu allowing the users to switch pages. (see `gtk_notebook_enable_popup()`, `gtk_noteobook_disable_popup()`)

Details

GtkNotebook

```
typedef struct _GtkNotebook GtkNotebook;
```

GtkNotebookPage

```
typedef struct _GtkNotebookPage GtkNotebookPage;
```

The GtkNotebookPage is an opaque implementation detail of GtkNotebook.

---

gtk_notebook_new ()

```
GtkWidget*  gtk_notebook_new                (void);
```

Creates a new GtkNotebook widget with no pages.

Returns : the newly created GtkNotebook

---

gtk_notebook_append_page ()

```
gint        gtk_notebook_append_page        (GtkNotebook *notebook,
                                             GtkWidget *child,
                                             GtkWidget *tab_label);
```

Appends a page to *notebook*.

notebook :	a GtkNotebook
child :	the GtkWidget to use as the contents of the page.
tab_label :	the GtkWidget to be used as the label for the page, or NULL to use the default label, 'page N'.
Returns :	the index (starting from 0) of the appended page in the notebook, or -1 if function fails

---

gtk_notebook_append_page_menu ()

```
gint        gtk_notebook_append_page_menu   (GtkNotebook *notebook,
```

```
                                    GtkWidget *child,
                                    GtkWidget *tab_label,
                                    GtkWidget *menu_label);
```

Appends a page to *notebook*, specifying the widget to use as the label in the popup menu.

notebook :	a GtkNotebook
child :	the GtkWidget to use as the contents of the page.
tab_label :	the GtkWidget to be used as the label for the page, or NULL to use the default label, 'page N'.
menu_label :	the widget to use as a label for the page-switch menu, if that is enabled. If NULL, and *tab_label* is a GtkLabel or NULL, then the menu label will be a newly created label with the same text as *tab_label*; If *tab_label* is not a GtkLabel, *menu_label* must be specified if the page-switch menu is to be used.
Returns :	the index (starting from 0) of the appended page in the notebook, or -1 if function fails

---

gtk_notebook_prepend_page ()

```
gint        gtk_notebook_prepend_page       (GtkNotebook *notebook,
                                             GtkWidget *child,
                                             GtkWidget *tab_label);
```

Prepends a page to *notebook*.

notebook :	a GtkNotebook
child :	the GtkWidget to use as the contents of the page.
tab_label :	the GtkWidget to be used as the label for the page, or NULL to use the default label, 'page N'.
Returns :	the index (starting from 0) of the prepended page in the notebook, or -1 if function fails

---

gtk_notebook_prepend_page_menu ()

```
gint        gtk_notebook_prepend_page_menu  (GtkNotebook *notebook,
                                             GtkWidget *child,
                                             GtkWidget *tab_label,
                                             GtkWidget *menu_label);
```

Prepends a page to *notebook*, specifying the widget to use as the label in the popup menu.

notebook :	a GtkNotebook
child :	the GtkWidget to use as the contents of the page.
tab_label :	the GtkWidget to be used as the label for the page, or NULL to use the default label, 'page N'.
menu_label :	the widget to use as a label for the page-switch menu, if that is enabled. If NULL, and *tab_label* is a GtkLabel or NULL, then the menu label will be a newly created label with the same text as *tab_label*; If *tab_label* is not a GtkLabel, *menu_label* must be specified if the page-switch menu is to be used.
Returns :	the index (starting from 0) of the prepended page in the notebook, or -1 if function fails

---

gtk_notebook_insert_page ()

```
gint        gtk_notebook_insert_page        (GtkNotebook *notebook,
                                             GtkWidget *child,
                                             GtkWidget *tab_label,
                                             gint position);
```

Insert a page into *notebook* at the given position.

notebook :	a GtkNotebook
child :	the GtkWidget to use as the contents of the page.
tab_label :	the GtkWidget to be used as the label for the page, or NULL to use the default label, 'page N'.
position :	the index (starting at 0) at which to insert the page, or -1 to append the page after all other pages.
Returns :	the index (starting from 0) of the inserted page in the notebook, or -1 if function fails

---

gtk_notebook_insert_page_menu ()

```
gint        gtk_notebook_insert_page_menu   (GtkNotebook *notebook,
                                             GtkWidget *child,
                                             GtkWidget *tab_label,
                                             GtkWidget *menu_label,
```

```
                                                gint position);
```

Insert a page into *notebook* at the given position, specifying the widget to use as the label in the popup menu.

notebook :	a GtkNotebook
child :	the GtkWidget to use as the contents of the page.
tab_label :	the GtkWidget to be used as the label for the page, or NULL to use the default label, 'page N'.
menu_label :	the widget to use as a label for the page-switch menu, if that is enabled. If NULL, and *tab_label* is a GtkLabel or NULL, then the menu label will be a newly created label with the same text as *tab_label*; If *tab_label* is not a GtkLabel, *menu_label* must be specified if the page-switch menu is to be used.
position :	the index (starting at 0) at which to insert the page, or -1 to append the page after all other pages.
Returns :	the index (starting from 0) of the inserted page in the notebook, or -1 if function fails

---

gtk_notebook_remove_page ()

```
void        gtk_notebook_remove_page        (GtkNotebook *notebook,
                                             gint page_num);
```

Removes a page from the notebook given its index in the notebook.

notebook : a GtkNotebook.

page_num : the index of a notebook page, starting from 0. If -1, the last page will be removed.

---

gtk_notebook_current_page

```
#define gtk_notebook_current_page                gtk_notebook_get_current_page
```

Warning

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

Deprecated compatibility macro. Use `gtk_notebook_get_current_page()` instead.

gtk_notebook_page_num ()

```
gint        gtk_notebook_page_num           (GtkNotebook *notebook,
                                             GtkWidget *child);
```

Finds the index of the page which contains the given child widget.

notebook : a GtkNotebook
child : a GtkWidget
Returns : the index of the page containing *child*, or -1 if *child* is not in the notebook.

gtk_notebook_set_page

```
#define gtk_notebook_set_page                   gtk_notebook_set_current_page
```

Warning

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

Deprecated compatibility macro. Use `gtk_notebook_set_current_page()` instead.

gtk_notebook_next_page ()

```
void        gtk_notebook_next_page          (GtkNotebook *notebook);
```

Switches to the next page. Nothing happens if the current page is the last page.

notebook : a GtkNotebook

gtk_notebook_prev_page ()

```
void        gtk_notebook_prev_page          (GtkNotebook *notebook);
```

Switches to the previous page. Nothing happens if the current page is the first page.

notebook : a GtkNotebook

---

gtk_notebook_reorder_child ()

```
void        gtk_notebook_reorder_child      (GtkNotebook *notebook,
                                             GtkWidget *child,
                                             gint position);
```

Reorders the page containing *child*, so that it appears in position *position*. If *position* is greater than or equal to the number of children in the list or negative, *child* will be moved to the end of the list.

notebook : a GtkNotebook
child : the child to move
position : the new position, or -1 to move to the end

---

gtk_notebook_set_tab_pos ()

```
void        gtk_notebook_set_tab_pos        (GtkNotebook *notebook,
                                             GtkPositionType pos);
```

Sets the edge at which the tabs for switching pages in the notebook are drawn.

notebook : a GtkNotebook.
pos : the edge to draw the tabs at.

---

gtk_notebook_set_show_tabs ()

```
void        gtk_notebook_set_show_tabs      (GtkNotebook *notebook,
                                             gboolean show_tabs);
```

Sets whether to show the tabs for the notebook or not.

notebook : a GtkNotebook
show_tabs : TRUE if the tabs should be shown.

---

gtk_notebook_set_show_border ()

```
void        gtk_notebook_set_show_border    (GtkNotebook *notebook,
                                             gboolean show_border);
```

Sets whether a bevel will be drawn around the notebook pages. This only has a visual effect when the tabs are not shown. See gtk_notebook_set_show_tabs().

notebook : a GtkNotebook
show_border : TRUE if a bevel should be drawn around the notebook.

---

gtk_notebook_set_scrollable ()

```
void        gtk_notebook_set_scrollable     (GtkNotebook *notebook,
                                             gboolean scrollable);
```

Sets whether the tab label area will have arrows for scrolling if there are too many tabs to fit in the area.

notebook : a GtkNotebook
scrollable : TRUE if scroll arrows should be added

---

gtk_notebook_set_tab_border ()

```
void        gtk_notebook_set_tab_border     (GtkNotebook *notebook,
                                             guint border_width);
```

Warning

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

Sets the width the border around the tab labels in a notebook. This is equivalent to calling gtk_notebook_set_tab_hborder (*`notebook`*, *`border_width`*) followed by gtk_notebook_set_tab_vborder (*`notebook`*, *`border_width`*).

`notebook` : a GtkNotebook

`border_width` : width of the border around the tab labels.

---

gtk_notebook_popup_enable ()

```
void        gtk_notebook_popup_enable       (GtkNotebook *notebook);
```

Enables the popup menu: if the user clicks with the right mouse button on the bookmarks, a menu with all the pages will be popped up.

`notebook` : a GtkNotebook

---

gtk_notebook_popup_disable ()

```
void        gtk_notebook_popup_disable      (GtkNotebook *notebook);
```

Disables the popup menu.

`notebook` : a GtkNotebook

---

gtk_notebook_get_current_page ()

```
gint        gtk_notebook_get_current_page   (GtkNotebook *notebook);
```

Returns the page number of the current page.

`notebook` : a GtkNotebook

Returns : the index (starting from 0) of the current page in the notebook. If the notebook has no pages, then -1 will be returned.

---

gtk_notebook_get_menu_label ()

```
GtkWidget*  gtk_notebook_get_menu_label     (GtkNotebook *notebook,
                                             GtkWidget *child);
```

Retrieves the menu label widget of the page containing *child*.

notebook : a GtkNotebook

child : a widget contained in a page of *notebook*

Returns : the menu label, or NULL if the notebook page does not have a menu label other than the default (the tab label).

---

gtk_notebook_get_nth_page ()

```
GtkWidget*  gtk_notebook_get_nth_page       (GtkNotebook *notebook,
                                             gint page_num);
```

Returns the child widget contained in page number *page_num*.

notebook : a GtkNotebook

page_num : the index of a page in the noteobok, or -1 to get the last page.

Returns : the child widget, or NULL if *page_num* is out of bounds.

---

gtk_notebook_get_n_pages ()

```
gint        gtk_notebook_get_n_pages        (GtkNotebook *notebook);
```

Gets the number of pages in a notebook.

notebook : a GtkNotebook

Returns : the number of pages in the notebook.

Since 2.2

---

gtk_notebook_get_tab_label ()

```
GtkWidget*  gtk_notebook_get_tab_label      (GtkNotebook *notebook,
                                             GtkWidget *child);
```

Returns the tab label widget for the page *child*. NULL is returned if *child* is not in *notebook* or if no tab label has specifically been set for *child*.

notebook : a GtkNotebook
child : the page
Returns : the tab label

---

gtk_notebook_query_tab_label_packing ()

```
void        gtk_notebook_query_tab_label_packing
                                            (GtkNotebook *notebook,
                                             GtkWidget *child,
                                             gboolean *expand,
                                             gboolean *fill,
                                             GtkPackType *pack_type);
```

Query the packing attributes for the tab label of the page containing *child*.

notebook : a GtkNotebook
child : the page
expand : location to store the expand value (or NULL)
fill : location to store the fill value (or NULL)
pack_type : location to store the pack_type (or NULL)

---

gtk_notebook_set_homogeneous_tabs ()

```
void        gtk_notebook_set_homogeneous_tabs
                                            (GtkNotebook *notebook,
                                             gboolean homogeneous);
```

Warning

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

Sets whether the tabs must have all the same size or not.

notebook : a GtkNotebook
homogeneous : TRUE if all tabs should be the same size.

---

gtk_notebook_set_menu_label ()

```
void        gtk_notebook_set_menu_label     (GtkNotebook *notebook,
                                             GtkWidget *child,
                                             GtkWidget *menu_label);
```

Changes the menu label for the page containing *child*.

notebook : a GtkNotebook
child : the child widget
menu_label : the menu label, or NULL for default

---

gtk_notebook_set_menu_label_text ()

```
void        gtk_notebook_set_menu_label_text
                                            (GtkNotebook *notebook,
                                             GtkWidget *child,
                                             const gchar *menu_text);
```

Creates a new label and sets it as the menu label of *child*.

notebook : a GtkNotebook

child : the child widget
menu_text : the label text

---

gtk_notebook_set_tab_hborder ()

```
void        gtk_notebook_set_tab_hborder    (GtkNotebook *notebook,
                                             guint tab_hborder);
```

Warning

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

Sets the width of the horizontal border of tab labels.

notebook : a GtkNotebook
tab_hborder : width of the horizontal border of tab labels.

---

gtk_notebook_set_tab_label ()

```
void        gtk_notebook_set_tab_label      (GtkNotebook *notebook,
                                             GtkWidget *child,
                                             GtkWidget *tab_label);
```

Changes the tab label for *child*. If `NULL` is specified for *tab_label*, then the page will have the label 'page N'.

notebook : a GtkNotebook
child : the page
tab_label : the tab label widget to use, or `NULL` for default tab label.

---

gtk_notebook_set_tab_label_packing ()

```
void        gtk_notebook_set_tab_label_packing
                                            (GtkNotebook *notebook,
                                             GtkWidget *child,
                                             gboolean expand,
```

```
                                                gboolean fill,
                                                GtkPackType pack_type);
```

Sets the packing parameters for the tab label of the page containing *child*. See gtk_box_pack_start() for the exact meaning of the parameters.

notebook : a GtkNotebook
child : the child widget
expand : whether to expand the bookmark or not
fill : whether the bookmark should fill the allocated area or not
pack_type : the position of the bookmark

---

gtk_notebook_set_tab_label_text ()

```
void        gtk_notebook_set_tab_label_text (GtkNotebook *notebook,
                                             GtkWidget *child,
                                             const gchar *tab_text);
```

Creates a new label and sets it as the tab label for the page containing *child*.

notebook : a GtkNotebook
child : the page
tab_text : the label text

---

gtk_notebook_set_tab_vborder ()

```
void        gtk_notebook_set_tab_vborder    (GtkNotebook *notebook,
                                             guint tab_vborder);
```

Warning

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

Sets the width of the vertical border of tab labels.

notebook : a GtkNotebook
tab_vborder : width of the vertical border of tab labels.

---

gtk_notebook_get_menu_label_text ()

```
G_CONST_RETURN gchar* gtk_notebook_get_menu_label_text
                                                (GtkNotebook *notebook,
                                                 GtkWidget *child);
```

Retrieves the text of the menu label for the page containing *child*.

notebook : a GtkNotebook
child : the child widget of a page of the notebook.
Returns : value: the text of the tab label, or NULL if the widget does not have a menu label other than the default menu label, or the menu label widget is not a GtkLabel. The string is owned by the widget and must not be freed.

---

gtk_notebook_get_scrollable ()

```
gboolean    gtk_notebook_get_scrollable     (GtkNotebook *notebook);
```

Returns whether the tab label area has arrows for scrolling. See gtk_notebook_set_scrollable().

notebook : a GtkNotebook
Returns : TRUE if arrows for scrolling are present

---

gtk_notebook_get_show_border ()

```
gboolean    gtk_notebook_get_show_border    (GtkNotebook *notebook);
```

Returns whether a bevel will be drawn around the notebook pages. See gtk_notebook_set_show_border().

notebook : a GtkNotebook

Returns : TRUE if the bevel is drawn

---

gtk_notebook_get_show_tabs ()

```
gboolean    gtk_notebook_get_show_tabs      (GtkNotebook *notebook);
```

Returns whether the tabs of the notebook are shown. See `gtk_notebook_set_show_tabs()`.

notebook : a GtkNotebook
Returns : TRUE if the tabs are shown

---

gtk_notebook_get_tab_label_text ()

```
G_CONST_RETURN gchar* gtk_notebook_get_tab_label_text
                                        (GtkNotebook *notebook,
                                         GtkWidget *child);
```

Retrieves the text of the tab label for the page containing *child*.

notebook : a GtkNotebook
child : a widget contained in a page of *notebook*
Returns : value: the text of the tab label, or NULL if the tab label widget is not a GtkLabel. The string is owned by the widget and must not be freed.

---

gtk_notebook_get_tab_pos ()

```
GtkPositionType gtk_notebook_get_tab_pos    (GtkNotebook *notebook);
```

Gets the edge at which the tabs for switching pages in the notebook are drawn.

notebook : a GtkNotebook
Returns : the edge at which the tabs are drawn

---

gtk_notebook_set_current_page ()

```
void        gtk_notebook_set_current_page   (GtkNotebook *notebook,
                                             gint page_num);
```

Switches to the page number *page_num*.

notebook : a GtkNotebook

page_num : index of the page to switch to, starting from 0. If negative, the last page will be used. If greater than the number of pages in the notebook, nothing will be done.

Properties

The "enable-popup" property

```
  "enable-popup"          gboolean              : Read / Write
```

If TRUE, pressing the right mouse button on the notebook pops up a menu that you can use to go to a page.

Default value: FALSE

The "homogeneous" property

```
  "homogeneous"           gboolean              : Read / Write
```

Whether tabs should have homogeneous sizes.

Default value: FALSE

The "page" property

```
  "page"                  gint                  : Read / Write
```

The index of the current page.

Allowed values: >= 0

Default value: 0

The "scrollable" property

```
"scrollable"          gboolean              : Read / Write
```

If TRUE, scroll arrows are added if there are too many tabs to fit.

Default value: FALSE

The "show-border" property

```
"show-border"         gboolean              : Read / Write
```

Whether the border should be shown or not.

Default value: TRUE

The "show-tabs" property

```
"show-tabs"           gboolean              : Read / Write
```

Whether tabs should be shown or not.

Default value: TRUE

The "tab-border" property

```
"tab-border"          guint                 : Write
```

Width of the border around the tab labels.

Default value: 2

The "tab-hborder" property

```
"tab-hborder"         guint                 : Read / Write
```

Width of the horizontal border of tab labels.

Default value: 2

The "tab-pos" property

```
"tab-pos"             GtkPositionType       : Read / Write
```

Which side of the notebook holds the tabs.

Default value: GTK_POS_TOP

The "tab-vborder" property

```
"tab-vborder"         guint                 : Read / Write
```

Width of the vertical border of tab labels.

Default value: 2

Child Properties

The "menu-label" child property

```
"menu-label"          gchararray          : Read / Write
```

The string displayed in the child's menu entry.

Default value: NULL

The "position" child property

```
"position"            gint                : Read / Write
```

The index of the child in the parent.

Allowed values: >= -1

Default value: 0

The "tab-expand" child property

```
"tab-expand"          gboolean            : Read / Write
```

Whether to expand the child's tab or not.

Default value: TRUE

The "tab-fill" child property

```
"tab-fill"            gboolean            : Read / Write
```

Whether the child's tab should fill the allocated area or not.

Default value: TRUE

The "tab-label" child property

```
"tab-label"           gchararray           : Read / Write
```

The string displayed on the child's tab label.

Default value: NULL

The "tab-pack" child property

```
"tab-pack"            GtkPackType          : Read / Write
```

A GtkPackType indicating whether the child is packed with reference to the start or end of the parent.

Default value: GTK_PACK_START

Style Properties

The "has-backward-stepper" style property

```
"has-backward-stepper" gboolean            : Read
```

The "has-backward-stepper" property determines whether the standard backward arrow button is displayed.

Default value: TRUE

Since 2.4

The "has-forward-stepper" style property

```
  "has-forward-stepper"  gboolean              : Read
```

The "has-forward-stepper" property determines whether the standard forward arrow button is displayed.

Default value: TRUE

Since 2.4

The "has-secondary-backward-stepper" style property

```
  "has-secondary-backward-stepper" gboolean              : Read
```

The "has-secondary-backward-stepper" property determines whether a second backward arrow button is displayed on the opposite end of the tab area.

Default value: FALSE

Since 2.4

The "has-secondary-forward-stepper" style property

```
  "has-secondary-forward-stepper" gboolean              : Read
```

The "has-secondary-forward-stepper" property determines whether a second forward arrow button is displayed on the opposite end of the tab area.

Default value: FALSE

Since 2.4

Signals

The "change-current-page" signal

```
void        user_function                  (GtkNotebook *notebook,
```

```
                                        gint arg1,
                                        gpointer user_data);
```

notebook : the object which received the signal.
arg1 :
user_data : user data set when the signal handler was connected.

The "focus-tab" signal

```
gboolean    user_function               (GtkNotebook *notebook,
                                        GtkNotebookTab arg1,
                                        gpointer user_data);
```

notebook : the object which received the signal.
arg1 :
user_data : user data set when the signal handler was connected.
Returns :

The "move-focus-out" signal

```
void        user_function               (GtkNotebook *notebook,
                                        GtkDirectionType arg1,
                                        gpointer user_data);
```

notebook : the object which received the signal.
arg1 :
user_data : user data set when the signal handler was connected.

The "select-page" signal

```
gboolean    user_function               (GtkNotebook *notebook,
                                        gboolean arg1,
                                        gpointer user_data);
```

notebook : the object which received the signal.
arg1 :
user_data : user data set when the signal handler was connected.
Returns :

The "switch-page" signal

```
void        user_function                   (GtkNotebook *notebook,
                                             GtkNotebookPage *page,
                                             guint page_num,
                                             gpointer user_data);
```

Emitted when the user or a function changes the current page.

notebook : the object which received the signal.
page : the new current page
page_num : the index of the page
user_data : user data set when the signal handler was connected.

See Also

GtkContainer For functions that apply to every GtkContainer (like GtkList).

<< GtkLayout **GtkTable >>**

GtkTable

GtkTable — Pack widgets in regular patterns

Synopsis

```
#include <gtk/gtk.h>

            GtkTable;
            GtkTableChild;
            GtkTableRowCol;
GtkWidget*  gtk_table_new                   (guint rows,
                                             guint columns,
                                             gboolean homogeneous);
void        gtk_table_resize                (GtkTable *table,
                                             guint rows,
                                             guint columns);
void        gtk_table_attach                (GtkTable *table,
                                             GtkWidget *child,
                                             guint left_attach,
                                             guint right_attach,
                                             guint top_attach,
                                             guint bottom_attach,
                                             GtkAttachOptions xoptions,
                                             GtkAttachOptions yoptions,
                                             guint xpadding,
                                             guint ypadding);
void        gtk_table_attach_defaults       (GtkTable *table,
                                             GtkWidget *widget,
                                             guint left_attach,
                                             guint right_attach,
                                             guint top_attach,
```

```
                                                guint bottom_attach);
void        gtk_table_set_row_spacing       (GtkTable *table,
                                                guint row,
                                                guint spacing);
void        gtk_table_set_col_spacing       (GtkTable *table,
                                                guint column,
                                                guint spacing);
void        gtk_table_set_row_spacings      (GtkTable *table,
                                                guint spacing);
void        gtk_table_set_col_spacings      (GtkTable *table,
                                                guint spacing);
void        gtk_table_set_homogeneous       (GtkTable *table,
                                                gboolean homogeneous);
guint       gtk_table_get_default_row_spacing
                                            (GtkTable *table);
gboolean    gtk_table_get_homogeneous       (GtkTable *table);
guint       gtk_table_get_row_spacing       (GtkTable *table,
                                                guint row);
guint       gtk_table_get_col_spacing       (GtkTable *table,
                                                guint column);
guint       gtk_table_get_default_col_spacing
                                            (GtkTable *table);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkWidget
               +----GtkContainer
                     +----GtkTable
```

Implemented Interfaces

GtkTable implements AtkImplementorIface.

Properties

```
"column-spacing"       guint                : Read / Write
"homogeneous"          gboolean             : Read / Write
"n-columns"            guint                : Read / Write
"n-rows"               guint                : Read / Write
"row-spacing"          guint                : Read / Write
```

Child Properties

```
"bottom-attach"        guint                : Read / Write
"left-attach"          guint                : Read / Write
"right-attach"         guint                : Read / Write
"top-attach"           guint                : Read / Write
"x-options"            GtkAttachOptions     : Read / Write
"x-padding"            guint                : Read / Write
"y-options"            GtkAttachOptions     : Read / Write
"y-padding"            guint                : Read / Write
```

Description

The GtkTable functions allow the programmer to arrange widgets in rows and columns, making it easy to align many widgets next to each other, horizontally and vertically.

Tables are created with a call to `gtk_table_new()`, the size of which can later be changed with `gtk_table_resize()`.

Widgets can be added to a table using `gtk_table_attach()` or the more convenient (but slightly less flexible) `gtk_table_attach_defaults()`.

To alter the space next to a specific row, use `gtk_table_set_row_spacing()`, and for a column, `gtk_table_set_col_spacing()`.

The gaps between *all* rows or columns can be changed by calling `gtk_table_set_row_spacings()`

or gtk_table_set_col_spacings() respectively.

gtk_table_set_homogeneous(), can be used to set whether all cells in the table will resize themselves to the size of the largest widget in the table.

Details

GtkTable

```
typedef struct _GtkTable GtkTable;
```

The GtkTable structure holds the data for the actual table itself. *children* is a GList of all the widgets the table contains. *rows* and *columns* are pointers to GtkTableRowCol structures, which contain the default spacing and expansion details for the GtkTable's rows and columns, respectively.

nrows and *ncols* are 16bit integers storing the number of rows and columns the table has.

GtkTableChild

```
typedef struct {
  GtkWidget *widget;
  guint16 left_attach;
  guint16 right_attach;
  guint16 top_attach;
  guint16 bottom_attach;
  guint16 xpadding;
  guint16 ypadding;
  guint xexpand : 1;
  guint yexpand : 1;
  guint xshrink : 1;
  guint yshrink : 1;
  guint xfill : 1;
  guint yfill : 1;
} GtkTableChild;
```

The *widget* field is a pointer to the widget that this GtkTableChild structure is keeping track of. The

left_attach, *right_attach*, *top_attach*, and *bottom_attach* fields specify the row and column numbers which make up the invisible rectangle that the child widget is packed into.

xpadding and *ypadding* specify the space between this widget and the surrounding table cells.

GtkTableRowCol

```
typedef struct {
  guint16 requisition;
  guint16 allocation;
  guint16 spacing;
  guint need_expand : 1;
  guint need_shrink : 1;
  guint expand : 1;
  guint shrink : 1;
  guint empty : 1;
} GtkTableRowCol;
```

These fields should be considered read-only and not be modified directly.

---

gtk_table_new ()

```
GtkWidget*  gtk_table_new                       (guint rows,
                                                 guint columns,
                                                 gboolean homogeneous);
```

Used to create a new table widget. An initial size must be given by specifying how many rows and columns the table should have, although this can be changed later with gtk_table_resize(). *rows* and *columns* must both be in the range 0 .. 65535.

rows :	The number of rows the new table should have.
columns :	The number of columns the new table should have.
homogeneous :	If set to TRUE, all table cells are resized to the size of the cell containing the largest widget.

Returns : A pointer to the the newly created table widget.

---

gtk_table_resize ()

```
void        gtk_table_resize                    (GtkTable *table,
                                                 guint rows,
                                                 guint columns);
```

If you need to change a table's size *after* it has been created, this function allows you to do so.

table : The GtkTable you wish to change the size of.
rows : The new number of rows.
columns : The new number of columns.

---

gtk_table_attach ()

```
void        gtk_table_attach                    (GtkTable *table,
                                                 GtkWidget *child,
                                                 guint left_attach,
                                                 guint right_attach,
                                                 guint top_attach,
                                                 guint bottom_attach,
                                                 GtkAttachOptions xoptions,
                                                 GtkAttachOptions yoptions,
                                                 guint xpadding,
                                                 guint ypadding);
```

Adds a widget to a table. The number of 'cells' that a widget will occupy is specified by *left_attach*, *right_attach*, *top_attach* and *bottom_attach*. These each represent the leftmost, rightmost, uppermost and lowest column and row numbers of the table. (Columns and rows are indexed from zero).

table : The GtkTable to add a new widget to.
child : The widget to add.

left_attach :	the column number to attach the left side of a child widget to.
right_attach :	the column number to attach the right side of a child widget to.
top_attach :	the row number to attach the top of a child widget to.
bottom_attach :	the row number to attach the bottom of a child widget to.
xoptions :	Used to specify the properties of the child widget when the table is resized.
yoptions :	The same as xoptions, except this field determines behaviour of vertical resizing.
xpadding :	An integer value specifying the padding on the left and right of the widget being added to the table.
ypadding :	The amount of padding above and below the child widget.

---

gtk_table_attach_defaults ()

```
void        gtk_table_attach_defaults       (GtkTable *table,
                                             GtkWidget *widget,
                                             guint left_attach,
                                             guint right_attach,
                                             guint top_attach,
                                             guint bottom_attach);
```

As there are many options associated with `gtk_table_attach()`, this convenience function provides the programmer with a means to add children to a table with identical padding and expansion options. The values used for the GtkAttachOptions are `GTK_EXPAND | GTK_FILL`, and the padding is set to 0.

table :	The table to add a new child widget to.
widget :	The child widget to add.
left_attach :	The column number to attach the left side of the child widget to.
right_attach :	The column number to attach the right side of the child widget to.
top_attach :	The row number to attach the top of the child widget to.
bottom_attach :	The row number to attach the bottom of the child widget to.

---

gtk_table_set_row_spacing ()

```
void        gtk_table_set_row_spacing       (GtkTable *table,
                                             guint row,
                                             guint spacing);
```

Changes the space between a given table row and its surrounding rows.

table : a GtkTable containing the row whose properties you wish to change.
row : row number whose spacing will be changed.
spacing : number of pixels that the spacing should take up.

---

gtk_table_set_col_spacing ()

```
void        gtk_table_set_col_spacing       (GtkTable *table,
                                             guint column,
                                             guint spacing);
```

Alters the amount of space between a given table column and the adjacent columns.

table : a GtkTable.
column : the column whose spacing should be changed.
spacing : number of pixels that the spacing should take up.

---

gtk_table_set_row_spacings ()

```
void        gtk_table_set_row_spacings      (GtkTable *table,
                                             guint spacing);
```

Sets the space between every row in *table* equal to *spacing*.

table : a GtkTable.
spacing : the number of pixels of space to place between every row in the table.

---

gtk_table_set_col_spacings ()

```
void        gtk_table_set_col_spacings      (GtkTable *table,
                                             guint spacing);
```

Sets the space between every column in *table* equal to *spacing*.

table : a GtkTable.

spacing : the number of pixels of space to place between every column in the table.

---

gtk_table_set_homogeneous ()

```
void        gtk_table_set_homogeneous       (GtkTable *table,
                                             gboolean homogeneous);
```

Changes the homogenous property of table cells, ie. whether all cells are an equal size or not.

table : The GtkTable you wish to set the homogeneous properties of.

homogeneous : Set to TRUE to ensure all table cells are the same size. Set to FALSE if this is not your desired behaviour.

---

gtk_table_get_default_row_spacing ()

```
guint       gtk_table_get_default_row_spacing
                                            (GtkTable *table);
```

Gets the default row spacing for the table. This is the spacing that will be used for newly added rows. (See gtk_table_set_row_spacings())

table : a GtkTable

Returns : value: the default row spacing

gtk_table_get_homogeneous ()

```
gboolean    gtk_table_get_homogeneous       (GtkTable *table);
```

Returns whether the table cells are all constrained to the same width and height. (See `gtk_table_set_homogenous()`)

table : a GtkTable

Returns : TRUE if the cells are all constrained to the same size

gtk_table_get_row_spacing ()

```
guint       gtk_table_get_row_spacing       (GtkTable *table,
                                             guint row);
```

Gets the amount of space between row *row*, and row *row* + 1. See `gtk_table_set_row_spacing()`.

table : a GtkTable

row : a row in the table, 0 indicates the first row

Returns : the row spacing

gtk_table_get_col_spacing ()

```
guint       gtk_table_get_col_spacing       (GtkTable *table,
                                             guint column);
```

Gets the amount of space between column *col*, and column *col* + 1. See `gtk_table_set_col_spacing()`.

table : a GtkTable

column : a column in the table, 0 indicates the first column

Returns : the column spacing

---

gtk_table_get_default_col_spacing ()

```
guint       gtk_table_get_default_col_spacing
                                            (GtkTable *table);
```

Gets the default column spacing for the table. This is the spacing that will be used for newly added columns. (See `gtk_table_set_col_spacings()`)

table : a GtkTable

Returns : value: the default column spacing

Properties

The "column-spacing" property

```
  "column-spacing"       guint                : Read / Write
```

The amount of space between two consecutive columns.

Default value: 0

The "homogeneous" property

```
  "homogeneous"          gboolean             : Read / Write
```

If TRUE this means the table cells are all the same width/height.

Default value: FALSE

The "n-columns" property

```
"n-columns"            guint                : Read / Write
```

The number of columns in the table.

Default value: 0

The "n-rows" property

```
"n-rows"               guint                : Read / Write
```

The number of rows in the table.

Default value: 0

The "row-spacing" property

```
"row-spacing"          guint                : Read / Write
```

The amount of space between two consecutive rows.

Default value: 0

Child Properties

The "bottom-attach" child property

```
  "bottom-attach"        guint                 : Read / Write
```

The row number to attach the bottom of the child to.

Allowed values: [1,65535]

Default value: 1

The "left-attach" child property

```
  "left-attach"          guint                 : Read / Write
```

The column number to attach the left side of the child to.

Allowed values: <= 65535

Default value: 0

The "right-attach" child property

```
  "right-attach"         guint                 : Read / Write
```

The column number to attach the right side of a child widget to.

Allowed values: [1,65535]

Default value: 1

The "top-attach" child property

```
  "top-attach"           guint                 : Read / Write
```

The row number to attach the top of a child widget to.

Allowed values: <= 65535

Default value: 0

The "x-options" child property

```
  "x-options"            GtkAttachOptions      : Read / Write
```

Options specifying the horizontal behaviour of the child.

Default value: GTK_EXPAND|GTK_FILL

The "x-padding" child property

```
  "x-padding"            guint                 : Read / Write
```

Extra space to put between the child and its left and right neighbors, in pixels.

Allowed values: <= 65535

Default value: 0

The "y-options" child property

```
  "y-options"            GtkAttachOptions      : Read / Write
```

Options specifying the vertical behaviour of the child.

Default value: GTK_EXPAND|GTK_FILL

The "y-padding" child property

"y-padding"	guint	: Read / Write

Extra space to put between the child and its upper and lower neighbors, in pixels.

Allowed values: <= 65535

Default value: 0

See Also

GtkVBox For packing widgets vertically only.

GtkHBox For packing widgets horizontally only.

<< GtkNotebook **GtkExpander >>**

GtkExpander

GtkExpander — A container which can hide its child

Synopsis

```
#include <gtk/gtk.h>

            GtkExpander;
GtkWidget*  gtk_expander_new                (const gchar *label);
GtkWidget*  gtk_expander_new_with_mnemonic  (const gchar *label);
void        gtk_expander_set_expanded       (GtkExpander *expander,
                                             gboolean expanded);
gboolean    gtk_expander_get_expanded       (GtkExpander *expander);
void        gtk_expander_set_spacing        (GtkExpander *expander,
                                             gint spacing);
gint        gtk_expander_get_spacing        (GtkExpander *expander);
void        gtk_expander_set_label          (GtkExpander *expander,
                                             const gchar *label);
G_CONST_RETURN gchar* gtk_expander_get_label
                                            (GtkExpander *expander);
void        gtk_expander_set_use_underline  (GtkExpander *expander,
                                             gboolean use_underline);
gboolean    gtk_expander_get_use_underline  (GtkExpander *expander);
void        gtk_expander_set_use_markup     (GtkExpander *expander,
                                             gboolean use_markup);
gboolean    gtk_expander_get_use_markup     (GtkExpander *expander);
void        gtk_expander_set_label_widget   (GtkExpander *expander,
                                             GtkWidget *label_widget);
GtkWidget*  gtk_expander_get_label_widget   (GtkExpander *expander);
```

Object Hierarchy

```
GObject
 +----GtkObject
       +----GtkWidget
             +----GtkContainer
                   +----GtkBin
                         +----GtkExpander
```

Implemented Interfaces

GtkExpander implements AtkImplementorIface.

Properties

```
"expanded"              gboolean              : Read / Write / Construct
"label"                 gchararray            : Read / Write / Construct
"label-widget"          GtkWidget             : Read / Write
"spacing"               gint                  : Read / Write
"use-markup"            gboolean              : Read / Write / Construct
"use-underline"         gboolean              : Read / Write / Construct
```

Style Properties

```
"expander-size"         gint                  : Read
"expander-spacing"      gint                  : Read
```

Signal Prototypes

```
"activate"  void        user_function      (GtkExpander *expander,
                                            gpointer user_data);
```

Description

A GtkExpander allows the user to hide or show its child by clicking on an expander triangle similar to the triangles used in a GtkTreeView.

Normally you use an expander as you would use any other descendant of GtkBin; you create the child widget and use `gtk_container_add()` to add it to the expander. When the expander is toggled, it will take care of showing and hiding the child automatically.

Special Usage

There there are situations in which you may prefer to show and hide the expanded widget yourself, such as when you want to actually create the widget at expansion time. In this case, create a GtkExpander but do not add a child to it. The expander widget has an `expanded` property which can be used to monitor its expansion state. You should watch this property with a signal connection as follows:

```
expander = gtk_expander_new_with_mnemonic ("_More Options");
g_signal_connect (expander, "notify::expanded",
                  G_CALLBACK (expander_callback), NULL);

...

static void
expander_callback (GObject    *object,
                   GParamSpec *param_spec,
                   gpointer    user_data)
{
  GtkExpander *expander;

  expander = GTK_EXPANDER (object);

  if (gtk_expander_get_expanded (expander))
    {
      /* Show or create widgets */
    }
  else
    {
      /* Hide or destroy widgets */
    }
}
```

Details

GtkExpander

```
typedef struct _GtkExpander GtkExpander;
```

---

gtk_expander_new ()

```
GtkWidget*  gtk_expander_new                (const gchar *label);
```

Creates a new expander using *label* as the text of the label.

label : the text of the label
Returns : a new GtkExpander widget.

Since 2.4

---

gtk_expander_new_with_mnemonic ()

```
GtkWidget*  gtk_expander_new_with_mnemonic  (const gchar *label);
```

Creates a new expander using *label* as the text of the label. If characters in *label* are preceded by an underscore, they are underlined. If you need a literal underscore character in a label, use '__' (two underscores). The first underlined character represents a keyboard accelerator called a mnemonic. Pressing Alt and that key activates the button.

label : the text of the label with an underscore in front of the mnemonic character
Returns : a new GtkExpander widget.

Since 2.4

---

gtk_expander_set_expanded ()

```
void        gtk_expander_set_expanded       (GtkExpander *expander,
                                             gboolean expanded);
```

Sets the state of the expander. Set to TRUE, if you want the child widget to be revealed, and FALSE if you want the child widget to be hidden.

expander : a GtkExpander

expanded : whether the child widget is revealed

Since 2.4

---

gtk_expander_get_expanded ()

```
gboolean    gtk_expander_get_expanded       (GtkExpander *expander);
```

Queries a GtkExpander and returns its current state. Returns TRUE if the child widget is revealed.

See gtk_expander_set_expanded().

expander : a GtkExpander

Returns : the current state of the expander.

Since 2.4

---

gtk_expander_set_spacing ()

```
void        gtk_expander_set_spacing        (GtkExpander *expander,
                                             gint spacing);
```

Sets the spacing field of *expander*, which is the number of pixels to place between expander and the child.

expander : a GtkExpander
spacing : distance between the expander and child in pixels.

Since 2.4

---

gtk_expander_get_spacing ()

```
gint        gtk_expander_get_spacing        (GtkExpander *expander);
```

Gets the value set by gtk_expander_set_spacing().

expander : a GtkExpander
Returns : spacing between the expander and child.

Since 2.4

---

gtk_expander_set_label ()

```
void        gtk_expander_set_label          (GtkExpander *expander,
                                             const gchar *label);
```

Sets the text of the label of the expander to *label*.

This will also clear any previously set labels.

expander : a GtkExpander
label : a string

Since 2.4

---

gtk_expander_get_label ()

```
G_CONST_RETURN gchar* gtk_expander_get_label
                                                (GtkExpander *expander);
```

Fetches the text from the label of the expander, as set by `gtk_expander_set_label()`. If the label text has not been set the return value will be `NULL`. This will be the case if you create an empty button with `gtk_button_new()` to use as a container.

expander : a GtkExpander

Returns : The text of the label widget. This string is owned by the widget and must not be modified or freed.

Since 2.4

---

gtk_expander_set_use_underline ()

```
void        gtk_expander_set_use_underline  (GtkExpander *expander,
                                             gboolean use_underline);
```

If true, an underline in the text of the expander label indicates the next character should be used for the mnemonic accelerator key.

expander : a GtkExpander

use_underline : `TRUE` if underlines in the text indicate mnemonics

Since 2.4

---

gtk_expander_get_use_underline ()

```
gboolean    gtk_expander_get_use_underline  (GtkExpander *expander);
```

Returns whether an embedded underline in the expander label indicates a mnemonic. See gtk_expander_set_use_underline().

expander : a GtkExpander

Returns : TRUE if an embedded underline in the expander label indicates the mnemonic accelerator keys.

Since 2.4

---

gtk_expander_set_use_markup ()

```
void        gtk_expander_set_use_markup     (GtkExpander *expander,
                                             gboolean use_markup);
```

Sets whether the text of the label contains markup in Pango's text markup language. See gtk_label_set_markup().

expander : a GtkExpander

use_markup : TRUE if the label's text should be parsed for markup

Since 2.4

---

gtk_expander_get_use_markup ()

```
gboolean    gtk_expander_get_use_markup     (GtkExpander *expander);
```

Returns whether the label's text is interpreted as marked up with the Pango text markup language. See gtk_expander_set_use_markup().

expander : a GtkExpander

Returns : TRUE if the label's text will be parsed for markup

Since 2.4

---

gtk_expander_set_label_widget ()

```
void        gtk_expander_set_label_widget   (GtkExpander *expander,
                                             GtkWidget *label_widget);
```

Set the label widget for the expander. This is the widget that will appear embedded alongside the expander arrow.

expander : a GtkExpander

label_widget : the new label widget

Since 2.4

---

gtk_expander_get_label_widget ()

```
GtkWidget*  gtk_expander_get_label_widget   (GtkExpander *expander);
```

Retrieves the label widget for the frame. See gtk_expander_set_label_widget().

expander : a GtkExpander

Returns : the label widget, or NULL if there is none.

Since 2.4

Properties

The "expanded" property

"expanded"	gboolean	: Read / Write / Construct

Whether the expander has been opened to reveal the child widget.

Default value: FALSE

The "label" property

"label"	gchararray	: Read / Write / Construct

Text of the expander's label.

Default value: NULL

The "label-widget" property

"label-widget"	GtkWidget	: Read / Write

A widget to display in place of the usual expander label.

The "spacing" property

"spacing"	gint	: Read / Write

Space to put between the label and the child.

Allowed values: >= 0

Default value: 0

The "use-markup" property

```
"use-markup"           gboolean              : Read / Write / Construct
```

The text of the label includes XML markup. See pango_parse_markup().

Default value: FALSE

The "use-underline" property

```
"use-underline"        gboolean              : Read / Write / Construct
```

If set, an underline in the text indicates the next character should be used for the mnemonic accelerator key.

Default value: FALSE

Style Properties

The "expander-size" style property

```
"expander-size"        gint                  : Read
```

Size of the expander arrow.

Allowed values: >= 0

Default value: 10

The "expander-spacing" style property

```
  "expander-spacing"     gint                  : Read
```

Spacing around expander arrow.

Allowed values: >= 0

Default value: 2

Signals

The "activate" signal

```
void        user_function                  (GtkExpander *expander,
                                            gpointer user_data);
```

expander : the object which received the signal.

user_data : user data set when the signal handler was connected.

<< GtkTable **Ornaments >>**

Ornaments

GtkFrame - A bin with a decorative frame and optional label
GtkHSeparator - A horizontal separator
GtkVSeparator - A vertical separator

<< GtkExpander **GtkFrame >>**

GtkFrame

GtkFrame — A bin with a decorative frame and optional label

Synopsis

```
#include <gtk/gtk.h>


            GtkFrame;
GtkWidget*  gtk_frame_new                   (const gchar *label);
void        gtk_frame_set_label             (GtkFrame *frame,
                                             const gchar *label);
void        gtk_frame_set_label_widget      (GtkFrame *frame,
                                             GtkWidget *label_widget);
void        gtk_frame_set_label_align       (GtkFrame *frame,
                                             gfloat xalign,
                                             gfloat yalign);
void        gtk_frame_set_shadow_type       (GtkFrame *frame,
                                             GtkShadowType type);
G_CONST_RETURN gchar* gtk_frame_get_label   (GtkFrame *frame);
void        gtk_frame_get_label_align       (GtkFrame *frame,
                                             gfloat *xalign,
                                             gfloat *yalign);
GtkWidget*  gtk_frame_get_label_widget      (GtkFrame *frame);
GtkShadowType gtk_frame_get_shadow_type     (GtkFrame *frame);
```

Object Hierarchy

```
GObject
 +----GtkObject
       +----GtkWidget
             +----GtkContainer
                   +----GtkBin
                         +----GtkFrame
                               +----GtkAspectFrame
```

Implemented Interfaces

GtkFrame implements AtkImplementorIface.

Properties

```
"label"                 gchararray            : Read / Write
"label-widget"          GtkWidget             : Read / Write
"label-xalign"          gfloat                : Read / Write
"label-yalign"          gfloat                : Read / Write
"shadow"                GtkShadowType         : Read / Write
"shadow-type"           GtkShadowType         : Read / Write
```

Description

The frame widget is a Bin that surrounds its child with a decorative frame and an optional label. If present, the label is drawn in a gap in the top side of the frame. The position of the label can be controlled with `gtk_frame_set_label_align()`.

Details

GtkFrame

```
typedef struct _GtkFrame GtkFrame;
```

---

gtk_frame_new ()

```
GtkWidget*  gtk_frame_new                   (const gchar *label);
```

Creates a new GtkFrame, with optional label *label*. If *label* is NULL, the label is omitted.

label : the text to use as the label of the frame
Returns : a new GtkFrame widget

---

gtk_frame_set_label ()

```
void        gtk_frame_set_label             (GtkFrame *frame,
                                             const gchar *label);
```

Sets the text of the label. If *label* is NULL, the current label is removed.

frame : a GtkFrame
label : the text to use as the label of the frame

---

gtk_frame_set_label_widget ()

```
void        gtk_frame_set_label_widget      (GtkFrame *frame,
                                             GtkWidget *label_widget);
```

Sets the label widget for the frame. This is the widget that will appear embedded in the top edge of the frame as a title.

frame : a GtkFrame
label_widget : the new label widget

---

gtk_frame_set_label_align ()

```
void        gtk_frame_set_label_align       (GtkFrame *frame,
                                             gfloat xalign,
                                             gfloat yalign);
```

Sets the alignment of the frame widget's label. The default values for a newly created frame are 0.0 and 0.5.

frame : a GtkFrame

xalign : The position of the label along the top edge of the widget. A value of 0.0 represents left alignment; 1.0 represents right alignment.

yalign : The y alignment of the label. A value of 0.0 aligns under the frame; 1.0 aligns above the frame.

---

gtk_frame_set_shadow_type ()

```
void        gtk_frame_set_shadow_type       (GtkFrame *frame,
                                             GtkShadowType type);
```

Sets the shadow type for *frame*.

frame : a GtkFrame

type : the new GtkShadowType

---

gtk_frame_get_label ()

```
G_CONST_RETURN gchar* gtk_frame_get_label   (GtkFrame *frame);
```

If the frame's label widget is a GtkLabel, returns the text in the label widget. (The frame will have a GtkLabel for the label widget if a non-NULL argument was passed to gtk_frame_new().)

frame : a GtkFrame

Returns : the text in the label, or NULL if there was no label widget or the lable widget was not a GtkLabel. This string is owned by GTK+ and must not be modified or freed.

---

gtk_frame_get_label_align ()

```
void        gtk_frame_get_label_align       (GtkFrame *frame,
                                             gfloat *xalign,
                                             gfloat *yalign);
```

Retrieves the X and Y alignment of the frame's label. See gtk_frame_set_label_align().

frame : a GtkFrame

xalign : location to store X alignment of frame's label, or NULL

yalign : location to store X alignment of frame's label, or NULL

---

gtk_frame_get_label_widget ()

```
GtkWidget*  gtk_frame_get_label_widget      (GtkFrame *frame);
```

Retrieves the label widget for the frame. See gtk_frame_set_label_widget().

frame : a GtkFrame

Returns : the label widget, or NULL if there is none.

---

gtk_frame_get_shadow_type ()

```
GtkShadowType gtk_frame_get_shadow_type     (GtkFrame *frame);
```

Retrieves the shadow type of the frame. See gtk_frame_set_shadow_type().

frame : a GtkFrame

Returns : the current shadow type of the frame.

Properties

The "label" property

```
"label"                gchararray            : Read / Write
```

Text of the frame's label.

Default value: NULL

The "label-widget" property

```
"label-widget"         GtkWidget             : Read / Write
```

A widget to display in place of the usual frame label.

The "label-xalign" property

```
"label-xalign"         gfloat                : Read / Write
```

The horizontal alignment of the label.

Allowed values: [0,1]

Default value: 0.5

The "label-yalign" property

"label-yalign"	gfloat	: Read / Write

The vertical alignment of the label.

Allowed values: [0,1]

Default value: 0.5

The "shadow" property

"shadow"	GtkShadowType	: Read / Write

Deprecated property, use shadow_type instead.

Default value: GTK_SHADOW_ETCHED_IN

The "shadow-type" property

"shadow-type"	GtkShadowType	: Read / Write

Appearance of the frame border.

Default value: GTK_SHADOW_ETCHED_IN

<< Ornaments **GtkHSeparator >>**

GtkHSeparator

GtkHSeparator — A horizontal separator

Synopsis

```
#include <gtk/gtk.h>

            GtkHSeparator;
GtkWidget*  gtk_hseparator_new              (void);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkWidget
               +----GtkSeparator
                     +----GtkHSeparator
```

Implemented Interfaces

GtkHSeparator implements AtkImplementorIface.

Description

The GtkHSeparator widget is a horizontal separator, used to group the widgets within a window. It

displays a horizontal line with a shadow to make it appear sunken into the interface.

Note

The GtkHSeparator widget is not used as a separator within menus. To create a separator in a menu create an empty GtkSeparatorMenuItem widget using `gtk_separator_menu_item_new()` and add it to the menu with `gtk_menu_shell_append()`.

Details

GtkHSeparator

```
typedef struct _GtkHSeparator GtkHSeparator;
```

The GtkHSeparator-struct struct contains private data only, and should be accessed using the functions below.

---

gtk_hseparator_new ()

```
GtkWidget*  gtk_hseparator_new              (void);
```

Creates a new GtkHSeparator.

Returns : a new GtkHSeparator.

See Also

GtkVSeparator a vertical separator.

<< GtkFrame **GtkVSeparator >>**

GtkVSeparator

GtkVSeparator — A vertical separator

Synopsis

```
#include <gtk/gtk.h>

            GtkVSeparator;
GtkWidget*  gtk_vseparator_new              (void);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkWidget
               +----GtkSeparator
                     +----GtkVSeparator
```

Implemented Interfaces

GtkVSeparator implements AtkImplementorIface.

Description

The GtkVSeparator widget is a vertical separator, used to group the widgets within a window. It displays

a vertical line with a shadow to make it appear sunken into the interface.

Details

GtkVSeparator

```
typedef struct _GtkVSeparator GtkVSeparator;
```

The GtkVSeparator-struct struct contains private data only, and should be accessed using the functions below.

---

gtk_vseparator_new ()

```
GtkWidget*  gtk_vseparator_new              (void);
```

Creates a new GtkVSeparator.

Returns : a new GtkVSeparator.

See Also

GtkHSeparator a horizontal separator.

<< GtkHSeparator **Scrolling >>**

Scrolling

GtkHScrollbar - A horizontal scrollbar
GtkVScrollbar - A vertical scrollbar
GtkScrolledWindow - Adds scrollbars to its child widget

<< GtkVSeparator **GtkHScrollbar >>**

GtkHScrollbar

GtkHScrollbar — A horizontal scrollbar

Synopsis

```
#include <gtk/gtk.h>

            GtkHScrollbar;
GtkWidget*  gtk_hscrollbar_new              (GtkAdjustment *adjustment);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkWidget
               +----GtkRange
                     +----GtkScrollbar
                           +----GtkHScrollbar
```

Implemented Interfaces

GtkHScrollbar implements AtkImplementorIface.

Description

The GtkHScrollbar widget is a widget arranged horizontally creating a scrollbar. See GtkScrollbar for details

on scrollbars. GtkAdjustment pointers may be added to handle the adjustment of the scrollbar or it may be left NULL in which case one will be created for you. See GtkAdjustment for details.

Details

GtkHScrollbar

```
typedef struct _GtkHScrollbar GtkHScrollbar;
```

The GtkHScrollbar struct contains private data and should be accessed using the functions below.

---

gtk_hscrollbar_new ()

```
GtkWidget*  gtk_hscrollbar_new              (GtkAdjustment *adjustment);
```

Creates a new horizontal scrollbar.

adjustment : the GtkAdjustment to use, or NULL to create a new adjustment.

Returns : the new GtkHScrollbar.

See Also

GtkScrollbar, GtkScrolledWindow

<< Scrolling **GtkVScrollbar >>**

GtkVScrollbar

GtkVScrollbar — A vertical scrollbar

Synopsis

```
#include <gtk/gtk.h>

            GtkVScrollbar;
GtkWidget*  gtk_vscrollbar_new              (GtkAdjustment *adjustment);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkWidget
               +----GtkRange
                     +----GtkScrollbar
                           +----GtkVScrollbar
```

Implemented Interfaces

GtkVScrollbar implements AtkImplementorIface.

Description

The GtkVScrollbar widget is a widget arranged vertically creating a scrollbar. See GtkScrollbar for details on

scrollbars. GtkAdjustment pointers may be added to handle the adjustment of the scrollbar or it may be left NULL in which case one will be created for you. See GtkAdjustment for details.

Details

GtkVScrollbar

```
typedef struct _GtkVScrollbar GtkVScrollbar;
```

The GtkVScrollbar struct contains private data and should be accessed using the functions below.

---

gtk_vscrollbar_new ()

```
GtkWidget*  gtk_vscrollbar_new              (GtkAdjustment *adjustment);
```

Creates a new vertical scrollbar.

adjustment : the GtkAdjustment to use, or NULL to create a new adjustment.

Returns : the new GtkVScrollbar

See Also

GtkScrollbar, GtkScrolledWindow

GtkScrolledWindow

GtkScrolledWindow — Adds scrollbars to its child widget

Synopsis

```
#include <gtk/gtk.h>

            GtkScrolledWindow;
GtkWidget*  gtk_scrolled_window_new         (GtkAdjustment *hadjustment,
                                             GtkAdjustment *vadjustment);
GtkAdjustment* gtk_scrolled_window_get_hadjustment
                                            (GtkScrolledWindow *scrolled_window);
GtkAdjustment* gtk_scrolled_window_get_vadjustment
                                            (GtkScrolledWindow *scrolled_window);
void        gtk_scrolled_window_set_policy  (GtkScrolledWindow *scrolled_window,
                                             GtkPolicyType hscrollbar_policy,
                                             GtkPolicyType vscrollbar_policy);
void        gtk_scrolled_window_add_with_viewport
                                            (GtkScrolledWindow *scrolled_window,
                                             GtkWidget *child);
void        gtk_scrolled_window_set_placement
                                            (GtkScrolledWindow *scrolled_window,
                                             GtkCornerType window_placement);
void        gtk_scrolled_window_set_shadow_type
                                            (GtkScrolledWindow *scrolled_window,
                                             GtkShadowType type);
void        gtk_scrolled_window_set_hadjustment
                                            (GtkScrolledWindow *scrolled_window,
                                             GtkAdjustment *hadjustment);
void        gtk_scrolled_window_set_vadjustment
                                            (GtkScrolledWindow *scrolled_window,
                                             GtkAdjustment *hadjustment);
GtkCornerType gtk_scrolled_window_get_placement
                                            (GtkScrolledWindow *scrolled_window);
void        gtk_scrolled_window_get_policy  (GtkScrolledWindow *scrolled_window,
                                             GtkPolicyType *hscrollbar_policy,
                                             GtkPolicyType *vscrollbar_policy);
```

```
GtkShadowType gtk_scrolled_window_get_shadow_type
                                              (GtkScrolledWindow *scrolled_window);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkWidget
               +----GtkContainer
                     +----GtkBin
                           +----GtkScrolledWindow
```

Implemented Interfaces

GtkScrolledWindow implements AtkImplementorIface.

Properties

```
  "hadjustment"          GtkAdjustment         : Read / Write / Construct
  "hscrollbar-policy"    GtkPolicyType         : Read / Write
  "shadow-type"          GtkShadowType         : Read / Write
  "vadjustment"          GtkAdjustment         : Read / Write / Construct
  "vscrollbar-policy"    GtkPolicyType         : Read / Write
  "window-placement"     GtkCornerType         : Read / Write
```

Style Properties

```
  "scrollbar-spacing"    gint                  : Read
```

Signal Prototypes

```
"move-focus-out"
            void        user_function      (GtkScrolledWindow *scrolledwindow,
```

```
                                                GtkDirectionType arg1,
                                                gpointer user_data);
"scroll-child"
            void        user_function      (GtkScrolledWindow *scrolledwindow,
                                                GtkScrollType arg1,
                                                gboolean arg2,
                                                gpointer user_data);
```

Description

GtkScrolledWindow is a GtkBin subclass: it's a container the accepts a single child widget. GtkScrolledWindow adds scrollbars to the child widget and optionally draws a beveled frame around the child widget.

The scrolled window can work in two ways. Some widgets have native scrolling support; these widgets have "slots" for GtkAdjustment objects. [5] Widgets with native scroll support include GtkTreeView, GtkTextView, and GtkLayout.

For widgets that lack native scrolling support, the GtkViewport widget acts as an adaptor class, implementing scrollability for child widgets that lack their own scrolling capabilities. Use GtkViewport to scroll child widgets such as GtkTable, GtkBox, and so on.

If a widget has native scrolling abilities, it can be added to the GtkScrolledWindow with `gtk_container_add()`. If a widget does not, you must first add the widget to a GtkViewport, then add the GtkViewport to the scrolled window. The convenience function `gtk_scrolled_window_add_with_viewport()` does exactly this, so you can ignore the presence of the viewport.

The position of the scrollbars is controlled by the scroll adjustments. See GtkAdjustment for the fields in an adjustment - for GtkScrollbar, used by GtkScrolledWindow, the "value" field represents the position of the scrollbar, which must be between the "lower" field and "upper - page_size." The "page_size" field represents the size of the visible scrollable area. The "step_increment" and "page_increment" fields are used when the user asks to step down (using the small stepper arrows) or page down (using for example the PageDown key).

If a GtkScrolledWindow doesn't behave quite as you would like, or doesn't have exactly the right layout, it's very possible to set up your own scrolling with GtkScrollbar and for example a GtkTable.

Details

GtkScrolledWindow

```
typedef struct _GtkScrolledWindow GtkScrolledWindow;
```

There are no public fields in the GtkScrolledWindow struct; it should only be accessed using the functions below.

---

gtk_scrolled_window_new ()

```
GtkWidget*  gtk_scrolled_window_new         (GtkAdjustment *hadjustment,
                                             GtkAdjustment *vadjustment);
```

Creates a new scrolled window. The two arguments are the scrolled window's adjustments; these will be shared with the scrollbars and the child widget to keep the bars in sync with the child. Usually you want to pass `NULL` for the adjustments, which will cause the scrolled window to create them for you.

`hadjustment` : Horizontal adjustment.
`vadjustment` : Vertical adjustment.
Returns : New scrolled window.

---

gtk_scrolled_window_get_hadjustment ()

```
GtkAdjustment* gtk_scrolled_window_get_hadjustment
                                            (GtkScrolledWindow *scrolled_window);
```

Returns the horizontal scrollbar's adjustment, used to connect the horizontal scrollbar to the child widget's horizontal scroll functionality.

`scrolled_window` : A GtkScrolledWindow.
Returns : The horizontal GtkAdjustment.

---

gtk_scrolled_window_get_vadjustment ()

```
GtkAdjustment* gtk_scrolled_window_get_vadjustment
                                            (GtkScrolledWindow *scrolled_window);
```

Returns the vertical scrollbar's adjustment, used to connect the vertical scrollbar to the child widget's vertical scroll functionality.

`scrolled_window` : A GtkScrolledWindow.
Returns : The vertical GtkAdjustment.

---

gtk_scrolled_window_set_policy ()

```
void        gtk_scrolled_window_set_policy  (GtkScrolledWindow *scrolled_window,
                                             GtkPolicyType hscrollbar_policy,
                                             GtkPolicyType vscrollbar_policy);
```

Sets the scrollbar policy for the horizontal and vertical scrollbars. The policy determines when the scrollbar should appear; it is a value from the GtkPolicyType enumeration. If GTK_POLICY_ALWAYS, the scrollbar is always present; if GTK_POLICY_NEVER, the scrollbar is never present; if GTK_POLICY_AUTOMATIC, the scrollbar is present only if needed (that is, if the slider part of the bar would be smaller than the trough - the display is larger than the page size).

scrolled_window : A GtkScrolledWindow.
hscrollbar_policy : Policy for horizontal bar.
vscrollbar_policy : Policy for vertical bar.

---

gtk_scrolled_window_add_with_viewport ()

```
void        gtk_scrolled_window_add_with_viewport
                                            (GtkScrolledWindow *scrolled_window,
                                             GtkWidget *child);
```

Used to add children without native scrolling capabilities. This is simply a convenience function; it is equivalent to adding the unscrollable child to a viewport, then adding the viewport to the scrolled window. If a child has native scrolling, use gtk_container_add() instead of this function.

The viewport scrolls the child by moving its GdkWindow, and takes the size of the child to be the size of its toplevel GdkWindow. This will be very wrong for most widgets that support native scrolling; for example, if you add a widget such as GtkTreeView with a viewport, the whole widget will scroll, including the column headings. Thus, widgets with native scrolling support should not be used with the GtkViewport proxy.

A widget supports scrolling natively if the set_scroll_adjustments_signal field in GtkWidgetClass is non-zero, i.e. has been filled in with a valid signal identifier.

scrolled_window : A GtkScrolledWindow.
child : Widget you want to scroll.

---

gtk_scrolled_window_set_placement ()

```
void        gtk_scrolled_window_set_placement
```

```
                                                (GtkScrolledWindow *scrolled_window,
                                                 GtkCornerType window_placement);
```

Determines the location of the child widget with respect to the scrollbars. The default is GTK_CORNER_TOP_LEFT, meaning the child is in the top left, with the scrollbars underneath and to the right. Other values in GtkCornerType are GTK_CORNER_TOP_RIGHT, GTK_CORNER_BOTTOM_LEFT, and GTK_CORNER_BOTTOM_RIGHT.

scrolled_window : A GtkScrolledWindow.
window_placement : Position of the child window.

---

gtk_scrolled_window_set_shadow_type ()

```
void        gtk_scrolled_window_set_shadow_type
                                                (GtkScrolledWindow *scrolled_window,
                                                 GtkShadowType type);
```

Changes the type of shadow drawn around the contents of *scrolled_window*.

scrolled_window : a GtkScrolledWindow
type : kind of shadow to draw around scrolled window contents

---

gtk_scrolled_window_set_hadjustment ()

```
void        gtk_scrolled_window_set_hadjustment
                                                (GtkScrolledWindow *scrolled_window,
                                                 GtkAdjustment *hadjustment);
```

Sets the GtkAdjustment for the horizontal scrollbar.

scrolled_window : A GtkScrolledWindow.
hadjustment : Horizontal scroll adjustment.

---

gtk_scrolled_window_set_vadjustment ()

```
void        gtk_scrolled_window_set_vadjustment
                                                (GtkScrolledWindow *scrolled_window,
```

```
                                                GtkAdjustment *hadjustment);
```

Sets the GtkAdjustment for the vertical scrollbar.

scrolled_window : A GtkScrolledWindow.
hadjustment :

---

gtk_scrolled_window_get_placement ()

```
GtkCornerType gtk_scrolled_window_get_placement
                                                (GtkScrolledWindow *scrolled_window);
```

Gets the placement of the scrollbars for the scrolled window. See `gtk_scrolled_window_set_placement()`.

scrolled_window : a GtkScrolledWindow
Returns : the current placement value.

---

gtk_scrolled_window_get_policy ()

```
void        gtk_scrolled_window_get_policy  (GtkScrolledWindow *scrolled_window,
                                             GtkPolicyType *hscrollbar_policy,
                                             GtkPolicyType *vscrollbar_policy);
```

Retrieves the current policy values for the horizontal and vertical scrollbars. See `gtk_scrolled_window_set_policy()`.

scrolled_window : a GtkScrolledWindow
hscrollbar_policy : location to store the policy for the horizontal scrollbar, or `NULL`.
vscrollbar_policy : location to store the policy for the horizontal scrollbar, or `NULL`.

---

gtk_scrolled_window_get_shadow_type ()

```
GtkShadowType gtk_scrolled_window_get_shadow_type
                                                (GtkScrolledWindow *scrolled_window);
```

Gets the shadow type of the scrolled window. See `gtk_scrolled_window_set_shadow_type()`.

`scrolled_window` : a GtkScrolledWindow

Returns : the current shadow type

Properties

The "hadjustment" property

```
"hadjustment"          GtkAdjustment         : Read / Write / Construct
```

The GtkAdjustment for the horizontal position.

The "hscrollbar-policy" property

```
"hscrollbar-policy"    GtkPolicyType         : Read / Write
```

When the horizontal scrollbar is displayed.

Default value: GTK_POLICY_ALWAYS

The "shadow-type" property

```
"shadow-type"          GtkShadowType         : Read / Write
```

Style of bevel around the contents.

Default value: GTK_SHADOW_NONE

The "vadjustment" property

```
"vadjustment"          GtkAdjustment         : Read / Write / Construct
```

The GtkAdjustment for the vertical position.

The "vscrollbar-policy" property

```
  "vscrollbar-policy"    GtkPolicyType         : Read / Write
```

When the vertical scrollbar is displayed.

Default value: GTK_POLICY_ALWAYS

The "window-placement" property

```
  "window-placement"     GtkCornerType         : Read / Write
```

Where the contents are located with respect to the scrollbars.

Default value: GTK_CORNER_TOP_LEFT

Style Properties

The "scrollbar-spacing" style property

```
  "scrollbar-spacing"    gint                  : Read
```

Number of pixels between the scrollbars and the scrolled window.

Allowed values: >= 0

Default value: 3

Signals

The "move-focus-out" signal

```
void        user_function                  (GtkScrolledWindow *scrolledwindow,
                                            GtkDirectionType arg1,
```

```
                                        gpointer user_data);
```

scrolledwindow : the object which received the signal.
arg1 :
user_data : user data set when the signal handler was connected.

The "scroll-child" signal

```
void        user_function                   (GtkScrolledWindow *scrolledwindow,
                                             GtkScrollType arg1,
                                             gboolean arg2,
                                             gpointer user_data);
```

scrolledwindow : the object which received the signal.
arg1 :
arg2 :
user_data : user data set when the signal handler was connected.

See Also

GtkViewport, GtkAdjustment, GtkWidgetClass

---

[5] The scrolled window installs GtkAdjustment objects in the child window's slots using the set_scroll_adjustments_signal, found in GtkWidgetClass. (Conceptually, these widgets implement a "Scrollable" interface; because GTK+ 1.2 lacked interface support in the object system, this interface is hackily implemented as a signal in GtkWidgetClass. The GTK+ 2.0 object system would allow a clean implementation, but it wasn't worth breaking the API.)

<< GtkVScrollbar **Miscellaneous >>**

Miscellaneous

GtkAdjustment - A GtkObject representing an adjustable bounded value
GtkArrow - Displays an arrow
GtkCalendar - Displays a calendar and allows the user to select a date
GtkDrawingArea - A widget for custom user interface elements
GtkEventBox - A widget used to catch events for widgets which do not have their own window
GtkHandleBox - a widget for detachable window portions
GtkIMContextSimple - An input method context supporting table-based input methods
GtkIMMulticontext - An input method context supporting multiple, loadable input methods
GtkSizeGroup - Grouping widgets so they request the same size
GtkTooltips - Add tips to your widgets
GtkViewport - An adapter which makes widgets scrollable
GtkAccessible - Accessibility support for widgets

GtkAdjustment

GtkAdjustment — A GtkObject representing an adjustable bounded value

Synopsis

```
#include <gtk/gtk.h>

            GtkAdjustment;
GtkObject*  gtk_adjustment_new              (gdouble value,
                                             gdouble lower,
                                             gdouble upper,
                                             gdouble step_increment,
                                             gdouble page_increment,
                                             gdouble page_size);
gdouble     gtk_adjustment_get_value        (GtkAdjustment *adjustment);
void        gtk_adjustment_set_value        (GtkAdjustment *adjustment,
                                             gdouble value);
void        gtk_adjustment_clamp_page       (GtkAdjustment *adjustment,
                                             gdouble lower,
                                             gdouble upper);
void        gtk_adjustment_changed          (GtkAdjustment *adjustment);
void        gtk_adjustment_value_changed    (GtkAdjustment *adjustment);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkAdjustment
```

Properties

```
  "lower"                gdouble               : Read / Write
  "page-increment"       gdouble               : Read / Write
  "page-size"            gdouble               : Read / Write
  "step-increment"       gdouble               : Read / Write
  "upper"                gdouble               : Read / Write
  "value"                gdouble               : Read / Write
```

Signal Prototypes

```
"changed"   void        user_function      (GtkAdjustment *adjustment,
                                            gpointer user_data);
"value-changed"
            void        user_function      (GtkAdjustment *adjustment,
                                            gpointer user_data);
```

Description

The GtkAdjustment object represents a value which has an associated lower and upper bound, together with step and page increments, and a page size. It is used within several GTK+ widgets, including GtkSpinButton, GtkViewport, and GtkRange (which is a base class for GtkHScrollbar, GtkVScrollbar, GtkHScale, and GtkVScale).

The GtkAdjustment object does not update the value itself. Instead it is left up to the owner of the GtkAdjustment to control the value.

The owner of the GtkAdjustment typically calls the `gtk_adjustment_value_changed()` and `gtk_adjustment_changed()` functions after changing the value and its bounds. This results in the emission of the "value_changed" or "changed" signal respectively.

Details

GtkAdjustment

```
typedef struct _GtkAdjustment GtkAdjustment;
```

The GtkAdjustment-struct struct contains the following fields.

gdouble lower;	the minimum value.
gdouble upper;	the maximum value.
gdouble value;	the current value.
gdouble step_increment;	the increment to use to make minor changes to the value. In a GtkScrollbar this increment is used when the mouse is clicked on the arrows at the top and bottom of the scrollbar, to scroll by a small amount.
gdouble page_increment;	the increment to use to make major changes to the value. In a GtkScrollbar this increment is used when the mouse is clicked in the trough, to scroll by a large amount.
gdouble page_size;	the page size. In a GtkScrollbar this is the size of the area which is currently visible.

---

gtk_adjustment_new ()

```
GtkObject*  gtk_adjustment_new              (gdouble value,
                                             gdouble lower,
                                             gdouble upper,
                                             gdouble step_increment,
                                             gdouble page_increment,
                                             gdouble page_size);
```

Creates a new GtkAdjustment.

value :	the initial value.
lower :	the minimum value.
upper :	the maximum value.
step_increment :	the step increment.
page_increment :	the page increment.

page_size : the page size.

Returns : a new GtkAdjustment.

---

gtk_adjustment_get_value ()

```
gdouble     gtk_adjustment_get_value        (GtkAdjustment *adjustment);
```

Gets the current value of the adjustment. See `gtk_adjustment_set_value()`.

adjustment : a GtkAdjustment

Returns : The current value of the adjustment.

---

gtk_adjustment_set_value ()

```
void        gtk_adjustment_set_value        (GtkAdjustment *adjustment,
                                             gdouble value);
```

Sets the GtkAdjustment value. The value is clamped to lie between `adjustment->lower` and `adjustment->upper`.

Note that for adjustments which are used in a GtkScrollbar, the effective range of allowed values goes from `adjustment->lower` to `adjustment->upper - adjustment->page_size`.

adjustment : a GtkAdjustment.

value : the new value.

---

gtk_adjustment_clamp_page ()

```
void        gtk_adjustment_clamp_page       (GtkAdjustment *adjustment,
                                             gdouble lower,
                                             gdouble upper);
```

Updates the GtkAdjustment *value* to ensure that the range between *lower* and *upper* is in the current page (i.e. between *value* and *value* + *page_size*). If the range is larger than the page size, then only the start of it will be in the current page. A "changed" signal will be emitted if the value is changed.

adjustment : a GtkAdjustment.
lower : the lower value.
upper : the upper value.

---

gtk_adjustment_changed ()

```
void        gtk_adjustment_changed          (GtkAdjustment *adjustment);
```

Emits a "changed" signal from the GtkAdjustment. This is typically called by the owner of the GtkAdjustment after it has changed any of the GtkAdjustment fields other than the value.

adjustment :

---

gtk_adjustment_value_changed ()

```
void        gtk_adjustment_value_changed    (GtkAdjustment *adjustment);
```

Emits a "value_changed" signal from the GtkAdjustment. This is typically called by the owner of the GtkAdjustment after it has changed the GtkAdjustment value field.

adjustment :

Properties

The "lower" property

```
  "lower"                gdouble               : Read / Write
```

The minimum value of the adjustment.

Default value: 0

The "page-increment" property

```
"page-increment"       gdouble               : Read / Write
```

The page increment of the adjustment.

Default value: 0

The "page-size" property

```
"page-size"            gdouble               : Read / Write
```

The page size of the adjustment.

Default value: 0

The "step-increment" property

```
"step-increment"       gdouble               : Read / Write
```

The step increment of the adjustment.

Default value: 0

The "upper" property

```
  "upper"                gdouble               : Read / Write
```

The maximum value of the adjustment.

Default value: 0

The "value" property

```
  "value"                gdouble               : Read / Write
```

The value of the adjustment.

Default value: 0

Signals

The "changed" signal

```
void        user_function                      (GtkAdjustment *adjustment,
                                                gpointer user_data);
```

Emitted when one or more of the GtkAdjustment fields have been changed, other than the value field.

adjustment : the object which received the signal.
user_data : user data set when the signal handler was connected.

The "value-changed" signal

```
void        user_function                      (GtkAdjustment *adjustment,
```

```
                                                gpointer user_data);
```

Emitted when the GtkAdjustment value field has been changed.

adjustment : the object which received the signal.
user_data : user data set when the signal handler was connected.

<< Miscellaneous **GtkArrow >>**

GtkArrow

GtkArrow — Displays an arrow

Synopsis

```
#include <gtk/gtk.h>


            GtkArrow;
GtkWidget*  gtk_arrow_new                   (GtkArrowType arrow_type,
                                             GtkShadowType shadow_type);
void        gtk_arrow_set                   (GtkArrow *arrow,
                                             GtkArrowType arrow_type,
                                             GtkShadowType shadow_type);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkWidget
               +----GtkMisc
                     +----GtkArrow
```

Implemented Interfaces

GtkArrow implements AtkImplementorIface.

Properties

```
  "arrow-type"           GtkArrowType          : Read / Write
  "shadow-type"          GtkShadowType         : Read / Write
```

Description

GtkArrow should be used to draw simple arrows that need to point in one of the four cardinal directions (up, down, left, or right). The style of the arrow can be one of shadow in, shadow out, etched in, or etched out. Note that these directions and style types may be ammended in versions of Gtk to come.

GtkArrow will fill any space alloted to it, but since it is inherited from GtkMisc, it can be padded and/or aligned, to fill exactly the space the programmer desires.

Arrows are created with a call to `gtk_arrow_new()`. The direction or style of an arrow can be changed after creation by using `gtk_arrow_set()`.

Details

GtkArrow

```
typedef struct _GtkArrow GtkArrow;
```

The GtkArrow-struct containes the following fields. (These fields should be considered read-only. They should never be set by an application.)

gint16 *arrow_type*;	the direction of the arrow, one of GtkArrowType.
gint16 *shadow_type*;	the style of the arrow, one of GtkShadowType.

---

gtk_arrow_new ()

```
GtkWidget*  gtk_arrow_new                       (GtkArrowType arrow_type,
                                                 GtkShadowType shadow_type);
```

Creates a new arrow widget.

arrow_type :	a valid GtkArrowType.
shadow_type :	a valid GtkShadowType.
Returns :	the new GtkArrow widget.

---

gtk_arrow_set ()

```
void        gtk_arrow_set                       (GtkArrow *arrow,
                                                 GtkArrowType arrow_type,
                                                 GtkShadowType shadow_type);
```

Sets the direction and style of the GtkArrow, *arrow*.

arrow :	a widget of type GtkArrow.
arrow_type :	a valid GtkArrowType.
shadow_type :	a valid GtkShadowType.

Properties

The "arrow-type" property

```
  "arrow-type"           GtkArrowType          : Read / Write
```

The direction the arrow should point.

Default value: GTK_ARROW_RIGHT

The "shadow-type" property

```
  "shadow-type"          GtkShadowType         : Read / Write
```

Appearance of the shadow surrounding the arrow.

Default value: GTK_SHADOW_OUT

See Also

`gtk_paint_arrow()` the function used internally to paint the arrow.

<< GtkAdjustment **GtkCalendar >>**

GtkCalendar

GtkCalendar — Displays a calendar and allows the user to select a date

Synopsis

```
#include <gtk/gtk.h>


            GtkCalendar;
enum        GtkCalendarDisplayOptions;
GtkWidget*  gtk_calendar_new                (void);
gboolean    gtk_calendar_select_month       (GtkCalendar *calendar,
                                             guint month,
                                             guint year);
void        gtk_calendar_select_day         (GtkCalendar *calendar,
                                             guint day);
gboolean    gtk_calendar_mark_day           (GtkCalendar *calendar,
                                             guint day);
gboolean    gtk_calendar_unmark_day         (GtkCalendar *calendar,
                                             guint day);
void        gtk_calendar_clear_marks        (GtkCalendar *calendar);
GtkCalendarDisplayOptions gtk_calendar_get_display_options
                                            (GtkCalendar *calendar);
void        gtk_calendar_set_display_options
                                            (GtkCalendar *calendar,
                                             GtkCalendarDisplayOptions flags);
void        gtk_calendar_display_options    (GtkCalendar *calendar,
                                             GtkCalendarDisplayOptions flags);
void        gtk_calendar_get_date           (GtkCalendar *calendar,
                                             guint *year,
                                             guint *month,
                                             guint *day);
void        gtk_calendar_freeze             (GtkCalendar *calendar);
void        gtk_calendar_thaw               (GtkCalendar *calendar);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkWidget
               +----GtkCalendar
```

Implemented Interfaces

GtkCalendar implements AtkImplementorIface.

Properties

```
  "day"                 gint                  : Read / Write
  "month"               gint                  : Read / Write
  "no-month-change"     gboolean              : Read / Write
  "show-day-names"      gboolean              : Read / Write
  "show-heading"        gboolean              : Read / Write
  "show-week-numbers"   gboolean              : Read / Write
  "year"                gint                  : Read / Write
```

Signal Prototypes

```
"day-selected"
            void        user_function      (GtkCalendar *calendar,
                                            gpointer user_data);
"day-selected-double-click"
            void        user_function      (GtkCalendar *calendar,
                                            gpointer user_data);
"month-changed"
            void        user_function      (GtkCalendar *calendar,
                                            gpointer user_data);
"next-month"
            void        user_function      (GtkCalendar *calendar,
                                            gpointer user_data);
"next-year" void        user_function      (GtkCalendar *calendar,
```

```
                                                gpointer user_data);
"prev-month"
            void        user_function      (GtkCalendar *calendar,
                                                gpointer user_data);
"prev-year" void        user_function      (GtkCalendar *calendar,
                                                gpointer user_data);
```

Description

GtkCalendar is a widget that displays a calendar, one month at a time. It can be created with `gtk_calendar_new()`.

The month and year currently displayed can be altered with `gtk_calendar_select_month()`. The exact day can be selected from the displayed month using `gtk_calendar_select_day()`.

To place a visual marker on a particular day, use `gtk_calendar_mark_day()` and to remove the marker, `gtk_calendar_unmark_day()`. Alternative, all marks can be cleared with `gtk_calendar_clear_marks()`.

The way in which the calendar itself is displayed can be altered using `gtk_calendar_set_display_options()`.

The selected date can be retrieved from a GtkCalendar using `gtk_calendar_get_date()`.

If performing many 'mark' operations, the calendar can be frozen to prevent flicker, using `gtk_calendar_freeze()`, and 'thawed' again using `gtk_calendar_thaw()`.

Details

GtkCalendar

```
typedef struct _GtkCalendar GtkCalendar;
```

`num_marked_dates` is an integer containing the number of days that have a mark over them.

`marked_date` is an array containing the day numbers that currently have a mark over them.

`month`, *`year`*, and *`selected_day`* contain the currently visible month, year, and selected day respectively.

All of these fields should be considered read only, and everything in this struct should only be modified using the functions provided below.

Note

Note that *month* is zero-based (i.e it allowed values are 0-11) while *selected_day* is one-based (i.e. allowed values are 1-31).

---

enum GtkCalendarDisplayOptions

```
typedef enum
{
  GTK_CALENDAR_SHOW_HEADING             = 1 << 0,
  GTK_CALENDAR_SHOW_DAY_NAMES           = 1 << 1,
  GTK_CALENDAR_NO_MONTH_CHANGE          = 1 << 2,
  GTK_CALENDAR_SHOW_WEEK_NUMBERS        = 1 << 3,
  GTK_CALENDAR_WEEK_START_MONDAY        = 1 << 4
} GtkCalendarDisplayOptions;
```

These options can be used to influence the display and behaviour of a GtkCalendar.

GTK_CALENDAR_SHOW_HEADING	Specifies that the month and year should be displayed.
GTK_CALENDAR_SHOW_DAY_NAMES	Specifies that three letter day descriptions should be present.
GTK_CALENDAR_NO_MONTH_CHANGE	Prevents the user from switching months with the calendar.
GTK_CALENDAR_SHOW_WEEK_NUMBERS	Displays each week numbers of the current year, down the left side of the calendar.
GTK_CALENDAR_WEEK_START_MONDAY	Since GTK+ 2.4, this option is deprecated and ignored by GTK+. The information on which day the calendar week starts is derived from the locale.

---

gtk_calendar_new ()

```
GtkWidget*  gtk_calendar_new                (void);
```

Creates a new calendar, with the current date being selected.

Returns : a GtkCalendar.

---

gtk_calendar_select_month ()

```
gboolean    gtk_calendar_select_month       (GtkCalendar *calendar,
                                             guint month,
                                             guint year);
```

Shifts the calendar to a different month.

calendar : a GtkCalendar.
month : a month number between 0 and 11.
year : the year the month is in.
Returns : TRUE.

---

gtk_calendar_select_day ()

```
void        gtk_calendar_select_day         (GtkCalendar *calendar,
                                             guint day);
```

Selects a day from the current month.

calendar : a GtkCalendar.
day : the day number between 1 and 31, or 0 to unselect the currently selected day.

---

gtk_calendar_mark_day ()

```
gboolean    gtk_calendar_mark_day           (GtkCalendar *calendar,
                                             guint day);
```

Places a visual marker on a particular day.

calendar : a GtkCalendar.
day : the day number to mark between 1 and 31.
Returns : TRUE.

---

gtk_calendar_unmark_day ()

```
gboolean    gtk_calendar_unmark_day         (GtkCalendar *calendar,
                                             guint day);
```

Removes the visual marker from a particular day.

calendar : a GtkCalendar.
day : the day number to unmark between 1 and 31.
Returns : TRUE.

---

gtk_calendar_clear_marks ()

```
void        gtk_calendar_clear_marks        (GtkCalendar *calendar);
```

Remove all visual markers.

calendar : a GtkCalendar.

---

gtk_calendar_get_display_options ()

```
GtkCalendarDisplayOptions gtk_calendar_get_display_options
                                            (GtkCalendar *calendar);
```

Returns the current display options of *calendar*.

calendar : a GtkCalendar
Returns : the display options.

Since 2.4

---

gtk_calendar_set_display_options ()

```
void        gtk_calendar_set_display_options
                                            (GtkCalendar *calendar,
                                             GtkCalendarDisplayOptions flags);
```

Sets display options (whether to display the heading and the month headings).

calendar : a GtkCalendar
flags : the display options to set

Since 2.4

---

gtk_calendar_display_options ()

```
void        gtk_calendar_display_options    (GtkCalendar *calendar,
                                             GtkCalendarDisplayOptions flags);
```

Warning

`gtk_calendar_display_options` is deprecated and should not be used in newly-written code. Use `gtk_calendar_set_display_options()` instead

Sets display options (whether to display the heading and the month headings).

calendar : a GtkCalendar.
flags : the display options to set.

---

gtk_calendar_get_date ()

```
void        gtk_calendar_get_date           (GtkCalendar *calendar,
                                             guint *year,
                                             guint *month,
                                             guint *day);
```

Obtains the selected date from a GtkCalendar.

calendar : a GtkCalendar.
year : location to store the year number.
month : location to store the month number.
day : location to store the day number.

---

gtk_calendar_freeze ()

```
void        gtk_calendar_freeze             (GtkCalendar *calendar);
```

Locks the display of the calendar until it is thawed with `gtk_calendar_thaw()`.

calendar : a GtkCalendar.

---

gtk_calendar_thaw ()

```
void        gtk_calendar_thaw               (GtkCalendar *calendar);
```

Defrosts a calendar; all the changes made since the last `gtk_calendar_freeze()` are displayed.

calendar : a GtkCalendar.

Properties

The "day" property

```
  "day"                 gint                  : Read / Write
```

The selected day (as a number between 1 and 31, or 0 to unselect the currently selected day).

Allowed values: [0,31]

Default value: 0

The "month" property

```
"month"               gint                : Read / Write
```

The selected month (as a number between 0 and 11).

Allowed values: [0,11]

Default value: 0

The "no-month-change" property

```
"no-month-change"     gboolean            : Read / Write
```

Determines whether the selected month can be changed.

Default value: FALSE

Since 2.4

The "show-day-names" property

```
"show-day-names"      gboolean            : Read / Write
```

Determines whether day names are displayed.

Default value: TRUE

Since 2.4

The "show-heading" property

```
"show-heading"        gboolean            : Read / Write
```

Determines whether a heading is displayed.

Default value: TRUE

Since 2.4

The "show-week-numbers" property

```
  "show-week-numbers"    gboolean              : Read / Write
```

Determines whether week numbers are displayed.

Default value: FALSE

Since 2.4

The "year" property

```
  "year"                 gint                  : Read / Write
```

The selected year.

Allowed values: >= 0

Default value: 0

Signals

The "day-selected" signal

```
void        user_function                     (GtkCalendar *calendar,
                                               gpointer user_data);
```

Emitted when the user selects a day.

calendar : the object which received the signal.
user_data : user data set when the signal handler was connected.

The "day-selected-double-click" signal

```
void        user_function                   (GtkCalendar *calendar,
                                             gpointer user_data);
```

calendar : the object which received the signal.
user_data : user data set when the signal handler was connected.

The "month-changed" signal

```
void        user_function                   (GtkCalendar *calendar,
                                             gpointer user_data);
```

Emitted when the user clicks a button to change the selected month on a calendar.

calendar : the object which received the signal.
user_data : user data set when the signal handler was connected.

The "next-month" signal

```
void        user_function                   (GtkCalendar *calendar,
                                             gpointer user_data);
```

calendar : the object which received the signal.
user_data : user data set when the signal handler was connected.

The "next-year" signal

```
void        user_function                   (GtkCalendar *calendar,
```

```
                                        gpointer user_data);
```

calendar : the object which received the signal.
user_data : user data set when the signal handler was connected.

The "prev-month" signal

```
void        user_function                   (GtkCalendar *calendar,
                                             gpointer user_data);
```

calendar : the object which received the signal.
user_data : user data set when the signal handler was connected.

The "prev-year" signal

```
void        user_function                   (GtkCalendar *calendar,
                                             gpointer user_data);
```

calendar : the object which received the signal.
user_data : user data set when the signal handler was connected.

<< GtkArrow **GtkDrawingArea >>**

GtkDrawingArea

GtkDrawingArea — A widget for custom user interface elements

Synopsis

```
#include <gtk/gtk.h>


            GtkDrawingArea;
GtkWidget*  gtk_drawing_area_new            (void);
void        gtk_drawing_area_size           (GtkDrawingArea *darea,
                                             gint width,
                                             gint height);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkWidget
               +----GtkDrawingArea
                     +----GtkCurve
```

Implemented Interfaces

GtkDrawingArea implements AtkImplementorIface.

Description

The GtkDrawingArea widget is used for creating custom user interface elements. It's essentially a blank widget; you can draw on `widget->window`. After creating a drawing area, the application may want to connect to:

- Mouse and button press signals to respond to input from the user. (Use `gtk_widget_add_events()` to enable events you wish to receive.)
- The "realize" signal to take any necessary actions when the widget is instantiated on a particular display. (Create GDK resources in response to this signal.)
- The "configure_event" signal to take any necessary actions when the widget changes size.
- The "expose_event" signal to handle redrawing the contents of the widget.

The following code portion demonstrates using a drawing area to display a circle in the normal widget foreground color. Note that GDK automatically clears the exposed area to the background color before sending the expose event, and that drawing is implicitly clipped to the exposed area.

Example 1. Simple GtkDrawingArea usage.

```
gboolean
expose_event_callback (GtkWidget *widget, GdkEventExpose *event, gpointer data)
{
  gdk_draw_arc (widget->window,
                widget->style->fg_gc[GTK_WIDGET_STATE (widget)],
                TRUE,
                0, 0, widget->allocation.width, widget->allocation.height,
                0, 64 * 360);

  return TRUE;
}
[...]
  GtkWidget *drawing_area = gtk_drawing_area_new ();
  gtk_widget_set_size_request (drawing_area, 100, 100);
  g_signal_connect (G_OBJECT (drawing_area), "expose_event",
                    G_CALLBACK (expose_event_callback), NULL);
```

Expose events are normally delivered when a drawing area first comes onscreen, or when it's covered by another window and then uncovered (exposed). You can also force an expose event by adding to the "damage region" of the drawing area's window; `gtk_widget_queue_draw_area()` and `gdk_window_invalidate_rect()` are equally good ways to do this. You'll then get an expose event for the invalid region.

The available routines for drawing are documented on the GDK Drawing Primitives page. See also `gdk_pixbuf_render_to_drawable()` for drawing a GdkPixbuf.

To receive mouse events on a drawing area, you will need to enable them with `gtk_widget_add_events()`. To receive keyboard events, you will need to set the GTK_CAN_FOCUS flag on the drawing area, and should probably draw some user-visible indication that the drawing area is focused. Use the `GTK_HAS_FOCUS()` macro in your expose event handler to decide whether to draw the focus indicator. See `gtk_paint_focus()` for one way to draw focus.

Details

GtkDrawingArea

```
typedef struct _GtkDrawingArea GtkDrawingArea;
```

The GtkDrawingArea struct contains private data only, and should be accessed using the functions below.

---

gtk_drawing_area_new ()

```
GtkWidget*  gtk_drawing_area_new            (void);
```

Creates a new drawing area.

Returns : a new GtkDrawingArea

---

gtk_drawing_area_size ()

```
void        gtk_drawing_area_size           (GtkDrawingArea *darea,
                                             gint width,
                                             gint height);
```

Warning

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

(Use `gtk_widget_set_size_request()` instead.) Sets the size that the drawing area will request in response to a "size_request" signal. The drawing area may actually be allocated a size larger than this depending on how it is packed within the enclosing containers.

darea : a GtkDrawingArea.
width : the width to request.
height : the height to request.

See Also

Sometimes GtkImage is a useful alternative to a drawing area. You can put a GdkPixmap in the GtkImage and draw to

the GdkPixmap, calling `gtk_widget_queue_draw()` on the GtkImage when you want to refresh to the screen.

<< GtkCalendar **GtkEventBox >>**

GtkEventBox

GtkEventBox — A widget used to catch events for widgets which do not have their own window

Synopsis

```
#include <gtk/gtk.h>


            GtkEventBox;
GtkWidget*  gtk_event_box_new               (void);
void        gtk_event_box_set_above_child   (GtkEventBox *event_box,
                                             gboolean above_child);
gboolean    gtk_event_box_get_above_child   (GtkEventBox *event_box);
void        gtk_event_box_set_visible_window
                                            (GtkEventBox *event_box,
                                             gboolean visible_window);
gboolean    gtk_event_box_get_visible_window
                                            (GtkEventBox *event_box);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkWidget
               +----GtkContainer
                     +----GtkBin
                           +----GtkEventBox
```

Implemented Interfaces

GtkEventBox implements AtkImplementorIface.

Properties

```
  "above-child"          gboolean              : Read / Write
  "visible-window"       gboolean              : Read / Write
```

Description

The GtkEventBox widget is a subclass of GtkBin which also has its own window. It is useful since it allows you to catch events for widgets which do not have their own window.

Details

GtkEventBox

```
typedef struct _GtkEventBox GtkEventBox;
```

The GtkEventBox-struct struct contains private data only, and should be accessed using the functions below.

---

gtk_event_box_new ()

```
GtkWidget*  gtk_event_box_new                (void);
```

Creates a new GtkEventBox.

Returns : a new GtkEventBox.

gtk_event_box_set_above_child ()

```
void        gtk_event_box_set_above_child   (GtkEventBox *event_box,
                                             gboolean above_child);
```

Set whether the event box window is positioned above the windows of its child, as opposed to below it. If the window is above, all events inside the event box will go to the event box. If the window is below, events in windows of child widgets will first got to that widget, and then to its parents.

The default is to keep the window below the child.

event_box : a GtkEventBox

above_child : TRUE if the event box window is above the windows of its child

Since 2.4

gtk_event_box_get_above_child ()

```
gboolean    gtk_event_box_get_above_child   (GtkEventBox *event_box);
```

Returns whether the event box window is above or below the windows of its child. See gtk_event_box_set_above_child() for details.

event_box : a GtkEventBox

Returns : TRUE if the event box window is above the window of its child.

Since 2.4

gtk_event_box_set_visible_window ()

```
void        gtk_event_box_set_visible_window
                                            (GtkEventBox *event_box,
                                             gboolean visible_window);
```

Set whether the event box uses a visible or invisible child window. The default is to use visible windows.

In an invisible window event box, the window that that the event box creates is a GDK_INPUT_ONLY window, which means that it is invisible and only serves to receive events.

A visible window event box creates a visible (GDK_INPUT_OUTPUT) window that acts as the parent window for all the widgets contained in the event box.

You should generally make your event box invisible if you just want to trap events. Creating a visible window may cause artifacts that are visible to the user, especially if the user is using a theme with gradients or pixmaps.

The main reason to create a non input-only event box is if you want to set the background to a different color or draw on it.

Note

There is one unexpected issue for an invisible event box that has its window below the child. (See gtk_event_box_set_above_child().) Since the input-only window is not an ancestor window of any windows that descendent widgets of the event box create, events on these windows aren't propagated up by the windowing system, but only by GTK+. The practical effect of this is if an event isn't in the event mask for the descendant window (see gtk_widget_add_events()), it won't be received by the event box.

This problem doesn't occur for visible event boxes, because in that case, the event box window is actually the ancestor of the descendant windows, not just at the same place on the screen.

event_box : a GtkEventBox

visible_window : boolean value

Since 2.4

---

gtk_event_box_get_visible_window ()

```
gboolean    gtk_event_box_get_visible_window
                                            (GtkEventBox *event_box);
```

Returns whether the event box has a visible window. See `gtk_event_box_set_visible_window()` for details.

event_box : a GtkEventBox

Returns : TRUE if the event box window is visible

Since 2.4

Properties

The "above-child" property

```
  "above-child"          gboolean              : Read / Write
```

Whether the event-trapping window of the eventbox is above the window of the child widget as opposed to below it.

Default value: FALSE

The "visible-window" property

```
  "visible-window"       gboolean              : Read / Write
```

Whether the event box is visible, as opposed to invisible and only used to trap events.

Default value: TRUE

<< GtkDrawingArea **GtkHandleBox >>**

GtkHandleBox

GtkHandleBox — a widget for detachable window portions

Synopsis

```
#include <gtk/gtk.h>


            GtkHandleBox;
GtkWidget*  gtk_handle_box_new              (void);
void        gtk_handle_box_set_shadow_type  (GtkHandleBox *handle_box,
                                             GtkShadowType type);
void        gtk_handle_box_set_handle_position
                                            (GtkHandleBox *handle_box,
                                             GtkPositionType position);
void        gtk_handle_box_set_snap_edge    (GtkHandleBox *handle_box,
                                             GtkPositionType edge);
GtkPositionType gtk_handle_box_get_handle_position
                                            (GtkHandleBox *handle_box);
GtkShadowType gtk_handle_box_get_shadow_type
                                            (GtkHandleBox *handle_box);
GtkPositionType gtk_handle_box_get_snap_edge
                                            (GtkHandleBox *handle_box);
```

Object Hierarchy

```
  GObject
   +----GtkObject
```

```
          +----GtkWidget
                +----GtkContainer
                      +----GtkBin
                            +----GtkHandleBox
```

Implemented Interfaces

GtkHandleBox implements AtkImplementorIface.

Properties

```
  "handle-position"      GtkPositionType       : Read / Write
  "shadow"               GtkShadowType         : Read / Write
  "shadow-type"          GtkShadowType         : Read / Write
  "snap-edge"            GtkPositionType       : Read / Write
  "snap-edge-set"        gboolean              : Read / Write
```

Signal Prototypes

```
"child-attached"
            void        user_function      (GtkHandleBox *handlebox,
                                            GtkWidget *widget,
                                            gpointer user_data);
"child-detached"
            void        user_function      (GtkHandleBox *handlebox,
                                            GtkWidget *widget,
                                            gpointer user_data);
```

Description

The GtkHandleBox widget allows a portion of a window to be "torn off". It is a bin widget which displays its child and a handle that the user can drag to tear off a separate window (the *float window*) containing the child widget. A thin *ghost* is drawn in the original location of the handlebox. By dragging the separate

window back to its original location, it can be reattached.

When reattaching, the ghost and float window, must be aligned along one of the edges, the *snap edge*. This either can be specified by the application programmer explicitely, or GTK+ will pick a reasonable default based on the handle position.

To make detaching and reattaching the handlebox as minimally confusing as possible to the user, it is important to set the snap edge so that the snap edge does not move when the handlebox is deattached. For instance, if the handlebox is packed at the bottom of a VBox, then when the handlebox is detached, the bottom edge of the handlebox's allocation will remain fixed as the height of the handlebox shrinks, so the snap edge should be set to `GTK_POS_BOTTOM`.

Details

GtkHandleBox

```
typedef struct _GtkHandleBox GtkHandleBox;
```

The GtkHandleBox-struct struct contains the following fields. (These fields should be considered read-only. They should never be set by an application.)

GtkShadowType shadow_type;	The shadow type for the entry. (See `gtk_handle_box_set_shadow_type()`).
GtkPositionType handle_position;	The position of the handlebox's handle with respect to the child. (See `gtk_handle_box_set_handle_position()`)
gint snap_edge;	A value of type GtkPosition type indicating snap edge for the widget. (See gtk_handle_box_set_snap_edge). The value of -1 indicates that this value has not been set.
gboolean child_detached;	A boolean value indicating whether the handlebox's child is attached or detached.

---

gtk_handle_box_new ()

```
GtkWidget*  gtk_handle_box_new              (void);
```

Create a new handle box.

Returns : a new GtkHandleBox.

---

gtk_handle_box_set_shadow_type ()

```
void        gtk_handle_box_set_shadow_type  (GtkHandleBox *handle_box,
                                             GtkShadowType type);
```

Sets the type of shadow to be drawn around the border of the handle box.

handle_box : a GtkHandleBox
type : the shadow type.

---

gtk_handle_box_set_handle_position ()

```
void        gtk_handle_box_set_handle_position
                                            (GtkHandleBox *handle_box,
                                             GtkPositionType position);
```

Sets the side of the handlebox where the handle is drawn.

handle_box : a GtkHandleBox
position : the side of the handlebox where the handle should be drawn.

---

gtk_handle_box_set_snap_edge ()

```
void        gtk_handle_box_set_snap_edge    (GtkHandleBox *handle_box,
                                             GtkPositionType edge);
```

Sets the snap edge of a handlebox. The snap edge is the edge of the detached child that must be aligned with

the corresponding edge of the "ghost" left behind when the child was detached to reattach the torn-off window. Usually, the snap edge should be chosen so that it stays in the same place on the screen when the handlebox is torn off.

If the snap edge is not set, then an appropriate value will be guessed from the handle position. If the handle position is `GTK_POS_RIGHT` or `GTK_POS_LEFT`, then the snap edge will be `GTK_POS_TOP`, otherwise it will be `GTK_POS_LEFT`.

`handle_box` : a GtkHandleBox

`edge` : the snap edge, or -1 to unset the value; in which case GTK+ will try to guess an appropriate value in the future.

---

gtk_handle_box_get_handle_position ()

```
GtkPositionType gtk_handle_box_get_handle_position
                                                (GtkHandleBox *handle_box);
```

Gets the handle position of the handle box. See `gtk_handle_box_set_handle_position()`.

`handle_box` : a GtkHandleBox

Returns : the current handle position.

---

gtk_handle_box_get_shadow_type ()

```
GtkShadowType gtk_handle_box_get_shadow_type
                                                (GtkHandleBox *handle_box);
```

Gets the type of shadow drawn around the handle box. See `gtk_handle_box_set_shadow_type()`.

`handle_box` : a GtkHandleBox

Returns : the type of shadow currently drawn around the handle box.

---

gtk_handle_box_get_snap_edge ()

```
GtkPositionType gtk_handle_box_get_snap_edge
                                                (GtkHandleBox *handle_box);
```

Gets the edge used for determining reattachment of the handle box. See gtk_handle_box_set_snap_edge().

handle_box : a GtkHandleBox

Returns : the edge used for determining reattachment, or (GtkPositionType)-1 if this is determined (as per default) from the handle position.

Properties

The "handle-position" property

```
  "handle-position"      GtkPositionType       : Read / Write
```

Position of the handle relative to the child widget.

Default value: GTK_POS_LEFT

The "shadow" property

```
  "shadow"               GtkShadowType         : Read / Write
```

Deprecated property, use shadow_type instead.

Default value: GTK_SHADOW_ETCHED_OUT

The "shadow-type" property

```
  "shadow-type"          GtkShadowType         : Read / Write
```

Appearance of the shadow that surrounds the container.

Default value: GTK_SHADOW_ETCHED_OUT

The "snap-edge" property

```
  "snap-edge"            GtkPositionType       : Read / Write
```

Side of the handlebox that's lined up with the docking point to dock the handlebox.

Default value: GTK_POS_TOP

The "snap-edge-set" property

```
  "snap-edge-set"        gboolean              : Read / Write
```

Whether to use the value from the snap_edge property or a value derived from handle_position.

Default value: FALSE

Signals

The "child-attached" signal

```
void        user_function                     (GtkHandleBox *handlebox,
                                               GtkWidget *widget,
                                               gpointer user_data);
```

This signal is emitted when the contents of the handlebox are reattached to the main window.

handlebox : the object which received the signal.

widget : the child widget of the handlebox. (this argument provides no extra information and is here only for backwards-compatibility)

user_data : user data set when the signal handler was connected.

The "child-detached" signal

```
void        user_function                   (GtkHandleBox *handlebox,
                                             GtkWidget *widget,
                                             gpointer user_data);
```

This signal is emitted when the contents of the handlebox are detached from the main window.

handlebox : the object which received the signal.

widget : the child widget of the handlebox. (this argument provides no extra information and is here only for backwards-compatibility)

user_data : user data set when the signal handler was connected.

<< GtkEventBox **GtkIMContextSimple >>**

GtkIMContextSimple

GtkIMContextSimple — An input method context supporting table-based input methods

Synopsis

```
#include <gtk/gtk.h>


            GtkIMContextSimple;
GtkIMContext* gtk_im_context_simple_new     (void);
void        gtk_im_context_simple_add_table (GtkIMContextSimple *context_simple,
                                             guint16 *data,
                                             gint max_seq_len,
                                             gint n_seqs);
#define     GTK_MAX_COMPOSE_LEN
```

Object Hierarchy

```
  GObject
   +----GtkIMContext
         +----GtkIMContextSimple
```

Description

Details

GtkIMContextSimple

```
typedef struct _GtkIMContextSimple GtkIMContextSimple;
```

gtk_im_context_simple_new ()

```
GtkIMContext* gtk_im_context_simple_new     (void);
```

Creates a new GtkIMContextSimple.

Returns : a new GtkIMContextSimple.

---

gtk_im_context_simple_add_table ()

```
void        gtk_im_context_simple_add_table (GtkIMContextSimple *context_simple,
                                             guint16 *data,
                                             gint max_seq_len,
                                             gint n_seqs);
```

Adds an additional table to search to the input context. Each row of the table consists of *max_seq_len* key symbols followed by two guint16 interpreted as the high and low words of a gunicode value. Tables are searched starting from the last added.

The table must be sorted in dictionary order on the numeric value of the key symbol fields. (Values beyond the length of the sequence should be zero.)

context_simple :	A GtkIMContextSimple
data :	the table
max_seq_len :	Maximum length of a sequence in the table (cannot be greater than GTK_MAX_COMPOSE_LEN)
n_seqs :	number of sequences in the table

GTK_MAX_COMPOSE_LEN

```
#define GTK_MAX_COMPOSE_LEN 7
```

The maximum length of sequences in compose tables.

<< GtkHandleBox **GtkIMMulticontext >>**

GtkIMMulticontext

GtkIMMulticontext — An input method context supporting multiple, loadable input methods

Synopsis

```
#include <gtk/gtk.h>

            GtkIMMulticontext;
GtkIMContext* gtk_im_multicontext_new       (void);
void        gtk_im_multicontext_append_menuitems
                                            (GtkIMMulticontext *context,
                                             GtkMenuShell *menushell);
```

Object Hierarchy

```
  GObject
   +----GtkIMContext
         +----GtkIMMulticontext
```

Description

Details

GtkIMMulticontext

```
typedef struct _GtkIMMulticontext GtkIMMulticontext;
```

---

gtk_im_multicontext_new ()

```
GtkIMContext* gtk_im_multicontext_new       (void);
```

Creates a new GtkIMMulticontext.

Returns : a new GtkIMMulticontext.

---

gtk_im_multicontext_append_menuitems ()

```
void        gtk_im_multicontext_append_menuitems
                                            (GtkIMMulticontext *context,
                                             GtkMenuShell *menushell);
```

Add menuitems for various available input methods to a menu; the menuitems, when selected, will switch the input method for the context and the global default input method.

context : a GtkIMMultiContext
menushell : a GtkMenuShell

<< GtkIMContextSimple **GtkSizeGroup >>**

GtkSizeGroup

GtkSizeGroup — Grouping widgets so they request the same size

Synopsis

```
#include <gtk/gtk.h>


            GtkSizeGroup;
enum        GtkSizeGroupMode;
GtkSizeGroup* gtk_size_group_new            (GtkSizeGroupMode mode);
void        gtk_size_group_set_mode         (GtkSizeGroup *size_group,
                                             GtkSizeGroupMode mode);
GtkSizeGroupMode gtk_size_group_get_mode    (GtkSizeGroup *size_group);
void        gtk_size_group_add_widget       (GtkSizeGroup *size_group,
                                             GtkWidget *widget);
void        gtk_size_group_remove_widget    (GtkSizeGroup *size_group,
                                             GtkWidget *widget);
```

Object Hierarchy

```
  GObject
   +----GtkSizeGroup
```

Properties

```
  "mode"                GtkSizeGroupMode      : Read / Write
```

Description

GtkSizeGroup provides a mechanism for grouping a number of widgets together so they all request the same amount of space. This is typically useful when you want a column of widgets to have the same size, but you can't use a GtkTable widget.

In detail, the size requested for each widget in a GtkSizeGroup is the maximum of the sizes that would have been requested for each widget in the size group if they were not in the size group. The mode of the size group (see `gtk_size_group_set_mode()`) determines whether this applies to the horizontal size, the vertical size, or both sizes.

Note that size groups only affect the amount of space requested, not the size that the widgets finally receive. If you want the widgets in a GtkSizeGroup to actually be the same size, you need to pack them in such a way that they get the size they request and not more. For example, if you are packing your widgets into a table, you would not include the `GTK_FILL` flag.

GtkSizeGroup objects are referenced by each widget in the size group, so once you have added all widgets to a GtkSizeGroup, you can drop the initial reference to the size group with `g_object_unref()`. If the widgets in the size group are subsequently destroyed, then they will be removed from the size group and drop their references on the size group; when all widgets have been removed, the size group will be freed.

Widgets can be part of multiple size groups; GTK+ will compute the horizontal size of a widget from the horizontal requisition of all widgets that can be reached from the widget by a chain of size groups of type `GTK_SIZE_GROUP_HORIZONTAL` or `GTK_SIZE_GROUP_BOTH`, and the vertical size from the vertical requisition of all widgets that can be reached from the widget by a chain of size groups of type `GTK_SIZE_GROUP_VERTICAL` or `GTK_SIZE_GROUP_BOTH`.

Details

GtkSizeGroup

```
typedef struct _GtkSizeGroup GtkSizeGroup;
```

---

enum GtkSizeGroupMode

```
typedef enum {
  GTK_SIZE_GROUP_NONE,
  GTK_SIZE_GROUP_HORIZONTAL,
  GTK_SIZE_GROUP_VERTICAL,
  GTK_SIZE_GROUP_BOTH
} GtkSizeGroupMode;
```

The mode of the size group determines the directions in which the size group effects the requested sizes of its component widgets.

GTK_SIZE_GROUP_NONE	group has no effect
GTK_SIZE_GROUP_HORIZONTAL	group effects horizontal requisition
GTK_SIZE_GROUP_VERTICAL	group effects vertical requisition
GTK_SIZE_GROUP_BOTH	group effects both horizontal and vertical requisition

---

gtk_size_group_new ()

```
GtkSizeGroup* gtk_size_group_new           (GtkSizeGroupMode mode);
```

Create a new GtkSizeGroup.

mode : the mode for the new size group.

Returns : a newly created GtkSizeGroup

---

gtk_size_group_set_mode ()

```
void        gtk_size_group_set_mode        (GtkSizeGroup *size_group,
                                            GtkSizeGroupMode mode);
```

Sets the GtkSizeGroupMode of the size group. The mode of the size group determines whether the widgets in the size group should all have the same horizontal requisition (GTK_SIZE_GROUP_MODE_HORIZONTAL) all have the same vertical requisition

(GTK_SIZE_GROUP_MODE_VERTICAL), or should all have the same requisition in both directions (GTK_SIZE_GROUP_MODE_BOTH).

size_group : a GtkSizeGroup
mode : the mode to set for the size group.

---

gtk_size_group_get_mode ()

```
GtkSizeGroupMode gtk_size_group_get_mode    (GtkSizeGroup *size_group);
```

Gets the current mode of the size group. See gtk_size_group_set_mode().

size_group : a GtkSizeGroup
Returns : the current mode of the size group.

---

gtk_size_group_add_widget ()

```
void        gtk_size_group_add_widget       (GtkSizeGroup *size_group,
                                             GtkWidget *widget);
```

Adds a widget to a GtkSizeGroup. In the future, the requisition of the widget will be determined as the maximum of its requisition and the requisition of the other widgets in the size group. Whether this applies horizontally, vertically, or in both directions depends on the mode of the size group. See gtk_size_group_set_mode().

size_group : a GtkSizeGroup
widget : the GtkWidget to add

---

gtk_size_group_remove_widget ()

```
void        gtk_size_group_remove_widget    (GtkSizeGroup *size_group,
```

```
                                                GtkWidget *widget);
```

Removes a widget from a GtkSizeGroup.

size_group : a GtkSizeGrup
widget : the GtkWidget to remove

Properties

The "mode" property

```
  "mode"                 GtkSizeGroupMode      : Read / Write
```

The directions in which the size group effects the requested sizes of its component widgets.

Default value: GTK_SIZE_GROUP_HORIZONTAL

<< GtkIMMulticontext **GtkTooltips >>**

GtkTooltips

GtkTooltips — Add tips to your widgets

Synopsis

```
#include <gtk/gtk.h>

            GtkTooltips;
            GtkTooltipsData;
GtkTooltips* gtk_tooltips_new               (void);
void        gtk_tooltips_enable             (GtkTooltips *tooltips);
void        gtk_tooltips_disable            (GtkTooltips *tooltips);
void        gtk_tooltips_set_delay          (GtkTooltips *tooltips,
                                             guint delay);
void        gtk_tooltips_set_tip            (GtkTooltips *tooltips,
                                             GtkWidget *widget,
                                             const gchar *tip_text,
                                             const gchar *tip_private);
GtkTooltipsData* gtk_tooltips_data_get      (GtkWidget *widget);
void        gtk_tooltips_force_window       (GtkTooltips *tooltips);
gboolean    gtk_tooltips_get_info_from_tip_window
                                            (GtkWindow *tip_window,
                                             GtkTooltips **tooltips,
                                             GtkWidget **current_widget);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkTooltips
```

Description

Tooltips are the messages that appear next to a widget when the mouse pointer is held over it for a short amount of time. They are especially helpful for adding more verbose descriptions of things such as buttons in a toolbar.

An individual tooltip belongs to a group of tooltips. A group is created with a call to `gtk_tooltips_new()`. Every tooltip in the group can then be turned off with a call to `gtk_tooltips_disable()` and enabled with `gtk_tooltips_enable()`.

The length of time the user must keep the mouse over a widget before the tip is shown, can be altered with `gtk_tooltips_set_delay()`. This is set on a 'per group of tooltips' basis.

To assign a tip to a particular GtkWidget, `gtk_tooltips_set_tip()` is used.

Note

Tooltips can only be set on widgets which have their own X window. To check if a widget has its own window use `GTK_WIDGET_NO_WINDOW()`. To add a tooltip to a widget that doesn't have its own window, place the widget inside a GtkEventBox and add a tooltip to that instead.

The default appearance of all tooltips in a program is determined by the current GTK+ theme that the user has selected.

Information about the tooltip (if any) associated with an arbitrary widget can be retrieved using `gtk_tooltips_data_get()`.

Example 2. Adding tooltips to buttons.

```
GtkWidget *load_button, *save_button, *hbox;
GtkTooltips *button_bar_tips;

button_bar_tips = gtk_tooltips_new ();

/* Create the buttons and pack them into a GtkHBox */
hbox = gtk_hbox_new (TRUE, 2);

load_button = gtk_button_new_with_label ("Load a file");
gtk_box_pack_start (GTK_BOX (hbox), load_button, TRUE, TRUE, 2);
gtk_widget_show (load_button);

save_button = gtk_button_new_with_label ("Save a file");
gtk_box_pack_start (GTK_BOX (hbox), save_button, TRUE, TRUE, 2);
gtk_widget_show (save_button);
gtk_widget_show (hbox);
```

```
   /* Add the tips */
   gtk_tooltips_set_tip (GTK_TOOLTIPS (button_bar_tips), load_button,
                                 "Load a new document into this window",
                                 "Requests the filename of a document.
                                  This will then be loaded into the current
                                  window, replacing the contents of whatever
                                  is already loaded.");
   gtk_tooltips_set_tip (GTK_TOOLTIPS (button_bar_tips), save_button,
                                 "Saves the current document to a file",
                                 "If you have saved the document previously,
                                  then the new version will be saved over the
                                  old one. Otherwise, you will be prompted for
                                  a filename.");
```

Details

GtkTooltips

```
typedef struct _GtkTooltips GtkTooltips;
```

Holds information about a group of tooltips. Fields should be changed using the functions provided, rather than directly accessing the struct's members.

GtkTooltipsData

```
typedef struct {
  GtkTooltips *tooltips;
  GtkWidget *widget;
  gchar *tip_text;
  gchar *tip_private;
} GtkTooltipsData;
```

`tooltips` is the GtkTooltips group that this tooltip belongs to. *`widget`* is the GtkWidget that this tooltip data is associated with. *`tip_text`* is a string containing the tooltip message itself.

`tip_private` is a string that is not shown as the default tooltip. Instead, this message may be more informative and go towards forming a context-sensitive help system for your application. (FIXME: how to actually "switch on" private tips?)

---

gtk_tooltips_new ()

```
GtkTooltips* gtk_tooltips_new                (void);
```

Creates an empty group of tooltips. This function initialises a GtkTooltips structure. Without at least one such structure, you can not add tips to your application.

Returns : a new GtkTooltips group for you to use.

---

gtk_tooltips_enable ()

```
void        gtk_tooltips_enable              (GtkTooltips *tooltips);
```

Allows the user to see your tooltips as they navigate your application.

tooltips : a GtkTooltips.

---

gtk_tooltips_disable ()

```
void        gtk_tooltips_disable             (GtkTooltips *tooltips);
```

Causes all tooltips in *tooltips* to become inactive. Any widgets that have tips associated with that group will no longer display their tips until they are enabled again with gtk_tooltips_enable().

tooltips : a GtkTooltips.

---

gtk_tooltips_set_delay ()

```
void        gtk_tooltips_set_delay           (GtkTooltips *tooltips,
                                              guint delay);
```

Warning

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

Sets the time between the user moving the mouse over a widget and the widget's tooltip appearing.

`tooltips` : a GtkTooltips.
`delay` : an integer value representing seconds (FIXME: double-check this).

---

gtk_tooltips_set_tip ()

```
void        gtk_tooltips_set_tip            (GtkTooltips *tooltips,
                                             GtkWidget *widget,
                                             const gchar *tip_text,
                                             const gchar *tip_private);
```

Adds a tooltip containing the message *`tip_text`* to the specified GtkWidget.

`tooltips` : a GtkTooltips.
`widget` : the GtkWidget you wish to associate the tip with.
`tip_text` : a string containing the tip itself.
`tip_private` : a string of any further information that may be useful if the user gets stuck.

---

gtk_tooltips_data_get ()

```
GtkTooltipsData* gtk_tooltips_data_get      (GtkWidget *widget);
```

Retrieves any GtkTooltipsData previously associated with the given widget.

`widget` : a GtkWidget.
Returns : a GtkTooltipsData struct, or `NULL` if the widget has no tooltip.

---

gtk_tooltips_force_window ()

```
void        gtk_tooltips_force_window       (GtkTooltips *tooltips);
```

Ensures that the window used for displaying the given *tooltips* is created.

Applications should never have to call this function, since GTK+ takes care of this.

tooltips : a GtkToolTips

---

gtk_tooltips_get_info_from_tip_window ()

```
gboolean    gtk_tooltips_get_info_from_tip_window
                                            (GtkWindow *tip_window,
                                             GtkTooltips **tooltips,
                                             GtkWidget **current_widget);
```

Determines the tooltips and the widget they belong to from the window in which they are displayed.

This function is mostly intended for use by accessibility technologies; applications should have little use for it.

tip_window :	a GtkWindow
tooltips :	the return location for the tooltips which are displayed in *tip_window*, or NULL
current_widget :	the return location for the widget whose tooltips are displayed, or NULL
Returns :	TRUE if *tip_window* is displaying tooltips, otherwise FALSE.

Since 2.4

See Also

GtkToolbar	Create groups of widgets with their own tooltips.
GtkTipsQuery	Query tooltips to create context-sensitive help.

<< GtkSizeGroup **GtkViewport >>**

GtkViewport

GtkViewport — An adapter which makes widgets scrollable

Synopsis

```
#include <gtk/gtk.h>


            GtkViewport;
GtkWidget*  gtk_viewport_new                (GtkAdjustment *hadjustment,
                                             GtkAdjustment *vadjustment);
GtkAdjustment* gtk_viewport_get_hadjustment (GtkViewport *viewport);
GtkAdjustment* gtk_viewport_get_vadjustment (GtkViewport *viewport);
void        gtk_viewport_set_hadjustment    (GtkViewport *viewport,
                                             GtkAdjustment *adjustment);
void        gtk_viewport_set_vadjustment    (GtkViewport *viewport,
                                             GtkAdjustment *adjustment);
void        gtk_viewport_set_shadow_type    (GtkViewport *viewport,
                                             GtkShadowType type);
GtkShadowType gtk_viewport_get_shadow_type  (GtkViewport *viewport);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkWidget
               +----GtkContainer
                     +----GtkBin
                           +----GtkViewport
```

Implemented Interfaces

GtkViewport implements AtkImplementorIface.

Properties

```
  "hadjustment"          GtkAdjustment         : Read / Write / Construct
  "shadow-type"          GtkShadowType         : Read / Write
  "vadjustment"          GtkAdjustment         : Read / Write / Construct
```

Signal Prototypes

```
"set-scroll-adjustments"
            void        user_function      (GtkViewport *viewport,
                                            GtkAdjustment *arg1,
                                            GtkAdjustment *arg2,
                                            gpointer user_data);
```

Description

Details

GtkViewport

```
typedef struct _GtkViewport GtkViewport;
```

---

gtk_viewport_new ()

```
GtkWidget*  gtk_viewport_new                (GtkAdjustment *hadjustment,
```

```
                                                GtkAdjustment *vadjustment);
```

Creates a new GtkViewport with the given adjustments.

hadjustment : horizontal adjustment.
vadjustment : vertical adjustment.
Returns : a new GtkViewport.

---

gtk_viewport_get_hadjustment ()

```
GtkAdjustment* gtk_viewport_get_hadjustment (GtkViewport *viewport);
```

Returns the horizontal adjustment of the viewport.

viewport : a GtkViewport.
Returns : the horizontal adjustment of *viewport*.

---

gtk_viewport_get_vadjustment ()

```
GtkAdjustment* gtk_viewport_get_vadjustment (GtkViewport *viewport);
```

Returns the vertical adjustment of the viewport.

viewport : a GtkViewport.
Returns : the vertical adjustment of *viewport*.

---

gtk_viewport_set_hadjustment ()

```
void        gtk_viewport_set_hadjustment    (GtkViewport *viewport,
                                             GtkAdjustment *adjustment);
```

Sets the horizontal adjustment of the viewport.

viewport : a GtkViewport.
adjustment : a GtkAdjustment.

---

gtk_viewport_set_vadjustment ()

```
void        gtk_viewport_set_vadjustment    (GtkViewport *viewport,
                                             GtkAdjustment *adjustment);
```

Sets the vertical adjustment of the viewport.

viewport : a GtkViewport.
adjustment : a GtkAdjustment.

---

gtk_viewport_set_shadow_type ()

```
void        gtk_viewport_set_shadow_type    (GtkViewport *viewport,
                                             GtkShadowType type);
```

Sets the shadow type of the viewport.

viewport : a GtkViewport.
type : the new shadow type.

---

gtk_viewport_get_shadow_type ()

```
GtkShadowType gtk_viewport_get_shadow_type  (GtkViewport *viewport);
```

Gets the shadow type of the GtkViewport. See `gtk_viewport_set_shadow_type()`.

viewport : a GtkViewport

Returns : the shadow type

Properties

The "hadjustment" property

```
  "hadjustment"          GtkAdjustment         : Read / Write / Construct
```

The GtkAdjustment that determines the values of the horizontal position for this viewport.

The "shadow-type" property

```
  "shadow-type"          GtkShadowType         : Read / Write
```

Determines how the shadowed box around the viewport is drawn.

Default value: GTK_SHADOW_IN

The "vadjustment" property

```
  "vadjustment"          GtkAdjustment         : Read / Write / Construct
```

The GtkAdjustment that determines the values of the vertical position for this viewport.

Signals

The "set-scroll-adjustments" signal

```
void        user_function                     (GtkViewport *viewport,
                                               GtkAdjustment *arg1,
```

```
                                                GtkAdjustment *arg2,
                                                gpointer user_data);
```

viewport : the object which received the signal.

arg1 :

arg2 :

user_data : user data set when the signal handler was connected.

<< GtkTooltips **GtkAccessible >>**

GtkAccessible

GtkAccessible — Accessibility support for widgets

Synopsis

```
#include <gtk/gtk.h>

            GtkAccessible;
void        gtk_accessible_connect_widget_destroyed
                                            (GtkAccessible *accessible);
```

Object Hierarchy

```
  GObject
   +----AtkObject
         +----GtkAccessible
```

Description

Details

GtkAccessible

```
typedef struct _GtkAccessible GtkAccessible;
```

gtk_accessible_connect_widget_destroyed ()

```
void        gtk_accessible_connect_widget_destroyed
                                            (GtkAccessible *accessible);
```

This function specifies the callback function to be called when the widget corresponding to a GtkAccessible is destroyed.

accessible : a GtkAccessible

<< GtkViewport **Abstract Base Classes >>**

Abstract Base Classes

GtkBin - A container with just one child
GtkBox - Base class for box containers
GtkButtonBox - Base class for GtkHButtonBox and GtkVButtonBox
GtkContainer - Base class for widgets which contain other widgets
GtkItem - Abstract base class for GtkMenuItem, GtkListItem and GtkTreeItem
GtkMisc - Base class for widgets with alignments and padding
GtkObject - The base class of the GTK+ type hierarchy
GtkPaned - Base class for widgets with two adjustable panes
GtkRange - Base class for widgets which visualize an adjustment
GtkScale - Base class for GtkHScale and GtkVScale
GtkScrollbar - Base class for GtkHScrollbar and GtkVScrollbar
GtkSeparator - Base class for GtkHSeparator and GtkVSeparator
GtkWidget - Base class for all widgets
GtkIMContext - Base class for input method contexts

GtkBin

GtkBin — A container with just one child

Synopsis

```
#include <gtk/gtk.h>


            GtkBin;
GtkWidget*  gtk_bin_get_child               (GtkBin *bin);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkWidget
               +----GtkContainer
                     +----GtkBin
                           +----GtkWindow
                           +----GtkAlignment
                           +----GtkFrame
                           +----GtkButton
                           +----GtkItem
                           +----GtkComboBox
                           +----GtkEventBox
                           +----GtkExpander
                           +----GtkHandleBox
```

```
                                +----GtkToolItem
                                +----GtkScrolledWindow
                                +----GtkViewport
```

Implemented Interfaces

GtkBin implements AtkImplementorIface.

Description

The GtkBin widget is a container with just one child. It is not very useful itself, but it is useful for deriving subclasses, since it provides common code needed for handling a single child widget.

Many GTK+ widgets are subclasses of GtkBin, including GtkWindow, GtkButton, GtkFrame, GtkHandleBox, and GtkScrolledWindow.

Details

GtkBin

```
typedef struct _GtkBin GtkBin;
```

The GtkBin-struct struct contains the following fields. (These fields should be considered read-only. They should never be set by an application.)

GtkWidget *child;	the child widget.

---

gtk_bin_get_child ()

```
GtkWidget*  gtk_bin_get_child                (GtkBin *bin);
```

Gets the child of the GtkBin, or NULL if the bin contains no child widget. The returned widget does not

have a reference added, so you do not need to unref it.

bin : a GtkBin

Returns : pointer to child of the GtkBin

<< Abstract Base Classes **GtkBox >>**

GtkBox

GtkBox — Base class for box containers

Synopsis

```
#include <gtk/gtk.h>

            GtkBox;
            GtkBoxChild;
void        gtk_box_pack_start              (GtkBox *box,
                                             GtkWidget *child,
                                             gboolean expand,
                                             gboolean fill,
                                             guint padding);
void        gtk_box_pack_end                (GtkBox *box,
                                             GtkWidget *child,
                                             gboolean expand,
                                             gboolean fill,
                                             guint padding);
void        gtk_box_pack_start_defaults     (GtkBox *box,
                                             GtkWidget *widget);
void        gtk_box_pack_end_defaults       (GtkBox *box,
                                             GtkWidget *widget);
gboolean    gtk_box_get_homogeneous         (GtkBox *box);
void        gtk_box_set_homogeneous         (GtkBox *box,
                                             gboolean homogeneous);
gint        gtk_box_get_spacing             (GtkBox *box);
void        gtk_box_set_spacing             (GtkBox *box,
                                             gint spacing);
void        gtk_box_reorder_child           (GtkBox *box,
```

```
                                                    GtkWidget *child,
                                                    gint position);
void        gtk_box_query_child_packing     (GtkBox *box,
                                                    GtkWidget *child,
                                                    gboolean *expand,
                                                    gboolean *fill,
                                                    guint *padding,
                                                    GtkPackType *pack_type);
void        gtk_box_set_child_packing       (GtkBox *box,
                                                    GtkWidget *child,
                                                    gboolean expand,
                                                    gboolean fill,
                                                    guint padding,
                                                    GtkPackType pack_type);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkWidget
               +----GtkContainer
                     +----GtkBox
                           +----GtkButtonBox
                           +----GtkVBox
                           +----GtkHBox
```

Implemented Interfaces

GtkBox implements AtkImplementorIface.

Properties

```
  "homogeneous"          gboolean              : Read / Write
  "spacing"              gint                  : Read / Write
```

Child Properties

```
  "expand"               gboolean              : Read / Write
  "fill"                 gboolean              : Read / Write
  "pack-type"            GtkPackType           : Read / Write
  "padding"              guint                 : Read / Write
  "position"             gint                  : Read / Write
```

Description

GtkBox is an abstract widget which encapsulates functionallity for a particular kind of container, one that organizes a variable number of widgets into a rectangular area. GtkBox currently has two derived classes, GtkHBox and GtkVBox.

The rectangular area of a GtkBox is organized into either a single row or a single column of child widgets depending upon whether the box is of type GtkHBox or GtkVBox, respectively. Thus, all children of a GtkBox are allocated one dimension in common, which is the height of a row, or the width of a column.

GtkBox uses a notion of *packing*. Packing refers to adding widgets with reference to a particular position in a GtkContainer. For a GtkBox, there are two reference positions: the *start* and the *end* of the box. For a GtkVBox, the start is defined as the top of the box and the end is defined as the bottom. For a GtkHBox the start is defined as the left side and the end is defined as the right side.

Use repeated calls to `gtk_box_pack_start()` to pack widgets into a GtkBox from start to end. Use `gtk_box_pack_end()` to add widgets from end to start. You may intersperse these calls and add widgets from both ends of the same GtkBox.

Use `gtk_box_pack_start_defaults()` or `gtk_box_pack_end_defaults()` to pack widgets into a GtkBox if you do not need to specify the *`expand`*, *`fill`*, or *`padding`* attributes of the child to be added.

Because GtkBox is a GtkContainer, you may also use `gtk_container_add()` to insert widgets into the box, and they will be packed as if with `gtk_box_pack_start_defaults()`. Use `gtk_container_remove()` to remove widgets from the GtkBox.

Use `gtk_box_set_homogeneous()` to specify whether or not all children of the GtkBox are forced to get the same amount of space.

Use `gtk_box_set_spacing()` to determine how much space will be minimally placed between all children in the GtkBox.

Use `gtk_box_reorder_child()` to move a GtkBox child to a different place in the box.

Use `gtk_box_set_child_packing()` to reset the *`expand`*, *`fill`*, and *`padding`* attributes of any GtkBox child. Use `gtk_box_query_child_packing()` to query these fields.

Details

GtkBox

```
typedef struct {
  GList *children;
  gint16 spacing;
  guint homogeneous : 1;
} GtkBox;
```

The GtkBox-struct describes an instance of GtkBox and contains the following fields. (These fields should be considered read-only. They should never be set by an application.)

GList * *`children`*;	a list of children belonging the GtkBox. The data is a list of structures of type GtkBoxChild-struct.
gint16 *`spacing`*;	the number of pixels to put between children of the GtkBox, zero by default. Use `gtk_box_set_spacing()` to set this field.
guint *`homogeneous`*;	a flag that if TRUE forces all children to get equal space in the GtkBox; FALSE by default. Use `gtk_box_set_homogeneous()` to set this field.

GtkBoxChild

```
typedef struct {
  GtkWidget *widget;
  guint16 padding;
  guint expand : 1;
  guint fill : 1;
  guint pack : 1;
  guint is_secondary : 1;
} GtkBoxChild;
```

The GtkBoxChild-struct holds a child widget of GtkBox and describes how the child is to be packed into the GtkBox. Use `gtk_box_query_child_packing()` and `gtk_box_set_child_packing()` to query and reset the *`padding`*, *`expand`*, *`fill`*, and *`pack`* fields.

GtkBoxChild-struct contains the following fields. (These fields should be considered read-only. They should never be directly set by an application.)

GtkWidget * *`widget`*;	the child widget, packed into the GtkBox.
guint16 *`padding`*;	the number of extra pixels to put between this child and its neighbors, set when packed, zero by default.
guint *`expand`*;	flag indicates whether extra space should be given to this child. Any extra space given to the parent GtkBox is divided up among all children with this attribute set to TRUE; set when packed, TRUE by default.
guint *`fill`*;	flag indicates whether any extra space given to this child due to its *`expand`* attribute being set is actually allocated to the child, rather than being used as padding around the widget; set when packed, TRUE by default.
guint *`pack`*;	one of GtkPackType indicating whether the child is packed with reference to the start (top/left) or end (bottom/right) of the GtkBox.

---

gtk_box_pack_start ()

```
void        gtk_box_pack_start              (GtkBox *box,
                                             GtkWidget *child,
```

```
                                    gboolean expand,
                                    gboolean fill,
                                    guint padding);
```

Adds *child* to *box*, packed with reference to the start of *box*. The *child* is packed after any other child packed with reference to the start of *box*.

box :	a GtkBox.
child :	the GtkWidget to be added to *box*.
expand :	TRUE if the new child is to be given extra space allocated to *box*. The extra space will be divided evenly between all children of *box* that use this option.
fill :	TRUE if space given to *child* by the *expand* option is actually allocated to *child*, rather than just padding it. This parameter has no effect if *expand* is set to FALSE. A child is always allocated the full height of a GtkHBox and the full width of a GtkVBox. This option affects the other dimension.
padding :	extra space in pixels to put between this child and its neighbors, over and above the global amount specified by *spacing* in GtkBox-struct. If *child* is a widget at one of the reference ends of *box*, then *padding* pixels are also put between *child* and the reference edge of *box*.

---

gtk_box_pack_end ()

```
void        gtk_box_pack_end                (GtkBox *box,
                                             GtkWidget *child,
                                             gboolean expand,
                                             gboolean fill,
                                             guint padding);
```

Adds *child* to *box*, packed with reference to the end of *box*. The *child* is packed after (away from end of) any other child packed with reference to the end of *box*.

box :	a GtkBox.
child :	the GtkWidget to be added to *box*.

expand : TRUE if the new child is to be given extra space allocated to *box*. The extra space will be divided evenly between all children of *box* that use this option.

fill : TRUE if space given to *child* by the *expand* option is actually allocated to *child*, rather than just padding it. This parameter has no effect if *expand* is set to FALSE. A child is always allocated the full height of a GtkHBox and the full width of a GtkVBox. This option affects the other dimension.

padding : extra space in pixels to put between this child and its neighbors, over and above the global amount specified by *spacing* in GtkBox-struct. If *child* is a widget at one of the reference ends of *box*, then *padding* pixels are also put between *child* and the reference edge of *box*.

---

gtk_box_pack_start_defaults ()

```
void        gtk_box_pack_start_defaults     (GtkBox *box,
                                             GtkWidget *widget);
```

Adds *widget* to *box*, packed with reference to the start of *box*. The child is packed after any other child packed with reference to the start of *box*.

Parameters for how to pack the child *widget*, *expand*, *fill*, and *padding* in GtkBoxChild-struct, are given their default values, TRUE, TRUE, and 0, respectively.

box : a GtkBox.

widget : the GtkWidget to be added to *box*.

---

gtk_box_pack_end_defaults ()

```
void        gtk_box_pack_end_defaults       (GtkBox *box,
                                             GtkWidget *widget);
```

Adds *widget* to *box*, packed with reference to the end of *box*. The child is packed after (away from end of) any other child packed with reference to the end of *box*.

Parameters for how to pack the child *widget*, *expand*, *fill*, and *padding* in GtkBoxChild-struct, are given their default values, TRUE, TRUE, and 0, respectively.

box : a GtkBox.

widget : the GtkWidget to be added to *box*.

---

gtk_box_get_homogeneous ()

```
gboolean    gtk_box_get_homogeneous         (GtkBox *box);
```

Returns whether the box is homogeneous (all children are the same size). See gtk_box_set_homogeneous().

box : a GtkBox

Returns : TRUE if the box is homogeneous.

---

gtk_box_set_homogeneous ()

```
void        gtk_box_set_homogeneous         (GtkBox *box,
                                             gboolean homogeneous);
```

Sets the *homogeneous* field of GtkBox-struct, controlling whether or not all children of *box* are given equal space in the box.

box : a GtkBox.

homogeneous : a boolean value, TRUE to create equal allotments, FALSE for variable allotments.

---

gtk_box_get_spacing ()

```
```

```
gint        gtk_box_get_spacing             (GtkBox *box);
```

Gets the value set by gtk_box_set_spacing().

box : a GtkBox

Returns : spacing between children

---

gtk_box_set_spacing ()

```
void        gtk_box_set_spacing             (GtkBox *box,
                                             gint spacing);
```

Sets the *spacing* field of GtkBox-struct, which is the number of pixels to place between children of *box*.

box : a GtkBox.

spacing : the number of pixels to put between children.

---

gtk_box_reorder_child ()

```
void        gtk_box_reorder_child           (GtkBox *box,
                                             GtkWidget *child,
                                             gint position);
```

Moves *child* to a new *position* in the list of *box* children. The list is the *children* field of GtkBox-struct, and contains both widgets packed GTK_PACK_START as well as widgets packed GTK_PACK_END, in the order that these widgets were added to *box*.

A widget's position in the *box* children list determines where the widget is packed into *box*. A child widget at some position in the list will be packed just after all other widgets of the same packing type that appear earlier in the list.

box : a GtkBox.

child : the GtkWidget to move.

position : the new position for *child* in the *children* list of GtkBox-struct, starting from 0. If negative, indicates the end of the list.

---

gtk_box_query_child_packing ()

```
void        gtk_box_query_child_packing     (GtkBox *box,
                                             GtkWidget *child,
                                             gboolean *expand,
                                             gboolean *fill,
                                             guint *padding,
                                             GtkPackType *pack_type);
```

Returns information about how *child* is packed into *box*.

box : a GtkBox.

child : the GtkWidget of the child to query.

expand : the returned value of the *expand* field in GtkBoxChild-struct.

fill : the returned value of the *fill* field in GtkBoxChild-struct.

padding : the returned value of the *padding* field in GtkBoxChild-struct.

pack_type : the returned value of the *pack* field in GtkBoxChild-struct.

---

gtk_box_set_child_packing ()

```
void        gtk_box_set_child_packing       (GtkBox *box,
                                             GtkWidget *child,
                                             gboolean expand,
                                             gboolean fill,
                                             guint padding,
                                             GtkPackType pack_type);
```

Sets the way *child* is packed into *box*.

box :	a GtkBox.
child :	the GtkWidget of the child to set.
expand :	the new value of the *expand* field in GtkBoxChild-struct.
fill :	the new value of the *fill* field in GtkBoxChild-struct.
padding :	the new value of the *padding* field in GtkBoxChild-struct.
pack_type :	the new value of the *pack* field in GtkBoxChild-struct.

Properties

The "homogeneous" property

```
"homogeneous"          gboolean                : Read / Write
```

Whether the children should all be the same size.

Default value: FALSE

The "spacing" property

```
"spacing"              gint                    : Read / Write
```

The amount of space between children.

Allowed values: >= 0

Default value: 0

Child Properties

The "expand" child property

"expand"	gboolean	: Read / Write

Whether the child should receive extra space when the parent grows.

Default value: TRUE

The "fill" child property

"fill"	gboolean	: Read / Write

Whether extra space given to the child should be allocated to the child or used as padding.

Default value: TRUE

The "pack-type" child property

"pack-type"	GtkPackType	: Read / Write

A GtkPackType indicating whether the child is packed with reference to the start or end of the parent.

Default value: GTK_PACK_START

The "padding" child property

"padding"	guint	: Read / Write

Extra space to put between the child and its neighbors, in pixels.

Allowed values: <= G_MAXINT

Default value: 0

The "position" child property

```
"position"             gint                  : Read / Write
```

The index of the child in the parent.

Allowed values: >= -1

Default value: 0

See Also

GtkHBox	a derived class that organizes widgets into a row.
GtkVBox	a derived class that organizes widgets into a column.
GtkFrame	a GtkWidget useful for drawing a border around a GtkBox.
GtkTable	a GtkContainer for organizing widgets into a grid, rather than independent rows or columns.
GtkLayout	a GtkContainer for organizing widgets into arbitrary layouts.

<< GtkBin **GtkButtonBox >>**

GtkButtonBox

GtkButtonBox — Base class for GtkHButtonBox and GtkVButtonBox

Synopsis

```
#include <gtk/gtk.h>


            GtkButtonBox;
#define     GTK_BUTTONBOX_DEFAULT
#define     gtk_button_box_get_spacing      (b)
GtkButtonBoxStyle gtk_button_box_get_layout (GtkButtonBox *widget);
void        gtk_button_box_get_child_size   (GtkButtonBox *widget,
                                             gint *min_width,
                                             gint *min_height);
void        gtk_button_box_get_child_ipadding
                                            (GtkButtonBox *widget,
                                             gint *ipad_x,
                                             gint *ipad_y);
gboolean    gtk_button_box_get_child_secondary
                                            (GtkButtonBox *widget,
                                             GtkWidget *child);
#define     gtk_button_box_set_spacing      (b,s)
void        gtk_button_box_set_layout       (GtkButtonBox *widget,
                                             GtkButtonBoxStyle layout_style);
void        gtk_button_box_set_child_size   (GtkButtonBox *widget,
                                             gint min_width,
                                             gint min_height);
void        gtk_button_box_set_child_ipadding
                                            (GtkButtonBox *widget,
                                             gint ipad_x,
                                             gint ipad_y);
void        gtk_button_box_set_child_secondary
                                            (GtkButtonBox *widget,
                                             GtkWidget *child,
                                             gboolean is_secondary);
```

Object Hierarchy

```
GObject
 +----GtkObject
       +----GtkWidget
             +----GtkContainer
                   +----GtkBox
                         +----GtkButtonBox
                               +----GtkHButtonBox
                               +----GtkVButtonBox
```

Implemented Interfaces

GtkButtonBox implements AtkImplementorIface.

Properties

```
"layout-style"         GtkButtonBoxStyle     : Read / Write
```

Child Properties

```
"secondary"            gboolean              : Read / Write
```

Style Properties

```
"child-internal-pad-x" gint                  : Read
"child-internal-pad-y" gint                  : Read
"child-min-height"     gint                  : Read
"child-min-width"      gint                  : Read
```

Description

The primary purpose of this class is to keep track of the various properties of GtkHButtonBox and GtkVButtonBox widgets.

`gtk_button_box_get_child_size()` retrieves the minimum width and height for widgets in a given button box. `gtk_button_box_set_child_size()` allows those properties to be changed.

The internal padding of buttons can be retrieved and changed per button box using `gtk_button_box_get_child_ipadding()` and `gtk_button_box_set_child_ipadding()` respectively.

`gtk_button_box_get_spacing()` and `gtk_button_box_set_spacing()` retrieve and change default number of pixels between buttons, respectively.

`gtk_button_box_get_layout()` and `gtk_button_box_set_layout()` retrieve and alter the method used to spread the buttons in a button box across the container, respectively.

Details

GtkButtonBox

```
typedef struct _GtkButtonBox GtkButtonBox;
```

This is a read-only struct; no members should be modified directly.

GTK_BUTTONBOX_DEFAULT

```
#define GTK_BUTTONBOX_DEFAULT -1
```

Used internally only.

---

gtk_button_box_get_spacing()

```
#define gtk_button_box_get_spacing(b)   gtk_box_get_spacing (GTK_BOX (b))
```

Warning

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

Retrieves how much space a button box is placing between each child button.

b :
Returns : the current spacing applied to the buttons in *`widget`*.

---

gtk_button_box_get_layout ()

```
GtkButtonBoxStyle gtk_button_box_get_layout (GtkButtonBox *widget);
```

Retrieves the method being used to arrange the buttons in a button box.

`widget` : a GtkButtonBox.
Returns : the method used to layout buttons in *`widget`*.

---

gtk_button_box_get_child_size ()

```
void        gtk_button_box_get_child_size   (GtkButtonBox *widget,
                                             gint *min_width,
                                             gint *min_height);
```

Warning

`gtk_button_box_get_child_size` is deprecated and should not be used in newly-written code. Use the style properties `"child-min-width/-height"` instead.

Retrieves the current width and height of all child widgets in a button box. *`min_width`* and *`min_height`* are filled with those values, respectively.

`widget` : a GtkButtonBox.
`min_width` : the width of the buttons contained by *`widget`*.
`min_height` : the height of the buttons contained by *`widget`*.

gtk_button_box_get_child_ipadding ()

```
void        gtk_button_box_get_child_ipadding
                                            (GtkButtonBox *widget,
                                             gint *ipad_x,
                                             gint *ipad_y);
```

Warning

`gtk_button_box_get_child_ipadding` is deprecated and should not be used in newly-written code. Use the style properties `"child-internal-pad-x/-y"` instead.

Gets the default number of pixels that pad the buttons in a given button box.

widget : a GtkButtonBox.
ipad_x : the horizontal padding used by buttons in *widget*.
ipad_y : the vertical padding used by buttons in *widget*.

gtk_button_box_get_child_secondary ()

```
gboolean    gtk_button_box_get_child_secondary
                                            (GtkButtonBox *widget,
                                             GtkWidget *child);
```

Returns whether *child* should appear in a secondary group of children.

widget : a GtkButtonBox
child : a child of *widget*
Returns : whether *child* should appear in a secondary group of children.

Since 2.4

gtk_button_box_set_spacing()

```
#define gtk_button_box_set_spacing(b,s) gtk_box_set_spacing (GTK_BOX (b), s)
```

Warning

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

Sets the amount of spacing between buttons in a given button box.

b :
s :

---

gtk_button_box_set_layout ()

```
void        gtk_button_box_set_layout       (GtkButtonBox *widget,
                                             GtkButtonBoxStyle layout_style);
```

Changes the way buttons are arranged in their container.

widget : a GtkButtonBox.
layout_style : the new layout style.

---

gtk_button_box_set_child_size ()

```
void        gtk_button_box_set_child_size   (GtkButtonBox *widget,
                                             gint min_width,
                                             gint min_height);
```

Warning

`gtk_button_box_set_child_size` is deprecated and should not be used in newly-written code. Use the style properties `"child-min-width/-height"` instead.

Sets a new default size for the children of a given button box.

widget : a GtkButtonBox.
min_width : a default width for buttons in *widget*.

min_height : a default height for buttons in *widget*.

---

gtk_button_box_set_child_ipadding ()

```
void        gtk_button_box_set_child_ipadding
                                            (GtkButtonBox *widget,
                                             gint ipad_x,
                                             gint ipad_y);
```

Warning

`gtk_button_box_set_child_ipadding` is deprecated and should not be used in newly-written code. Use the style properties `"child-internal-pad-x/-y"` instead.

Changes the amount of internal padding used by all buttons in a given button box.

widget : a GtkButtonBox.

ipad_x : the horizontal padding that should be used by each button in *widget*.

ipad_y : the vertical padding that should be used by each button in *widget*.

---

gtk_button_box_set_child_secondary ()

```
void        gtk_button_box_set_child_secondary
                                            (GtkButtonBox *widget,
                                             GtkWidget *child,
                                             gboolean is_secondary);
```

Sets whether *child* should appear in a secondary group of children. A typical use of a secondary child is the help button in a dialog.

This group appears after the other children if the style is `GTK_BUTTONBOX_START`, `GTK_BUTTONBOX_SPREAD` or `GTK_BUTTONBOX_EDGE`, and before the the other children if the style is `GTK_BUTTONBOX_END`. For horizontal button boxes, the definition of before/after depends on direction of the widget (see `gtk_widget_set_direction()`). If the style is `GTK_BUTTONBOX_START` or `GTK_BUTTONBOX_END`, then the secondary children are aligned at the other end of the button box from the main children. For the other styles, they appear immediately next to the main children.

widget : a GtkButtonBox
child : a child of *widget*
is_secondary : if TRUE, the *child* appears in a secondary group of the button box.

Properties

The "layout-style" property

```
"layout-style"         GtkButtonBoxStyle     : Read / Write
```

How to layout the buttons in the box. Possible values are default, spread, edge, start and end.

Default value: GTK_BUTTONBOX_DEFAULT_STYLE

Child Properties

The "secondary" child property

```
"secondary"            gboolean              : Read / Write
```

If TRUE, the child appears in a secondary group of children, suitable for, e.g., help buttons.

Default value: FALSE

Style Properties

The "child-internal-pad-x" style property

```
"child-internal-pad-x" gint                  : Read
```

Amount to increase child's size on either side.

Allowed values: >= 0

Default value: 4

The "child-internal-pad-y" style property

```
  "child-internal-pad-y" gint                : Read
```

Amount to increase child's size on the top and bottom.

Allowed values: >= 0

Default value: 0

The "child-min-height" style property

```
  "child-min-height"     gint                : Read
```

Minimum height of buttons inside the box.

Allowed values: >= 0

Default value: 27

The "child-min-width" style property

```
  "child-min-width"      gint                : Read
```

Minimum width of buttons inside the box.

Allowed values: >= 0

Default value: 85

See Also

GtkVButtonBox Vertical sub-class of GtkButtonBox.

GtkHButtonBox Horizontal sub-class of GtkButtonBox.

<< GtkBox **GtkContainer >>**

GtkContainer

GtkContainer — Base class for widgets which contain other widgets

Synopsis

```
#include <gtk/gtk.h>


            GtkContainer;
#define     GTK_IS_RESIZE_CONTAINER         (widget)
#define     GTK_CONTAINER_WARN_INVALID_CHILD_PROPERTY_ID(object, property_id, pspec)
#define     gtk_container_border_width
void        gtk_container_add               (GtkContainer *container,
                                             GtkWidget *widget);
void        gtk_container_remove            (GtkContainer *container,
                                             GtkWidget *widget);
void        gtk_container_add_with_properties
                                            (GtkContainer *container,
                                             GtkWidget *widget,
                                             const gchar *first_prop_name,
                                             ...);
GtkResizeMode gtk_container_get_resize_mode (GtkContainer *container);
void        gtk_container_set_resize_mode   (GtkContainer *container,
                                             GtkResizeMode resize_mode);
void        gtk_container_check_resize      (GtkContainer *container);
void        gtk_container_foreach           (GtkContainer *container,
                                             GtkCallback callback,
                                             gpointer callback_data);
void        gtk_container_foreach_full      (GtkContainer *container,
                                             GtkCallback callback,
                                             GtkCallbackMarshal marshal,
                                             gpointer callback_data,
                                             GtkDestroyNotify notify);
#define     gtk_container_children
GList*      gtk_container_get_children      (GtkContainer *container);
void        gtk_container_set_reallocate_redraws
                                            (GtkContainer *container,
                                             gboolean needs_redraws);
void        gtk_container_set_focus_child   (GtkContainer *container,
                                             GtkWidget *child);
```

```
GtkAdjustment* gtk_container_get_focus_vadjustment
                                            (GtkContainer *container);
void        gtk_container_set_focus_vadjustment
                                            (GtkContainer *container,
                                             GtkAdjustment *adjustment);
GtkAdjustment* gtk_container_get_focus_hadjustment
                                            (GtkContainer *container);
void        gtk_container_set_focus_hadjustment
                                            (GtkContainer *container,
                                             GtkAdjustment *adjustment);
void        gtk_container_resize_children   (GtkContainer *container);
GType       gtk_container_child_type        (GtkContainer *container);
void        gtk_container_child_get         (GtkContainer *container,
                                             GtkWidget *child,
                                             const gchar *first_prop_name,
                                             ...);
void        gtk_container_child_set         (GtkContainer *container,
                                             GtkWidget *child,
                                             const gchar *first_prop_name,
                                             ...);
void        gtk_container_child_get_property
                                            (GtkContainer *container,
                                             GtkWidget *child,
                                             const gchar *property_name,
                                             GValue *value);
void        gtk_container_child_set_property
                                            (GtkContainer *container,
                                             GtkWidget *child,
                                             const gchar *property_name,
                                             const GValue *value);
void        gtk_container_child_get_valist  (GtkContainer *container,
                                             GtkWidget *child,
                                             const gchar *first_property_name,
                                             va_list var_args);
void        gtk_container_child_set_valist  (GtkContainer *container,
                                             GtkWidget *child,
                                             const gchar *first_property_name,
                                             va_list var_args);
void        gtk_container_forall            (GtkContainer *container,
                                             GtkCallback callback,
                                             gpointer callback_data);
guint       gtk_container_get_border_width  (GtkContainer *container);
void        gtk_container_set_border_width  (GtkContainer *container,
                                             guint border_width);
void        gtk_container_propagate_expose  (GtkContainer *container,
                                             GtkWidget *child,
                                             GdkEventExpose *event);
```

```
gboolean    gtk_container_get_focus_chain   (GtkContainer *container,
                                             GList **focusable_widgets);
void        gtk_container_set_focus_chain   (GtkContainer *container,
                                             GList *focusable_widgets);
void        gtk_container_unset_focus_chain (GtkContainer *container);
GParamSpec* gtk_container_class_find_child_property
                                            (GObjectClass *cclass,
                                             const gchar *property_name);
void        gtk_container_class_install_child_property
                                            (GtkContainerClass *cclass,
                                             guint property_id,
                                             GParamSpec *pspec);
GParamSpec** gtk_container_class_list_child_properties
                                            (GObjectClass *cclass,
                                             guint *n_properties);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkWidget
               +----GtkContainer
                     +----GtkBin
                     +----GtkBox
                     +----GtkCList
                     +----GtkFixed
                     +----GtkPaned
                     +----GtkIconView
                     +----GtkLayout
                     +----GtkList
                     +----GtkMenuShell
                     +----GtkNotebook
                     +----GtkSocket
                     +----GtkTable
                     +----GtkTextView
                     +----GtkToolbar
                     +----GtkTree
                     +----GtkTreeView
```

Implemented Interfaces

GtkContainer implements AtkImplementorIface.

Properties

```
  "border-width"         guint                : Read / Write
  "child"                GtkWidget            : Write
  "resize-mode"          GtkResizeMode        : Read / Write
```

Signal Prototypes

```
"add"       void        user_function      (GtkContainer *container,
                                            GtkWidget *widget,
                                            gpointer user_data);
"check-resize"
            void        user_function      (GtkContainer *container,
                                            gpointer user_data);
"remove"    void        user_function      (GtkContainer *container,
                                            GtkWidget *widget,
                                            gpointer user_data);
"set-focus-child"
            void        user_function      (GtkContainer *container,
                                            GtkWidget *widget,
                                            gpointer user_data);
```

Description

A GTK+ user interface is constructed by nesting widgets inside widgets. Container widgets are the inner nodes in the resulting tree of widgets: they contain other widgets. So, for example, you might have a GtkWindow containing a GtkFrame containing a GtkLabel. If you wanted an image instead of a textual label inside the frame, you might replace the GtkLabel widget with a GtkImage widget.

There are two major kinds of container widgets in GTK+. Both are subclasses of the abstract GtkContainer base class.

The first type of container widget has a single child widget and derives from GtkBin. These containers are *decorators*, which add some kind of functionality to the child. For example, a GtkButton makes its child into a clickable button; a GtkFrame draws a frame around its child and a GtkWindow places its child widget inside a top-level window.

The second type of container can have more than one child; its purpose is to manage *layout*. This means that these containers assign sizes and positions to their children. For example, a GtkHBox arranges its children in a horizontal row, and a GtkTable arranges the widgets it contains in a two-dimensional grid.

To fulfill its task, a layout container must negotiate the size requirements with its parent and its children. This negotiation is carried out in two phases, *size requisition* and *size allocation*.

Size Requisition

The size requisition of a widget is it's desired width and height. This is represented by a GtkRequisition.

How a widget determines its desired size depends on the widget. A GtkLabel, for example, requests enough space to display all its text. Container widgets generally base their size request on the requisitions of their children.

The size requisition phase of the widget layout process operates top-down. It starts at a top-level widget, typically a GtkWindow. The top-level widget asks its child for its size requisition by calling `gtk_widget_size_request()`. To determine its requisition, the child asks its own children for their requisitions and so on. Finally, the top-level widget will get a requisition back from its child.

Size Allocation

When the top-level widget has determined how much space its child would like to have, the second phase of the size negotiation, size allocation, begins. Depending on its configuration (see `gtk_window_set_resizable()`), the top-level widget may be able to expand in order to satisfy the size request or it may have to ignore the size request and keep its fixed size. It then tells its child widget how much space it gets by calling `gtk_widget_size_allocate()`. The child widget divides the space among its children and tells each child how much space it got, and so on. Under normal circumstances, a GtkWindow will always give its child the amount of space the child requested.

A child's size allocation is represented by a GtkAllocation. This struct contains not only a width and height, but also a position (i.e. X and Y coordinates), so that containers can tell their children not only how much space they have gotten, but also where they are positioned inside the space available to the container.

Widgets are required to honor the size allocation they receive; a size request is only a request, and widgets must be able to cope with any size.

Child properties

GtkContainer introduces *child properties* - these are object properties that are not specific to either the container or the contained widget, but rather to their relation. Typical examples of child properties are the position or pack-type of a widget which is contained in a GtkBox.

Use `gtk_container_class_install_child_property()` to install child properties for a container class and `gtk_container_class_find_child_property()` or `gtk_container_class_list_child_properties()` to get information about existing child properties.

To set the value of a child property, use `gtk_container_child_set_property()`, `gtk_container_child_set()` or `gtk_container_child_set_valist()`. To obtain the value of a child property, use `gtk_container_child_get_property()`, `gtk_container_child_get()` or `gtk_container_child_get_valist()`. To emit notification about child property changes, use `gtk_widget_child_notify()`.

Details

GtkContainer

```
typedef struct _GtkContainer GtkContainer;
```

GTK_IS_RESIZE_CONTAINER()

```
#define GTK_IS_RESIZE_CONTAINER(widget) (GTK_IS_CONTAINER (widget) &&
((GtkContainer*) (widget))->resize_mode != GTK_RESIZE_PARENT)
```

widget :

GTK_CONTAINER_WARN_INVALID_CHILD_PROPERTY_ID()

```
#define     GTK_CONTAINER_WARN_INVALID_CHILD_PROPERTY_ID(object, property_id, pspec)
```

object :
property_id :
pspec :

---

gtk_container_border_width

```
#define gtk_container_border_width              gtk_container_set_border_width
```

Warning

`gtk_container_border_width` is deprecated and should not be used in newly-written code. Use `gtk_container_set_border_width()` instead.

Does the same as `gtk_container_set_border_width()`.

---

gtk_container_add ()

```
void        gtk_container_add               (GtkContainer *container,
                                             GtkWidget *widget);
```

Adds *widget* to *container*. Typically used for simple containers such as GtkWindow, GtkFrame, or GtkButton; for more complicated layout containers such as GtkBox or GtkTable, this function will pick default packing parameters that may not be correct. So consider functions such as gtk_box_pack_start() and gtk_table_attach() as an alternative to gtk_container_add() in those cases. A widget may be added to only one container at a time; you can't place the same widget inside two different containers.

container : a GtkContainer
widget : a widget to be placed inside *container*

---

gtk_container_remove ()

```
void        gtk_container_remove            (GtkContainer *container,
                                             GtkWidget *widget);
```

Removes *widget* from *container*. *widget* must be inside *container*. Note that *container* will own a reference to *widget*, and that this may be the last reference held; so removing a widget from its container can destroy that widget. If you want to use *widget* again, you need to add a reference to it while it's not inside a container, using g_object_ref(). If you don't want to use *widget* again it's usually more efficient to simply destroy it directly using gtk_widget_destroy() since this will remove it from the container and help break any circular reference count cycles.

container : a GtkContainer
widget : a current child of *container*

---

gtk_container_add_with_properties ()

```
void        gtk_container_add_with_properties
                                            (GtkContainer *container,
                                             GtkWidget *widget,
                                             const gchar *first_prop_name,
                                             ...);
```

Adds *widget* to *container*, setting child properties at the same time. See gtk_container_add() and gtk_container_child_set() for more details.

container : a GtkContainer
widget : a widget to be placed inside *container*

first_prop_name : the name of the first child property to set

... : a NULL-terminated list of property names and values, starting with *first_prop_name*.

---

gtk_container_get_resize_mode ()

```
GtkResizeMode gtk_container_get_resize_mode (GtkContainer *container);
```

Returns the resize mode for the container. See gtk_container_set_resize_mode().

container : a GtkContainer

Returns : the current resize mode

---

gtk_container_set_resize_mode ()

```
void        gtk_container_set_resize_mode   (GtkContainer *container,
                                             GtkResizeMode resize_mode);
```

Sets the resize mode for the container.

The resize mode of a container determines whether a resize request will be passed to the container's parent, queued for later execution or executed immediately.

container : a GtkContainer.

resize_mode : the new resize mode.

---

gtk_container_check_resize ()

```
void        gtk_container_check_resize      (GtkContainer *container);
```

container :

---

gtk_container_foreach ()

```
void        gtk_container_foreach           (GtkContainer *container,
                                             GtkCallback callback,
```

```
                                                gpointer callback_data);
```

Invokes *callback* on each non-internal child of *container*. See gtk_container_forall() for details on what constitutes an "internal" child. Most applications should use gtk_container_foreach(), rather than gtk_container_forall().

container : a GtkContainer
callback : a callback
callback_data : callback user data

---

gtk_container_foreach_full ()

```
void        gtk_container_foreach_full      (GtkContainer *container,
                                             GtkCallback callback,
                                             GtkCallbackMarshal marshal,
                                             gpointer callback_data,
                                             GtkDestroyNotify notify);
```

Warning

gtk_container_foreach_full is deprecated and should not be used in newly-written code. Use gtk_container_foreach() instead.

container :
callback :
marshal :
callback_data :
notify :

---

gtk_container_children

```
#define gtk_container_children gtk_container_get_children
```

Warning

gtk_container_children is deprecated and should not be used in newly-written code. Use gtk_container_get_children() instead.

Does the same as gtk_container_get_children().

Returns :

---

gtk_container_get_children ()

```
GList*      gtk_container_get_children      (GtkContainer *container);
```

Returns the the container's non-internal children. See `gtk_container_forall()` for details on what constitutes an "internal" child.

container : a GtkContainer.

Returns : a newly-allocated list of the container's non-internal children.

---

gtk_container_set_reallocate_redraws ()

```
void        gtk_container_set_reallocate_redraws
                                            (GtkContainer *container,
                                             gboolean needs_redraws);
```

Sets the *reallocate_redraws* flag of the container to the given value.

Containers requesting reallocation redraws get automatically redrawn if any of their children changed allocation.

container : a GtkContainer.

needs_redraws : the new value for the container's *reallocate_redraws* flag.

---

gtk_container_set_focus_child ()

```
void        gtk_container_set_focus_child   (GtkContainer *container,
                                             GtkWidget *child);
```

container :

child :

---

gtk_container_get_focus_vadjustment ()

```

```

```
GtkAdjustment* gtk_container_get_focus_vadjustment
                                                    (GtkContainer *container);
```

Retrieves the vertical focus adjustment for the container. See gtk_container_set_focus_vadjustment().

container : a GtkContainer

Returns : the vertical focus adjustment, or NULL if none has been set.

---

gtk_container_set_focus_vadjustment ()

```
void        gtk_container_set_focus_vadjustment
                                                    (GtkContainer *container,
                                                     GtkAdjustment *adjustment);
```

Hooks up an adjustment to focus handling in a container, so when a child of the container is focused, the adjustment is scrolled to show that widget. This function sets the vertical alignment. See gtk_scrolled_window_get_vadjustment() for a typical way of obtaining the adjustment and gtk_container_set_focus_hadjustment() for setting the horizontal adjustment.

The adjustments have to be in pixel units and in the same coordinate system as the allocation for immediate children of the container.

container : a GtkContainer

adjustment : an adjustment which should be adjusted when the focus is moved among the descendents of *container*

---

gtk_container_get_focus_hadjustment ()

```
GtkAdjustment* gtk_container_get_focus_hadjustment
                                                    (GtkContainer *container);
```

Retrieves the horizontal focus adjustment for the container. See gtk_container_set_focus_hadjustment().

container : a GtkContainer

Returns : the horizontal focus adjustment, or NULL if none has been set.

---

gtk_container_set_focus_hadjustment ()

```
void        gtk_container_set_focus_hadjustment
                                            (GtkContainer *container,
                                             GtkAdjustment *adjustment);
```

Hooks up an adjustment to focus handling in a container, so when a child of the container is focused, the adjustment is scrolled to show that widget. This function sets the horizontal alignment. See `gtk_scrolled_window_get_hadjustment()` for a typical way of obtaining the adjustment and `gtk_container_set_focus_vadjustment()` for setting the vertical adjustment.

The adjustments have to be in pixel units and in the same coordinate system as the allocation for immediate children of the container.

container : a GtkContainer

adjustment : an adjustment which should be adjusted when the focus is moved among the descendents of *container*

---

gtk_container_resize_children ()

```
void        gtk_container_resize_children   (GtkContainer *container);
```

container :

---

gtk_container_child_type ()

```
GType       gtk_container_child_type        (GtkContainer *container);
```

Returns the type of the children supported by the container.

Note that this may return `G_TYPE_NONE` to indicate that no more children can be added, e.g. for a GtkPaned which already has two children.

container : a GtkContainer.

Returns : a GType.

---

gtk_container_child_get ()

```
void        gtk_container_child_get         (GtkContainer *container,
                                             GtkWidget *child,
```

```
                                        const gchar *first_prop_name,
                                        ...);
```

Gets the values of one or more child properties for *child* and *container*.

container :	a GtkContainer
child :	a widget which is a child of *container*
first_prop_name :	the name of the first property to get
... :	a NULL-terminated list of property names and GValue*, starting with *first_prop_name*.

---

gtk_container_child_set ()

```
void        gtk_container_child_set         (GtkContainer *container,
                                             GtkWidget *child,
                                             const gchar *first_prop_name,
                                             ...);
```

Sets one or more child properties for *child* and *container*.

container :	a GtkContainer
child :	a widget which is a child of *container*
first_prop_name :	the name of the first property to set
... :	a NULL-terminated list of property names and values, starting with *first_prop_name*.

---

gtk_container_child_get_property ()

```
void        gtk_container_child_get_property
                                            (GtkContainer *container,
                                             GtkWidget *child,
                                             const gchar *property_name,
                                             GValue *value);
```

Gets the value of a child property for *child* and *container*.

container :	a GtkContainer
child :	a widget which is a child of *container*
property_name :	the name of the property to get

value : a location to return the value

---

gtk_container_child_set_property ()

```
void        gtk_container_child_set_property
                                            (GtkContainer *container,
                                             GtkWidget *child,
                                             const gchar *property_name,
                                             const GValue *value);
```

Sets a child property for *child* and *container*.

container : a GtkContainer
child : a widget which is a child of *container*
property_name : the name of the property to set
value : the value to set the property to

---

gtk_container_child_get_valist ()

```
void        gtk_container_child_get_valist  (GtkContainer *container,
                                             GtkWidget *child,
                                             const gchar *first_property_name,
                                             va_list var_args);
```

Gets the values of one or more child properties for *child* and *container*.

container : a GtkContainer
child : a widget which is a child of *container*
first_property_name : the name of the first property to get
var_args : a NULL-terminated list of property names and GValue*, starting with *first_prop_name*.

---

gtk_container_child_set_valist ()

```
void        gtk_container_child_set_valist  (GtkContainer *container,
                                             GtkWidget *child,
                                             const gchar *first_property_name,
                                             va_list var_args);
```

Sets one or more child properties for *child* and *container*.

container :	a GtkContainer
child :	a widget which is a child of *container*
first_property_name :	the name of the first property to set
var_args :	a NULL-terminated list of property names and values, starting with *first_prop_name*.

---

gtk_container_forall ()

```
void        gtk_container_forall            (GtkContainer *container,
                                             GtkCallback callback,
                                             gpointer callback_data);
```

Invokes *callback* on each child of *container*, including children that are considered "internal" (implementation details of the container). "Internal" children generally weren't added by the user of the container, but were added by the container implementation itself. Most applications should use gtk_container_foreach(), rather than gtk_container_forall().

container :	a GtkContainer
callback :	a callback
callback_data :	callback user data

---

gtk_container_get_border_width ()

```
guint       gtk_container_get_border_width  (GtkContainer *container);
```

Retrieves the border width of the container. See gtk_container_set_border_width().

container :	a GtkContainer
Returns :	the current border width

---

gtk_container_set_border_width ()

```
void        gtk_container_set_border_width  (GtkContainer *container,
                                             guint border_width);
```

Sets the border width of the container.

The border width of a container is the amount of space to leave around the outside of the container. The only exception to this is GtkWindow; because toplevel windows can't leave space outside, they leave the space inside. The border is added on all sides of the container. To add space to only one side, one approach is to create a GtkAlignment widget, call `gtk_widget_set_usize()` to give it a size, and place it on the side of the container as a spacer.

container : a GtkContainer

border_width : amount of blank space to leave *outside* the container. Valid values are in the range 0-65535 pixels.

---

gtk_container_propagate_expose ()

```
void        gtk_container_propagate_expose  (GtkContainer *container,
                                             GtkWidget *child,
                                             GdkEventExpose *event);
```

When a container receives an expose event, it must send synthetic expose events to all children that don't have their own GdkWindows. This function provides a convenient way of doing this. A container, when it receives an expose event, calls `gtk_container_propagate_expose()` once for each child, passing in the event the container received.

`gtk_container_propagate_expose()` takes care of deciding whether an expose event needs to be sent to the child, intersecting the event's area with the child area, and sending the event.

In most cases, a container can simply either simply inherit the ::expose implementation from GtkContainer, or, do some drawing and then chain to the ::expose implementation from GtkContainer.

container : a GtkContainer

child : a child of *container*

event : a expose event sent to container

---

gtk_container_get_focus_chain ()

```
gboolean    gtk_container_get_focus_chain   (GtkContainer *container,
                                             GList **focusable_widgets);
```

Retrieves the focus chain of the container, if one has been set explicitly. If no focus chain has been explicitly set, GTK+ computes the focus chain based on the positions of the children. In that case, GTK+ stores NULL in *focusable_widgets* and returns FALSE.

container : a GtkContainer

focusable_widgets : location to store the focus chain of the container, or NULL. You should free this list using g_list_free() when you are done with it, however no additional reference count is added to the individual widgets in the focus chain.

Returns : TRUE if the focus chain of the container has been set explicitly.

---

gtk_container_set_focus_chain ()

```
void        gtk_container_set_focus_chain   (GtkContainer *container,
                                             GList *focusable_widgets);
```

Sets a focus chain, overriding the one computed automatically by GTK+.

In principle each widget in the chain should be a descendant of the container, but this is not enforced by this method, since it's allowed to set the focus chain before you pack the widgets, or have a widget in the chain that isn't always packed. The necessary checks are done when the focus chain is actually traversed.

container : a GtkContainer.

focusable_widgets : the new focus chain.

---

gtk_container_unset_focus_chain ()

```
void        gtk_container_unset_focus_chain (GtkContainer *container);
```

Removes a focus chain explicitly set with gtk_container_set_focus_chain().

container : a GtkContainer.

---

gtk_container_class_find_child_property ()

```
GParamSpec* gtk_container_class_find_child_property
                                             (GObjectClass *cclass,
                                             const gchar *property_name);
```

Finds a child property of a container class by name.

cclass : a GtkContainerClass

property_name : the name of the child property to find

Returns : the GParamSpec of the child property or `NULL` if *`class`* has no child property with that name.

---

gtk_container_class_install_child_property ()

```
void        gtk_container_class_install_child_property
                                            (GtkContainerClass *cclass,
                                             guint property_id,
                                             GParamSpec *pspec);
```

Installs a child property on a container class.

`cclass` : a GtkContainerClass
`property_id` : the id for the property
`pspec` : the GParamSpec for the property

---

gtk_container_class_list_child_properties ()

```
GParamSpec** gtk_container_class_list_child_properties
                                            (GObjectClass *cclass,
                                             guint *n_properties);
```

Returns all child properties of a container class.

`cclass` : a GtkContainerClass
`n_properties` : location to return the number of child properties found
Returns : a newly allocated array of GParamSpec*. The array must be freed with `g_free()`.

Properties

The "border-width" property

```
  "border-width"         guint                 : Read / Write
```

The width of the empty border outside the containers children.

Allowed values: <= G_MAXINT

Default value: 0

The "child" property

```
  "child"               GtkWidget             : Write
```

Can be used to add a new child to the container.

The "resize-mode" property

```
  "resize-mode"         GtkResizeMode         : Read / Write
```

Specify how resize events are handled.

Default value: GTK_RESIZE_PARENT

Signals

The "add" signal

```
void        user_function                   (GtkContainer *container,
                                             GtkWidget *widget,
                                             gpointer user_data);
```

container : the object which received the signal.
widget :
user_data : user data set when the signal handler was connected.

The "check-resize" signal

```
void        user_function                   (GtkContainer *container,
                                             gpointer user_data);
```

container : the object which received the signal.
user_data : user data set when the signal handler was connected.

The "remove" signal

```
void        user_function                   (GtkContainer *container,
                                             GtkWidget *widget,
                                             gpointer user_data);
```

container : the object which received the signal.
widget :
user_data : user data set when the signal handler was connected.

The "set-focus-child" signal

```
void        user_function                   (GtkContainer *container,
                                             GtkWidget *widget,
                                             gpointer user_data);
```

container : the object which received the signal.
widget :
user_data : user data set when the signal handler was connected.

<< GtkButtonBox **GtkItem >>**

GtkItem

GtkItem — Abstract base class for GtkMenuItem, GtkListItem and GtkTreeItem

Synopsis

```
#include <gtk/gtk.h>

            GtkItem;
void        gtk_item_select                 (GtkItem *item);
void        gtk_item_deselect               (GtkItem *item);
void        gtk_item_toggle                 (GtkItem *item);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkWidget
               +----GtkContainer
                     +----GtkBin
                           +----GtkItem
                                 +----GtkMenuItem
                                 +----GtkListItem
                                 +----GtkTreeItem
```

Implemented Interfaces

GtkItem implements AtkImplementorIface.

Signal Prototypes

```
"deselect"  void        user_function      (GtkItem *item,
                                            gpointer user_data);
"select"    void        user_function      (GtkItem *item,
                                            gpointer user_data);
"toggle"    void        user_function      (GtkItem *item,
                                            gpointer user_data);
```

Description

The GtkItem widget is an abstract base class for GtkMenuItem, GtkListItem and GtkTreeItem.

Details

GtkItem

```
typedef struct _GtkItem GtkItem;
```

The GtkItem-struct struct contains private data only, and should be accessed using the functions below.

---

gtk_item_select ()

```
void        gtk_item_select                    (GtkItem *item);
```

Emits the "select" signal on the given item.

item : a GtkItem.

---

gtk_item_deselect ()

```
void        gtk_item_deselect                   (GtkItem *item);
```

Emits the "deselect" signal on the given item.

item : a GtkItem.

---

gtk_item_toggle ()

```
void        gtk_item_toggle                     (GtkItem *item);
```

Emits the "toggle" signal on the given item.

item : a GtkItem.

Signals

The "deselect" signal

```
void        user_function                       (GtkItem *item,
                                                 gpointer user_data);
```

Emitted when the item is deselected.

item : the object which received the signal.
user_data : user data set when the signal handler was connected.

The "select" signal

```
void        user_function                   (GtkItem *item,
                                             gpointer user_data);
```

Emitted when the item is selected.

item: the object which received the signal.
user_data: user data set when the signal handler was connected.

The "toggle" signal

```
void        user_function                   (GtkItem *item,
                                             gpointer user_data);
```

Emitted when the item is toggled.

item: the object which received the signal.
user_data: user data set when the signal handler was connected.

<< GtkContainer **GtkMisc >>**

GtkMisc

GtkMisc — Base class for widgets with alignments and padding

Synopsis

```
#include <gtk/gtk.h>

            GtkMisc;
void        gtk_misc_set_alignment          (GtkMisc *misc,
                                             gfloat xalign,
                                             gfloat yalign);
void        gtk_misc_set_padding            (GtkMisc *misc,
                                             gint xpad,
                                             gint ypad);
void        gtk_misc_get_alignment          (GtkMisc *misc,
                                             gfloat *xalign,
                                             gfloat *yalign);
void        gtk_misc_get_padding            (GtkMisc *misc,
                                             gint *xpad,
                                             gint *ypad);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkWidget
```

```
                +----GtkMisc
                      +----GtkLabel
                      +----GtkArrow
                      +----GtkImage
                      +----GtkPixmap
```

Implemented Interfaces

GtkMisc implements AtkImplementorIface.

Properties

```
  "xalign"              gfloat              : Read / Write
  "xpad"                gint                : Read / Write
  "yalign"              gfloat              : Read / Write
  "ypad"                gint                : Read / Write
```

Description

The GtkMisc widget is an abstract widget which is not useful itself, but is used to derive subclasses which have alignment and padding attributes.

The horizontal and vertical padding attributes allows extra space to be added around the widget.

The horizontal and vertical alignment attributes enable the widget to be positioned within its allocated area. Note that if the widget is added to a container in such a way that it expands automatically to fill its allocated area, the alignment settings will not alter the widgets position.

Details

GtkMisc

```
typedef struct _GtkMisc GtkMisc;
```

The GtkMisc-struct struct contains the following fields. (These fields should be considered read-only. They should never be set by an application.)

gfloat *xalign*;	the horizontal alignment, from 0 (left) to 1 (right).
gfloat *yalign*;	the vertical alignment, from 0 (top) to 1 (bottom).
guint16 *xpad*;	the amount of space to add on the left and right of the widget, in pixels.
guint16 *ypad*;	the amount of space to add on the top and bottom of the widget, in pixels.

---

gtk_misc_set_alignment ()

```
void        gtk_misc_set_alignment          (GtkMisc *misc,
                                             gfloat xalign,
                                             gfloat yalign);
```

Sets the alignment of the widget.

misc : a GtkMisc.
xalign : the horizontal alignment, from 0 (left) to 1 (right).
yalign : the vertical alignment, from 0 (top) to 1 (bottom).

---

gtk_misc_set_padding ()

```
void        gtk_misc_set_padding            (GtkMisc *misc,
                                             gint xpad,
                                             gint ypad);
```

Sets the amount of space to add around the widget.

misc : a GtkMisc.
xpad : the amount of space to add on the left and right of the widget, in pixels.

ypad : the amount of space to add on the top and bottom of the widget, in pixels.

---

gtk_misc_get_alignment ()

```
void        gtk_misc_get_alignment          (GtkMisc *misc,
                                             gfloat *xalign,
                                             gfloat *yalign);
```

Gets the X and Y alignment of the widget within its allocation. See `gtk_misc_set_alignment()`.

misc : a GtkMisc
xalign : location to store X alignment of *misc*, or `NULL`
yalign : location to store Y alignment of *misc*, or `NULL`

---

gtk_misc_get_padding ()

```
void        gtk_misc_get_padding            (GtkMisc *misc,
                                             gint *xpad,
                                             gint *ypad);
```

Gets the padding in the X and Y directions of the widget. See `gtk_misc_set_padding()`.

misc : a GtkMisc
xpad : location to store padding in the X direction, or `NULL`
ypad : location to store padding in the Y direction, or `NULL`

Properties

The "xalign" property

```
  "xalign"                gfloat                : Read / Write
```

The horizontal alignment, from 0 (left) to 1 (right). Reversed for RTL layouts.

Allowed values: [0,1]

Default value: 0.5

The "xpad" property

```
  "xpad"                  gint                  : Read / Write
```

The amount of space to add on the left and right of the widget, in pixels.

Allowed values: >= 0

Default value: 0

The "yalign" property

```
  "yalign"                gfloat                : Read / Write
```

The vertical alignment, from 0 (top) to 1 (bottom).

Allowed values: [0,1]

Default value: 0.5

The "ypad" property

```
  "ypad"                gint                : Read / Write
```

The amount of space to add on the top and bottom of the widget, in pixels.

Allowed values: >= 0

Default value: 0

<< GtkItem **GtkObject >>**

GtkObject

GtkObject — The base class of the GTK+ type hierarchy

Synopsis

```
#include <gtk/gtk.h>


            GtkObject;
#define     GTK_OBJECT_TYPE                 (object)
#define     GTK_OBJECT_TYPE_NAME            (object)
enum        GtkObjectFlags;
#define     GTK_OBJECT_FLAGS                (obj)
#define     GTK_OBJECT_FLOATING             (obj)
enum        GtkArgFlags;
GtkObject*  gtk_object_new                  (GtkType type,
                                             const gchar *first_property_name,
                                             ...);
void        gtk_object_sink                 (GtkObject *object);
GtkObject*  gtk_object_ref                  (GtkObject *object);
void        gtk_object_unref                (GtkObject *object);
void        gtk_object_weakref              (GtkObject *object,
                                             GtkDestroyNotify notify,
                                             gpointer data);
void        gtk_object_weakunref            (GtkObject *object,
                                             GtkDestroyNotify notify,
                                             gpointer data);
void        gtk_object_destroy              (GtkObject *object);
void        gtk_object_get                  (GtkObject *object,
                                             const gchar *first_property_name,
                                             ...);
void        gtk_object_set                  (GtkObject *object,
                                             const gchar *first_property_name,
                                             ...);
void        gtk_object_set_data             (GtkObject *object,
                                             const gchar *key,
                                             gpointer data);
void        gtk_object_set_data_full        (GtkObject *object,
                                             const gchar *key,
                                             gpointer data,
```

```
                                        GtkDestroyNotify destroy);
void        gtk_object_remove_data      (GtkObject *object,
                                         const gchar *key);
gpointer    gtk_object_get_data         (GtkObject *object,
                                         const gchar *key);
void        gtk_object_remove_no_notify (GtkObject *object,
                                         const gchar *key);
void        gtk_object_set_user_data    (GtkObject *object,
                                         gpointer data);
gpointer    gtk_object_get_user_data    (GtkObject *object);
void        gtk_object_add_arg_type     (const gchar *arg_name,
                                         GtkType arg_type,
                                         guint arg_flags,
                                         guint arg_id);
void        gtk_object_set_data_by_id   (GtkObject *object,
                                         GQuark data_id,
                                         gpointer data);
void        gtk_object_set_data_by_id_full  (GtkObject *object,
                                         GQuark data_id,
                                         gpointer data,
                                         GtkDestroyNotify destroy);
gpointer    gtk_object_get_data_by_id   (GtkObject *object,
                                         GQuark data_id);
void        gtk_object_remove_data_by_id    (GtkObject *object,
                                         GQuark data_id);
void        gtk_object_remove_no_notify_by_id
                                        (GtkObject *object,
                                         GQuark key_id);
#define     gtk_object_data_try_key
#define     gtk_object_data_force_id
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkWidget
         +----GtkAdjustment
         +----GtkCellRenderer
         +----GtkFileFilter
         +----GtkItemFactory
         +----GtkTooltips
         +----GtkTreeViewColumn
```

Properties

```
  "user-data"           gpointer             : Read / Write
```

Signal Prototypes

```
"destroy"   void        user_function      (GtkObject *object,
                                            gpointer user_data);
```

Description

Description

GtkObject is the base class for all widgets, and for a few non-widget objects such as GtkAdjustment. GtkObject predates GObject; non-widgets that derive from GtkObject rather than GObject do so for backward compatibility reasons.

The most interesting difference between GtkObject and GObject is the "floating" reference count. A GObject is created with a reference count of 1, owned by the creator of the GObject. (The owner of a reference is the code section that has the right to call `g_object_unref()` in order to remove that reference.) A GtkObject is created with a reference count of 1 also, but it isn't owned by anyone; calling `g_object_unref()` on the newly-created GtkObject is incorrect. Instead, the initial reference count of a GtkObject is "floating". The floating reference can be removed by anyone at any time, by calling `gtk_object_sink()`. `gtk_object_sink()` does nothing if an object is already sunk (has no floating reference).

When you add a widget to its parent container, the parent container will do this:

```
g_object_ref (G_OBJECT (child_widget));
gtk_object_sink (GTK_OBJECT (child_widget));
```

This means that the container now owns a reference to the child widget (since it called `g_object_ref()`), and the child widget has no floating reference.

The purpose of the floating reference is to keep the child widget alive until you add it to a parent container:

```
 button = gtk_button_new ();
 /* button has one floating reference to keep it alive */
 gtk_container_add (GTK_CONTAINER (container), button);
 /* button has one non-floating reference owned by the container */
```

GtkWindow is a special case, because GTK+ itself will ref/sink it on creation. That is, after calling `gtk_window_new()`, the GtkWindow will have one reference which is owned by GTK+, and no floating references.

One more factor comes into play: the "destroy" signal, emitted by the `gtk_object_destroy()` method. The "destroy" signal asks all code owning a reference to an object to release said reference. So, for example, if you call `gtk_object_destroy()` on a GtkWindow, GTK+ will release the reference count that it owns; if you call `gtk_object_destroy()` on a GtkButton, then the button will be removed from its parent container and the parent container will release its reference to the button. Because these references are released, calling `gtk_object_destroy()` should result in freeing all memory associated with an object, unless some buggy code fails to release its references in response to the "destroy" signal. Freeing memory (referred to as *finalization* only happens if the reference count reaches zero.

Some simple rules for handling GtkObject:

- Never call `g_object_unref()` unless you have previously called `g_object_ref()`, even if you created the GtkObject. (Note: this is *not* true for GObject; for GObject, the creator of the object owns a reference.)
- Call `gtk_object_destroy()` to get rid of most objects in most cases. In particular, widgets are almost always destroyed in this way.
- Because of the floating reference count, you don't need to worry about reference counting for widgets and toplevel windows, unless you explicitly call `g_object_ref()` yourself.

Details

GtkObject

```
typedef struct _GtkObject GtkObject;
```

The object itself. You should never use these members directly - use the accessing macros instead.

GTK_OBJECT_TYPE()

```
#define GTK_OBJECT_TYPE(object)          (G_TYPE_FROM_INSTANCE (object))
```

Gets the type of an object.

object : a GtkObject.

GTK_OBJECT_TYPE_NAME()

```
#define GTK_OBJECT_TYPE_NAME(object)     (g_type_name (GTK_OBJECT_TYPE (object)))
```

Gets the name of an objects type.

object : a GtkObject.

---

enum GtkObjectFlags

```
typedef enum
{
  GTK_IN_DESTRUCTION    = 1 << 0, /* Used internally during dispose */
  GTK_FLOATING          = 1 << 1,
  GTK_RESERVED_1        = 1 << 2,
  GTK_RESERVED_2        = 1 << 3
} GtkObjectFlags;
```

Tells about the state of the object.

GTK_IN_DESTRUCTION	the object is currently being destroyed. This is used internally by GTK+ to prevent reinvokations during destruction.
GTK_FLOATING	the object is orphaned. Objects that take strong hold of an object may gtk_object_sink() it, after obtaining their own references, if they believe they are nearly primary ownership of the object. GTK_CONNECTED: signals are connected to this object.
GTK_RESERVED_1	reserved for future use
GTK_RESERVED_2	reserved for future use

GTK_OBJECT_FLAGS()

```
#define GTK_OBJECT_FLAGS(obj)             (GTK_OBJECT (obj)->flags)
```

Gets the GtkObjectFlags for an object without directly accessing its members.

obj : the object whose flags are returned.

GTK_OBJECT_FLOATING()

```
#define GTK_OBJECT_FLOATING(obj)          ((GTK_OBJECT_FLAGS (obj) & GTK_FLOATING) !=
0)
```

Evaluates to TRUE if the object still has its floating reference count. See the overview documentation for GtkObject.

obj : the object to examine.

---

enum GtkArgFlags

```
typedef enum
{
  GTK_ARG_READABLE       = G_PARAM_READABLE,
  GTK_ARG_WRITABLE       = G_PARAM_WRITABLE,
  GTK_ARG_CONSTRUCT      = G_PARAM_CONSTRUCT,
  GTK_ARG_CONSTRUCT_ONLY = G_PARAM_CONSTRUCT_ONLY,
  GTK_ARG_CHILD_ARG      = 1 << 4
} GtkArgFlags;
```

Warning

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

Possible flags indicating how an argument should be treated. Deprecated in favor of GParamSpec features.

`GTK_ARG_READABLE`	the argument is readable. (i.e. can be queried)
`GTK_ARG_WRITABLE`	the argument is writable. (i.e. settable)
`GTK_ARG_CONSTRUCT`	the argument needs construction.
`GTK_ARG_CONSTRUCT_ONLY`	the argument needs construction (and will be set once during object creation), but is otherwise cannot be set. Hence this flag is not allowed with GTK_ARG_WRITABLE, and is redundant with GTK_ARG_CONSTRUCT.
`GTK_ARG_CHILD_ARG`	an argument type that applies to (and may be different for) each child. Used by GtkContainer.

---

gtk_object_new ()

```
GtkObject*  gtk_object_new                  (GtkType type,
                                             const gchar *first_property_name,
                                             ...);
```

Warning

`gtk_object_new` is deprecated and should not be used in newly-written code. Use `g_object_new()` instead.

Constructs an object given its arguments, enumerated in the call to the function.

`type` :	the type identifying this object. Returned by `gtk_type_unique()` (although for a properly-written object it should be accessible through a GTK_TYPE_FOO macro.)
`first_property_name` :	name of the first property to set when constructing the object.

`...` :	the first argument's value, followed by any number of name/argument-value pairs, terminated with `NULL`.
Returns :	the new GtkObject.

---

gtk_object_sink ()

```
void        gtk_object_sink                 (GtkObject *object);
```

Removes the floating reference from a GtkObject, if it exists; otherwise does nothing. See the GtkObject overview documentation at the top of the page.

`object` : the object to sink.

---

gtk_object_ref ()

```
GtkObject*  gtk_object_ref                  (GtkObject *object);
```

Warning

`gtk_object_ref` is deprecated and should not be used in newly-written code. Use `g_object_ref()` instead.

Increases the reference count of the object.

`object` : the object to reference.
Returns : *`object`*.

---

gtk_object_unref ()

```
void        gtk_object_unref                (GtkObject *object);
```

Warning

`gtk_object_unref` is deprecated and should not be used in newly-written code. Use `g_object_unref()` instead.

Decreases the reference count of an object. When its reference count drops to 0, the object is finalized (i.e. its memory is freed).

`object` : the object to dereference.

gtk_object_weakref ()

```
void        gtk_object_weakref              (GtkObject *object,
                                             GtkDestroyNotify notify,
                                             gpointer data);
```

Warning

`gtk_object_weakref` is deprecated and should not be used in newly-written code. Use `g_object_weak_ref()` instead.

Adds a weak reference callback to an object. Weak references are used for notification when an object is finalized. They are called "weak references" because they allow you to safely hold a pointer to an object without calling `g_object_ref()` (`g_object_ref()` adds a strong reference, that is, forces the object to stay alive).

`object` : object to weakly reference.
`notify` : callback to invoke before the object is freed.
`data` : extra data to pass to notify.

---

gtk_object_weakunref ()

```
void        gtk_object_weakunref            (GtkObject *object,
                                             GtkDestroyNotify notify,
                                             gpointer data);
```

Warning

`gtk_object_weakunref` is deprecated and should not be used in newly-written code. Use `g_object_weak_unref()` instead.

Removes a weak reference callback to an object.

`object` : object stop weakly referencing.
`notify` : callback to search for.
`data` : data to search for.

---

gtk_object_destroy ()

```
void        gtk_object_destroy              (GtkObject *object);
```

Emits the "destroy" signal notifying all reference holders that they should release the GtkObject. See the overview documentation at the top of the page for more details.

The memory for the object itself won't be deleted until its reference count actually drops to 0; `gtk_object_destroy()` merely asks reference holders to release their references, it does not free the object.

`object` : the object to destroy.

---

gtk_object_get ()

```
void        gtk_object_get                  (GtkObject *object,
                                             const gchar *first_property_name,
                                             ...);
```

Warning

`gtk_object_get` is deprecated and should not be used in newly-written code. Use `g_object_get()` instead.

Gets properties of an object.

`object` : a GtkObject.
`first_property_name` : name of first property to get the value for.
`...` : `NULL`-terminated list of name-return location pairs.

---

gtk_object_set ()

```
void        gtk_object_set                  (GtkObject *object,
                                             const gchar *first_property_name,
                                             ...);
```

Warning

`gtk_object_set` is deprecated and should not be used in newly-written code. Use `g_object_set()` instead.

Sets properties on an object.

```
void set_box_properties (GtkBox* box)
{
  gtk_object_set (GTK_OBJECT (box), "homogeneous", TRUE,
                                    "spacing", 8,
```

```
                                        NULL);
}
```

object :	a GtkObject.
first_property_name :	name of the first property to set
... :	the value of the first argument, followed optionally by more name/value pairs, followed by NULL.

---

gtk_object_set_data ()

```
void        gtk_object_set_data             (GtkObject *object,
                                             const gchar *key,
                                             gpointer data);
```

Warning

gtk_object_set_data is deprecated and should not be used in newly-written code. Use g_object_set_data() instead.

Each object carries around a table of associations from strings to pointers. This function lets you set an association.

If the object already had an association with that name, the old association will be destroyed.

object :	object containing the associations.
key :	name of the key.
data :	data to associate with that key.

---

gtk_object_set_data_full ()

```
void        gtk_object_set_data_full        (GtkObject *object,
                                             const gchar *key,
                                             gpointer data,
                                             GtkDestroyNotify destroy);
```

Warning

gtk_object_set_data_full is deprecated and should not be used in newly-written code. Use g_object_set_data_full() instead.

Like gtk_object_set_data() except it adds notification for when the association is destroyed, either by gtk_object_remove_data() or when the object is destroyed.

object : object containing the associations.
key : name of the key.
data : data to associate with that key.
destroy : function to call when the association is destroyed.

---

gtk_object_remove_data ()

```
void        gtk_object_remove_data          (GtkObject *object,
                                             const gchar *key);
```

Warning

`gtk_object_remove_data` is deprecated and should not be used in newly-written code. Use `g_object_set_data()` to set the object data to `NULL` instead.

Removes a specified datum from the object's data associations (the object_data). Subsequent calls to `gtk_object_get_data()` will return `NULL`.

If you specified a destroy handler with `gtk_object_set_data_full()`, it will be invoked.

object : the object maintaining the association.
key : name of the key for that association.

---

gtk_object_get_data ()

```
gpointer    gtk_object_get_data             (GtkObject *object,
                                             const gchar *key);
```

Warning

`gtk_object_get_data` is deprecated and should not be used in newly-written code. Use `g_object_get_data()` instead.

Get a named field from the object's table of associations (the object_data).

object : the object maintaining the associations.
key : name of the key for that association.
Returns : the data if found, or `NULL` if no such data exists.

---

gtk_object_remove_no_notify ()

```
void        gtk_object_remove_no_notify     (GtkObject *object,
                                             const gchar *key);
```

Warning

`gtk_object_remove_no_notify` is deprecated and should not be used in newly-written code. Use `g_object_steal_data()` instead.

Remove a specified datum from the object's data associations (the object_data), without invoking the association's destroy handler.

Just like `gtk_object_remove_data()` except that any destroy handler will be ignored. Therefore this only affects data set using `gtk_object_set_data_full()`.

object : the object maintaining the association.
key : name of the key for that association.

---

gtk_object_set_user_data ()

```
void        gtk_object_set_user_data        (GtkObject *object,
                                             gpointer data);
```

Warning

`gtk_object_set_user_data` is deprecated and should not be used in newly-written code. Use `g_object_set_data()` instead.

For convenience, every object offers a generic user data pointer. This function sets it.

object : the object whose user data should be set.
data : the new value for the user data.

---

gtk_object_get_user_data ()

```
gpointer    gtk_object_get_user_data        (GtkObject *object);
```

Warning

`gtk_object_get_user_data` is deprecated and should not be used in newly-written code. Use `g_object_get_data()` instead.

Get the object's user data pointer.

This is intended to be a pointer for your convenience in writing applications.

`object` : the object.
Returns : the user data field for object.

---

gtk_object_add_arg_type ()

```
void        gtk_object_add_arg_type         (const gchar *arg_name,
                                             GtkType arg_type,
                                             guint arg_flags,
                                             guint arg_id);
```

Warning

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

Deprecated in favor of the GObject property system including GParamSpec. Add a new type of argument to an object class. Usually this is called when registering a new type of object.

`arg_name` : fully qualify object name, for example GtkObject::user_data.
`arg_type` : type of the argument.
`arg_flags` : bitwise-OR of the GtkArgFlags enum. (Whether the argument is settable or gettable, whether it is set when the object is constructed.)
`arg_id` : an internal number, passed in from here to the "set_arg" and "get_arg" handlers of the object.

---

gtk_object_set_data_by_id ()

```
void        gtk_object_set_data_by_id       (GtkObject *object,
                                             GQuark data_id,
                                             gpointer data);
```

Warning

`gtk_object_set_data_by_id` is deprecated and should not be used in newly-written code. Use `g_object_set_qdata()` instead.

Just like gtk_object_set_data() except that it takes a GQuark instead of a string, so it is slightly faster.

Use gtk_object_data_try_key() and gtk_object_data_force_id() to get an id from a string.

object : object containing the associations.
data_id : quark of the key.
data : data to associate with that key.

---

gtk_object_set_data_by_id_full ()

```
void        gtk_object_set_data_by_id_full  (GtkObject *object,
                                             GQuark data_id,
                                             gpointer data,
                                             GtkDestroyNotify destroy);
```

Warning

gtk_object_set_data_by_id_full is deprecated and should not be used in newly-written code. Use g_object_set_qdata_full() instead.

Just like gtk_object_set_data_full() except that it takes a GQuark instead of a string, so it is slightly faster.

Use gtk_object_data_try_key() and gtk_object_data_force_id() to get an id from a string.

object : object containing the associations.
data_id : quark of the key.
data : data to associate with that key.
destroy : function to call when the association is destroyed.

---

gtk_object_get_data_by_id ()

```
gpointer    gtk_object_get_data_by_id       (GtkObject *object,
                                             GQuark data_id);
```

Warning

gtk_object_get_data_by_id is deprecated and should not be used in newly-written code. Use g_object_get_qdata() instead.

Just like gtk_object_get_data() except that it takes a GQuark instead of a string, so it is slightly faster.

Use gtk_object_data_try_key() and gtk_object_data_force_id() to get an id from a string.

object : object containing the associations.
data_id : quark of the key.
Returns : the data if found, or NULL if no such data exists.

---

gtk_object_remove_data_by_id ()

```
void        gtk_object_remove_data_by_id    (GtkObject *object,
                                             GQuark data_id);
```

Warning

gtk_object_remove_data_by_id is deprecated and should not be used in newly-written code. Use g_object_set_qdata() with data of NULL instead.

Just like gtk_object_remove_data() except that it takes a GQuark instead of a string, so it is slightly faster.

Remove a specified datum from the object's data associations. Subsequent calls to gtk_object_get_data() will return NULL.

Use gtk_object_data_try_key() and gtk_object_data_force_id() to get an id from a string.

object : object containing the associations.
data_id : quark of the key.

---

gtk_object_remove_no_notify_by_id ()

```
void        gtk_object_remove_no_notify_by_id
                                            (GtkObject *object,
                                             GQuark key_id);
```

Warning

gtk_object_remove_no_notify_by_id is deprecated and should not be used in newly-written code. Use g_object_steal_qdata() instead.

Just like gtk_object_remove_no_notify() except that it takes a GQuark instead of a string, so it is slightly faster.

Use gtk_object_data_try_key() and gtk_object_data_force_id() to get an id from a string.

object : object containing the associations.
key_id : quark of the key.

---

gtk_object_data_try_key

```
#define gtk_object_data_try_key     g_quark_try_string
```

Warning

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

Useless deprecated macro. Ignore it.

---

gtk_object_data_force_id

```
#define gtk_object_data_force_id    g_quark_from_string
```

Warning

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

Useless deprecated macro. Ignore it.

Properties

The "user-data" property

```
  "user-data"           gpointer              : Read / Write
```

Anonymous User Data Pointer.

Signals

The "destroy" signal

```
void        user_function                 (GtkObject *object,
                                           gpointer user_data);
```

Signals that all holders of a reference to the GtkObject should release the reference that they hold. May result in finalization of the object if all references are released.

object : the object which received the signal.
user_data : user data set when the signal handler was connected.

See Also

GObject

<< GtkMisc **GtkPaned >>**

GtkPaned

GtkPaned — Base class for widgets with two adjustable panes

Synopsis

```
#include <gtk/gtk.h>

            GtkPaned;
void        gtk_paned_add1                  (GtkPaned *paned,
                                             GtkWidget *child);
void        gtk_paned_add2                  (GtkPaned *paned,
                                             GtkWidget *child);
#define     gtk_paned_gutter_size           (p,s)
void        gtk_paned_pack1                 (GtkPaned *paned,
                                             GtkWidget *child,
                                             gboolean resize,
                                             gboolean shrink);
void        gtk_paned_pack2                 (GtkPaned *paned,
                                             GtkWidget *child,
                                             gboolean resize,
                                             gboolean shrink);
GtkWidget*  gtk_paned_get_child1            (GtkPaned *paned);
GtkWidget*  gtk_paned_get_child2            (GtkPaned *paned);
#define     gtk_paned_set_gutter_size       (p,s)
void        gtk_paned_set_position          (GtkPaned *paned,
                                             gint position);
gint        gtk_paned_get_position          (GtkPaned *paned);
```

Object Hierarchy

```
  GObject
   +----GtkObject
```

```
          +----GtkWidget
                +----GtkContainer
                      +----GtkPaned
                            +----GtkHPaned
                            +----GtkVPaned
```

Implemented Interfaces

GtkPaned implements AtkImplementorIface.

Properties

```
  "max-position"          gint                : Read
  "min-position"          gint                : Read
  "position"              gint                : Read / Write
  "position-set"          gboolean            : Read / Write
```

Child Properties

```
  "resize"                gboolean            : Read / Write
  "shrink"                gboolean            : Read / Write
```

Style Properties

```
  "handle-size"           gint                : Read
```

Signal Prototypes

```
"accept-position"
            gboolean    user_function      (GtkPaned *paned,
                                            gpointer user_data);
"cancel-position"
            gboolean    user_function      (GtkPaned *paned,
                                            gpointer user_data);
```

```
"cycle-child-focus"
            gboolean    user_function      (GtkPaned *paned,
                                            gboolean arg1,
                                            gpointer user_data);
"cycle-handle-focus"
            gboolean    user_function      (GtkPaned *paned,
                                            gboolean arg1,
                                            gpointer user_data);
"move-handle"
            gboolean    user_function      (GtkPaned *paned,
                                            GtkScrollType arg1,
                                            gpointer user_data);
"toggle-handle-focus"
            gboolean    user_function      (GtkPaned *paned,
                                            gpointer user_data);
```

Description

GtkPaned is the base class for widgets with two panes, arranged either horizontally (GtkHPaned) or vertically (GtkVPaned). Child widgets are added to the panes of the widget with `gtk_paned_pack1()` and `gtk_paned_pack2()`. The division beween the two children is set by default from the size requests of the children, but it can be adjusted by the user.

A paned widget draws a separator between the two child widgets and a small handle that the user can drag to adjust the division. It does not draw any relief around the children or around the separator. (The space in which the separator is called the gutter.) Often, it is useful to put each child inside a GtkFrame with the shadow type set to `GTK_SHADOW_IN` so that the gutter appears as a ridge.

Each child has two options that can be set, *`resize`* and *`shrink`*. If *`resize`* is true, then when the GtkPaned is resized, that child will expand or shrink along with the paned widget. If *`shrink`* is true, then when that child can be made smaller than its requisition by the user. Setting *`shrink`* to `FALSE` allows the application to set a minimum size. If *`resize`* is false for both children, then this is treated as if *`resize`* is true for both children.

The application can set the position of the slider as if it were set by the user, by calling `gtk_paned_set_position()`.

Example 1. Creating a paned widget with minimum sizes.

```
GtkWidget *hpaned = gtk_hpaned_new ();
GtkWidget *frame1 = gtk_frame_new (NULL);
GtkWidget *frame2 = gtk_frame_new (NULL);
gtk_frame_set_shadow_type (GTK_FRAME (frame1), GTK_SHADOW_IN);
gtk_frame_set_shadow_type (GTK_FRAME (frame2), GTK_SHADOW_IN);
```

```
gtk_widget_set_size_request (hpaned, 200 + GTK_PANED (hpaned)->gutter_size, -1);

gtk_paned_pack1 (GTK_PANED (hpaned), frame1, TRUE, FALSE);
gtk_widget_set_size_request (frame1, 50, -1);

gtk_paned_pack2 (GTK_PANED (hpaned), frame2, FALSE, FALSE);
gtk_widget_set_size_request (frame2, 50, -1);
```

Details

GtkPaned

```
typedef struct _GtkPaned GtkPaned;
```

---

gtk_paned_add1 ()

```
void        gtk_paned_add1                  (GtkPaned *paned,
                                             GtkWidget *child);
```

Adds a child to the top or left pane with default parameters. This is equivalent to `gtk_paned_pack1 (paned, child, FALSE, TRUE)`.

paned : a paned widget
child : the child to add

---

gtk_paned_add2 ()

```
void        gtk_paned_add2                  (GtkPaned *paned,
                                             GtkWidget *child);
```

Adds a child to the bottom or right pane with default parameters. This is equivalent to `gtk_paned_pack2 (paned, child, TRUE, TRUE)`.

paned : a paned widget
child : the child to add

---

gtk_paned_gutter_size()

```
#define gtk_paned_gutter_size(p,s)              (void) 0
```

Warning

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

Old name for `gtk_paned_set_gutter_size()`.

p :
s :

---

gtk_paned_pack1 ()

```
void        gtk_paned_pack1                 (GtkPaned *paned,
                                             GtkWidget *child,
                                             gboolean resize,
                                             gboolean shrink);
```

Adds a child to the top or left pane.

paned : a paned widget
child : the child to add
resize : should this child expand when the paned widget is resized.
shrink : can this child be made smaller than its requsition.

---

gtk_paned_pack2 ()

```
void        gtk_paned_pack2                 (GtkPaned *paned,
                                             GtkWidget *child,
                                             gboolean resize,
                                             gboolean shrink);
```

Adds a child to the bottom or right pane.

paned : a paned widget

child : the child to add
resize : should this child expand when the paned widget is resized.
shrink : can this child be made smaller than its requsition.

---

gtk_paned_get_child1 ()

```
GtkWidget*  gtk_paned_get_child1            (GtkPaned *paned);
```

Obtains the first child of the paned widget.

paned : a GtkPaned widget
Returns : first child, or NULL if it is not set.

Since 2.4

---

gtk_paned_get_child2 ()

```
GtkWidget*  gtk_paned_get_child2            (GtkPaned *paned);
```

Obtains the second child of the paned widget.

paned : a GtkPaned widget
Returns : second child, or NULL if it is not set.

Since 2.4

---

gtk_paned_set_gutter_size()

```
#define gtk_paned_set_gutter_size(p,s)          (void) 0
```

Warning

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

In older versions of GTK+, this function used to set the width of the gutter (the area between the two panes). It does nothing now.

p : a paned widget
s : the width of the gutter in pixels

---

gtk_paned_set_position ()

```
void        gtk_paned_set_position          (GtkPaned *paned,
                                             gint position);
```

Sets the position of the divider between the two panes.

paned : a GtkPaned widget
position : pixel position of divider, a negative value means that the position is unset.

---

gtk_paned_get_position ()

```
gint        gtk_paned_get_position          (GtkPaned *paned);
```

Obtains the position of the divider between the two panes.

paned : a GtkPaned widget
Returns : position of the divider

Properties

The "max-position" property

```
  "max-position"         gint                  : Read
```

The largest possible value for the position property. This property is derived from the size and shrinkability of the widget's children.

Allowed values: >= 0

Default value: 2147483647

Since 2.4

The "min-position" property

```
"min-position"          gint                : Read
```

The smallest possible value for the position property. This property is derived from the size and shrinkability of the widget's children.

Allowed values: >= 0

Default value: 0

Since 2.4

The "position" property

```
"position"              gint                : Read / Write
```

Position of paned separator in pixels (0 means all the way to the left/top).

Allowed values: >= 0

Default value: 0

The "position-set" property

```
"position-set"          gboolean            : Read / Write
```

TRUE if the Position property should be used.

Default value: FALSE

Child Properties

The "resize" child property

```
  "resize"              gboolean              : Read / Write
```

The "resize" child property determines whether the child expands and shrinks along with the paned widget.

Default value: TRUE

Since 2.4

The "shrink" child property

```
  "shrink"              gboolean              : Read / Write
```

The "shrink" child property determines whether the child can be made smaller than its requisition.

Default value: TRUE

Since 2.4

Style Properties

The "handle-size" style property

```
  "handle-size"         gint                  : Read
```

Width of handle.

Allowed values: >= 0

Default value: 5

Signals

The "accept-position" signal

```
gboolean    user_function                   (GtkPaned *paned,
                                             gpointer user_data);
```

paned : the object which received the signal.
user_data : user data set when the signal handler was connected.
Returns :

The "cancel-position" signal

```
gboolean    user_function                   (GtkPaned *paned,
                                             gpointer user_data);
```

paned : the object which received the signal.
user_data : user data set when the signal handler was connected.
Returns :

The "cycle-child-focus" signal

```
gboolean    user_function                   (GtkPaned *paned,
                                             gboolean arg1,
                                             gpointer user_data);
```

paned : the object which received the signal.
arg1 :
user_data : user data set when the signal handler was connected.
Returns :

The "cycle-handle-focus" signal

```
gboolean    user_function                   (GtkPaned *paned,
                                             gboolean arg1,
                                             gpointer user_data);
```

paned : the object which received the signal.
arg1 :
user_data : user data set when the signal handler was connected.
Returns :

The "move-handle" signal

```
gboolean    user_function                   (GtkPaned *paned,
                                             GtkScrollType arg1,
                                             gpointer user_data);
```

paned : the object which received the signal.
arg1 :
user_data : user data set when the signal handler was connected.
Returns :

The "toggle-handle-focus" signal

```
gboolean    user_function                   (GtkPaned *paned,
                                             gpointer user_data);
```

paned : the object which received the signal.
user_data : user data set when the signal handler was connected.
Returns :

<< GtkObject **GtkRange >>**

GtkRange

GtkRange — Base class for widgets which visualize an adjustment

Synopsis

```
#include <gtk/gtk.h>

            GtkRange;
GtkAdjustment* gtk_range_get_adjustment     (GtkRange *range);
void        gtk_range_set_update_policy     (GtkRange *range,
                                             GtkUpdateType policy);
void        gtk_range_set_adjustment        (GtkRange *range,
                                             GtkAdjustment *adjustment);
gboolean    gtk_range_get_inverted          (GtkRange *range);
void        gtk_range_set_inverted          (GtkRange *range,
                                             gboolean setting);
GtkUpdateType gtk_range_get_update_policy   (GtkRange *range);
gdouble     gtk_range_get_value             (GtkRange *range);
void        gtk_range_set_increments        (GtkRange *range,
                                             gdouble step,
                                             gdouble page);
void        gtk_range_set_range             (GtkRange *range,
                                             gdouble min,
                                             gdouble max);
void        gtk_range_set_value             (GtkRange *range,
                                             gdouble value);
```

Object Hierarchy

```
GObject
 +----GtkObject
       +----GtkWidget
             +----GtkRange
                   +----GtkScale
                   +----GtkScrollbar
```

Implemented Interfaces

GtkRange implements AtkImplementorIface.

Properties

```
"adjustment"           GtkAdjustment         : Read / Write / Construct
"inverted"             gboolean              : Read / Write
"update-policy"        GtkUpdateType         : Read / Write
```

Style Properties

```
"arrow-displacement-x" gint                  : Read
"arrow-displacement-y" gint                  : Read
"slider-width"         gint                  : Read
"stepper-size"         gint                  : Read
"stepper-spacing"      gint                  : Read
"trough-border"        gint                  : Read
```

Signal Prototypes

```
"adjust-bounds"
            void        user_function      (GtkRange *range,
                                            gdouble arg1,
                                            gpointer user_data);
```

```
"change-value"
            gboolean    user_function      (GtkRange *range,
                                            GtkScrollType scroll,
                                            gdouble value,
                                            gpointer user_data);
"move-slider"
            void        user_function      (GtkRange *range,
                                            GtkScrollType arg1,
                                            gpointer user_data);
"value-changed"
            void        user_function      (GtkRange *range,
                                            gpointer user_data);
```

Description

Details

GtkRange

```
typedef struct _GtkRange GtkRange;
```

---

gtk_range_get_adjustment ()

```
GtkAdjustment* gtk_range_get_adjustment     (GtkRange *range);
```

Get the GtkAdjustment which is the "model" object for GtkRange. See `gtk_range_set_adjustment()` for details. The return value does not have a reference added, so should not be unreferenced.

range : a GtkRange

Returns : a GtkAdjustment

---

gtk_range_set_update_policy ()

```
void        gtk_range_set_update_policy     (GtkRange *range,
                                             GtkUpdateType policy);
```

Sets the update policy for the range. GTK_UPDATE_CONTINUOUS means that anytime the range slider is moved, the range value will change and the value_changed signal will be emitted. GTK_UPDATE_DELAYED means that the value will be updated after a brief timeout where no slider motion occurs, so updates are spaced by a short time rather than continuous. GTK_UPDATE_DISCONTINUOUS means that the value will only be updated when the user releases the button and ends the slider drag operation.

range : a GtkRange
policy : update policy

---

gtk_range_set_adjustment ()

```
void        gtk_range_set_adjustment        (GtkRange *range,
                                             GtkAdjustment *adjustment);
```

Sets the adjustment to be used as the "model" object for this range widget. The adjustment indicates the current range value, the minimum and maximum range values, the step/page increments used for keybindings and scrolling, and the page size. The page size is normally 0 for GtkScale and nonzero for GtkScrollbar, and indicates the size of the visible area of the widget being scrolled. The page size affects the size of the scrollbar slider.

range : a GtkRange
adjustment : a GtkAdjustment

---

gtk_range_get_inverted ()

```
gboolean    gtk_range_get_inverted          (GtkRange *range);
```

Gets the value set by `gtk_range_set_inverted()`.

range : a GtkRange

Returns : TRUE if the range is inverted

---

gtk_range_set_inverted ()

```
void        gtk_range_set_inverted          (GtkRange *range,
                                             gboolean setting);
```

Ranges normally move from lower to higher values as the slider moves from top to bottom or left to right. Inverted ranges have higher values at the top or on the right rather than on the bottom or left.

range : a GtkRange

setting : TRUE to invert the range

---

gtk_range_get_update_policy ()

```
GtkUpdateType gtk_range_get_update_policy   (GtkRange *range);
```

Gets the update policy of *range*. See gtk_range_set_update_policy().

range : a GtkRange

Returns : the current update policy

---

gtk_range_get_value ()

```
gdouble     gtk_range_get_value             (GtkRange *range);
```

Gets the current value of the range.

range : a GtkRange

Returns : current value of the range.

---

gtk_range_set_increments ()

```
void        gtk_range_set_increments        (GtkRange *range,
                                             gdouble step,
                                             gdouble page);
```

Sets the step and page sizes for the range. The step size is used when the user clicks the GtkScrollbar arrows or moves GtkScale via arrow keys. The page size is used for example when moving via Page Up or Page Down keys.

range : a GtkRange
step : step size
page : page size

---

gtk_range_set_range ()

```
void        gtk_range_set_range             (GtkRange *range,
                                             gdouble min,
                                             gdouble max);
```

Sets the allowable values in the GtkRange, and clamps the range value to be between *min* and *max*. (If the range has a non-zero page size, it is clamped between *min* and *max* - page-size.)

range : a GtkRange
min : minimum range value
max : maximum range value

---

gtk_range_set_value ()

```
void        gtk_range_set_value             (GtkRange *range,
                                             gdouble value);
```

Sets the current value of the range; if the value is outside the minimum or maximum range values, it will be

clamped to fit inside them. The range emits the "value_changed" signal if the value changes.

range : a GtkRange
value : new value of the range

Properties

The "adjustment" property

```
"adjustment"          GtkAdjustment         : Read / Write / Construct
```

The GtkAdjustment that contains the current value of this range object.

The "inverted" property

```
"inverted"            gboolean              : Read / Write
```

Invert direction slider moves to increase range value.

Default value: FALSE

The "update-policy" property

```
"update-policy"       GtkUpdateType         : Read / Write
```

How the range should be updated on the screen.

Default value: GTK_UPDATE_CONTINUOUS

Style Properties

The "arrow-displacement-x" style property

```
"arrow-displacement-x" gint                : Read
```

How far in the x direction to move the arrow when the button is depressed.

Default value: 0

The "arrow-displacement-y" style property

```
"arrow-displacement-y" gint                : Read
```

How far in the y direction to move the arrow when the button is depressed.

Default value: 0

The "slider-width" style property

```
"slider-width"         gint                : Read
```

Width of scrollbar or scale thumb.

Allowed values: >= 0

Default value: 14

The "stepper-size" style property

```
"stepper-size"         gint                : Read
```

Length of step buttons at ends.

Allowed values: >= 0

Default value: 14

The "stepper-spacing" style property

```
  "stepper-spacing"      gint                  : Read
```

Spacing between step buttons and thumb.

Allowed values: >= 0

Default value: 0

The "trough-border" style property

```
  "trough-border"        gint                  : Read
```

Spacing between thumb/steppers and outer trough bevel.

Allowed values: >= 0

Default value: 1

Signals

The "adjust-bounds" signal

```
void        user_function                      (GtkRange *range,
                                                gdouble arg1,
                                                gpointer user_data);
```

range : the object which received the signal.
arg1 :
user_data : user data set when the signal handler was connected.

The "change-value" signal

```
gboolean    user_function                   (GtkRange *range,
                                             GtkScrollType scroll,
                                             gdouble value,
                                             gpointer user_data);
```

The ::change-value signal is emitted when a scroll action is performed on a range. It allows an application to determine the type of scroll event that occurred and the resultant new value. The application can handle the event itself and return TRUE to prevent further processing. Or, by returning FALSE, it can pass the event to other handlers until the default GTK+ handler is reached.

The value parameter is unrounded. An application that overrides the ::change-value signal is responsible for clamping the value to the desired number of digits; the default GTK+ handler clamps the value based on *range->round_digits*.

It is not possible to use delayed update policies in an overridden ::change-value handler.

range : the range that received the signal.
scroll : the type of scroll action that was performed.
value : the new value resulting from the scroll action.
returns : TRUE to prevent other handlers from being invoked for the signal. FALSE to propagate the signal further.
user_data : user data set when the signal handler was connected.

Since 2.6

The "move-slider" signal

```
void        user_function                   (GtkRange *range,
```

```
                                                GtkScrollType arg1,
                                                gpointer user_data);
```

Virtual function that moves the slider. Used for keybindings.

range : the GtkRange
arg1 :
user_data : user data set when the signal handler was connected.

The "value-changed" signal

```
void        user_function                   (GtkRange *range,
                                             gpointer user_data);
```

Emitted when the range value changes.

range : the GtkRange
user_data : user data set when the signal handler was connected.

<< GtkPaned **GtkScale >>**

GtkScale

GtkScale — Base class for GtkHScale and GtkVScale

Synopsis

```
#include <gtk/gtk.h>

            GtkScale;
void        gtk_scale_set_digits            (GtkScale *scale,
                                             gint digits);
void        gtk_scale_set_draw_value        (GtkScale *scale,
                                             gboolean draw_value);
void        gtk_scale_set_value_pos         (GtkScale *scale,
                                             GtkPositionType pos);
gint        gtk_scale_get_digits            (GtkScale *scale);
gboolean    gtk_scale_get_draw_value        (GtkScale *scale);
GtkPositionType gtk_scale_get_value_pos     (GtkScale *scale);
PangoLayout* gtk_scale_get_layout           (GtkScale *scale);
void        gtk_scale_get_layout_offsets    (GtkScale *scale,
                                             gint *x,
                                             gint *y);
```

Object Hierarchy

```
  GObject
   +----GtkObject
```

```
          +----GtkWidget
                +----GtkRange
                      +----GtkScale
                            +----GtkHScale
                            +----GtkVScale
```

Implemented Interfaces

GtkScale implements AtkImplementorIface.

Properties

```
  "digits"              gint                    : Read / Write
  "draw-value"          gboolean                : Read / Write
  "value-pos"           GtkPositionType         : Read / Write
```

Style Properties

```
  "slider-length"       gint                    : Read
  "value-spacing"       gint                    : Read
```

Signal Prototypes

```
"format-value"
            gchar*      user_function      (GtkScale *scale,
                                            gdouble arg1,
                                            gpointer user_data);
```

Description

A GtkScale is a slider control used to select a numeric value. To use it, you'll probably want to investigate the methods on its base class, GtkRange, in addition to the methods for GtkScale itself. To set the value of a scale, you would normally use `gtk_range_set_value()`. To detect changes to the value, you would normally use the "value_changed" signal.

The GtkScale widget is an abstract class, used only for deriving the subclasses GtkHScale and GtkVScale. To create a scale widget, call `gtk_hscale_new_with_range()` or `gtk_vscale_new_with_range()`.

Details

GtkScale

```
typedef struct _GtkScale GtkScale;
```

The GtkScale-struct struct contains the following fields. (These fields should be considered read-only. They should never be set by an application.)

guint *draw_value*; non-zero if the scale's current value is displayed next to the slider.

guint *value_pos*; the position in which the textual value is displayed, selected from GtkPositionType.

---

gtk_scale_set_digits ()

```
void        gtk_scale_set_digits            (GtkScale *scale,
                                             gint digits);
```

Sets the number of decimal places that are displayed in the value. Also causes the value of the adjustment to be rounded off to this number of digits, so the retrieved value matches the value the user saw.

scale: a GtkScale.

digits: the number of decimal places to display, e.g. use 1 to display 1.0, 2 to display 1.00 etc.

gtk_scale_set_draw_value ()

```
void        gtk_scale_set_draw_value        (GtkScale *scale,
                                             gboolean draw_value);
```

Specifies whether the current value is displayed as a string next to the slider.

scale : a GtkScale.

draw_value : a boolean.

gtk_scale_set_value_pos ()

```
void        gtk_scale_set_value_pos         (GtkScale *scale,
                                             GtkPositionType pos);
```

Sets the position in which the current value is displayed.

scale : a GtkScale.

pos : the position in which the current value is displayed.

gtk_scale_get_digits ()

```
gint        gtk_scale_get_digits            (GtkScale *scale);
```

Gets the number of decimal places that are displayed in the value.

scale : a GtkScale.

Returns : the number of decimal places that are displayed.

gtk_scale_get_draw_value ()

```
gboolean    gtk_scale_get_draw_value        (GtkScale *scale);
```

Returns whether the current value is displayed as a string next to the slider.

scale : a GtkScale.

Returns : whether the current value is displayed as a string.

---

gtk_scale_get_value_pos ()

```
GtkPositionType gtk_scale_get_value_pos     (GtkScale *scale);
```

Gets the position in which the current value is displayed.

scale : a GtkScale.

Returns : the position in which the current value is displayed.

---

gtk_scale_get_layout ()

```
PangoLayout* gtk_scale_get_layout           (GtkScale *scale);
```

Gets the PangoLayout used to display the scale. The returned object is owned by the scale so does not need to be freed by the caller.

scale : A GtkScale

Returns : the PangoLayout for this scale, or NULL if the draw_value property is FALSE.

Since 2.4

gtk_scale_get_layout_offsets ()

```
void        gtk_scale_get_layout_offsets    (GtkScale *scale,
                                             gint *x,
                                             gint *y);
```

Obtains the coordinates where the scale will draw the PangoLayout representing the text in the scale. Remember when using the PangoLayout function you need to convert to and from pixels using PANGO_PIXELS() or PANGO_SCALE.

If the draw_value property is FALSE, the return values are undefined.

scale : a GtkScale
x : location to store X offset of layout, or NULL
y : location to store Y offset of layout, or NULL

Since 2.4

Properties

The "digits" property

```
  "digits"               gint                  : Read / Write
```

The number of decimal places that are displayed in the value.

Allowed values: [-1,64]

Default value: 1

The "draw-value" property

```
"draw-value"          gboolean              : Read / Write
```

Whether the current value is displayed as a string next to the slider.

Default value: FALSE

The "value-pos" property

```
"value-pos"           GtkPositionType       : Read / Write
```

The position in which the current value is displayed.

Default value: GTK_POS_LEFT

Style Properties

The "slider-length" style property

```
"slider-length"       gint                  : Read
```

Length of scale's slider.

Allowed values: >= 0

Default value: 31

The "value-spacing" style property

```
  "value-spacing"        gint                  : Read
```

Space between value text and the slider/trough area.

Allowed values: >= 0

Default value: 2

Signals

The "format-value" signal

```
gchar*      user_function                   (GtkScale *scale,
                                             gdouble arg1,
                                             gpointer user_data);
```

Signal which allows you to change how the scale value is displayed. Connect a signal handler which returns an allocated string representing *value*. That string will then be used to display the scale's value. Here's an example signal handler which displays a value 1.0 as with "-->1.0<--".

```
static gchar*
format_value_callback (GtkScale *scale,
                       gdouble   value)
{
  return g_strdup_printf ("-->%0.*g<--",
                           gtk_scale_get_digits (scale), value);
}
```

scale : the object which received the signal.

arg1 :

user_data : user data set when the signal handler was connected.

Returns : allocated string representing *value*

<< GtkRange **GtkScrollbar >>**

GtkScrollbar

GtkScrollbar — Base class for GtkHScrollbar and GtkVScrollbar

Synopsis

```
#include <gtk/gtk.h>

            GtkScrollbar;
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkWidget
               +----GtkRange
                     +----GtkScrollbar
                           +----GtkHScrollbar
                           +----GtkVScrollbar
```

Implemented Interfaces

GtkScrollbar implements AtkImplementorIface.

Style Properties

```
  "fixed-slider-length"  gboolean              : Read
  "has-backward-stepper" gboolean              : Read
  "has-forward-stepper"  gboolean              : Read
  "has-secondary-backward-stepper" gboolean              : Read
  "has-secondary-forward-stepper" gboolean             : Read
  "min-slider-length"    gint                  : Read
```

Description

The GtkScrollbar widget is an abstract base class for GtkHScrollbar and GtkVScrollbar. It is not very useful in itself.

The position of the thumb in a scrollbar is controlled by the scroll adjustments. See GtkAdjustment for the fields in an adjustment - for GtkScrollbar, the "value" field represents the position of the scrollbar, which must be between the "lower" field and "upper - page_size." The "page_size" field represents the size of the visible scrollable area. The "step_increment" and "page_increment" fields are used when the user asks to step down (using the small stepper arrows) or page down (using for example the PageDown key).

Details

GtkScrollbar

```
typedef struct _GtkScrollbar GtkScrollbar;
```

The GtkScrollbar struct does not contain any public data.

Style Properties

The "fixed-slider-length" style property

```
  "fixed-slider-length"  gboolean              : Read
```

Don't change slider size, just lock it to the minimum length.

Default value: FALSE

The "has-backward-stepper" style property

```
"has-backward-stepper" gboolean                : Read
```

Display the standard backward arrow button.

Default value: TRUE

The "has-forward-stepper" style property

```
"has-forward-stepper"  gboolean                : Read
```

Display the standard forward arrow button.

Default value: TRUE

The "has-secondary-backward-stepper" style property

```
"has-secondary-backward-stepper" gboolean              : Read
```

Display a second backward arrow button on the opposite end of the scrollbar.

Default value: FALSE

The "has-secondary-forward-stepper" style property

```
"has-secondary-forward-stepper" gboolean                : Read
```

Display a secondary forward arrow button on the opposite end of the scrollbar.

Default value: FALSE

The "min-slider-length" style property

```
"min-slider-length"    gint                     : Read
```

Minimum length of scrollbar slider.

Allowed values: >= 0

Default value: 21

See Also

GtkHScrollbar	a horizontal scrollbar.
GtkVScrollbar	a vertical scrollbar.
GtkAdjustment	connects scrollbars to the widget being scrolled.
GtkScrolledWindow	convenient widget for setting up scrolling.

<< GtkScale **GtkSeparator >>**

GtkSeparator

GtkSeparator — Base class for GtkHSeparator and GtkVSeparator

Synopsis

```
#include <gtk/gtk.h>

            GtkSeparator;
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkWidget
               +----GtkSeparator
                     +----GtkHSeparator
                     +----GtkVSeparator
```

Implemented Interfaces

GtkSeparator implements AtkImplementorIface.

Description

The GtkSeparator widget is an abstract class, used only for deriving the subclasses GtkHSeparator and

GtkVSeparator.

Details

GtkSeparator

```
typedef struct _GtkSeparator GtkSeparator;
```

The GtkSeparator-struct struct contains private data only.

<< GtkScrollbar **GtkWidget >>**

GtkWidget

GtkWidget — Base class for all widgets

Synopsis

```
#include <gtk/gtk.h>

            GtkWidget;
            GtkWidgetClass;
enum        GtkWidgetFlags;
#define     GTK_WIDGET_TYPE                 (wid)
#define     GTK_WIDGET_STATE                (wid)
#define     GTK_WIDGET_SAVED_STATE          (wid)
#define     GTK_WIDGET_FLAGS                (wid)
#define     GTK_WIDGET_TOPLEVEL             (wid)
#define     GTK_WIDGET_NO_WINDOW            (wid)
#define     GTK_WIDGET_REALIZED             (wid)
#define     GTK_WIDGET_MAPPED               (wid)
#define     GTK_WIDGET_VISIBLE              (wid)
#define     GTK_WIDGET_DRAWABLE             (wid)
#define     GTK_WIDGET_SENSITIVE            (wid)
#define     GTK_WIDGET_PARENT_SENSITIVE     (wid)
#define     GTK_WIDGET_IS_SENSITIVE         (wid)
#define     GTK_WIDGET_CAN_FOCUS            (wid)
#define     GTK_WIDGET_HAS_FOCUS            (wid)
#define     GTK_WIDGET_CAN_DEFAULT          (wid)
#define     GTK_WIDGET_RECEIVES_DEFAULT     (wid)
#define     GTK_WIDGET_HAS_DEFAULT          (wid)
#define     GTK_WIDGET_HAS_GRAB             (wid)
#define     GTK_WIDGET_RC_STYLE             (wid)
#define     GTK_WIDGET_COMPOSITE_CHILD      (wid)
#define     GTK_WIDGET_APP_PAINTABLE        (wid)
#define     GTK_WIDGET_DOUBLE_BUFFERED      (wid)
#define     GTK_WIDGET_SET_FLAGS            (wid,flag)
#define     GTK_WIDGET_UNSET_FLAGS          (wid,flag)
void        (*GtkCallback)                  (GtkWidget *widget,
                                             gpointer data);
            GtkRequisition;
struct      GtkAllocation;
```

```
            GtkSelectionData;
            GtkWidgetAuxInfo;
            GtkWidgetShapeInfo;
enum        GtkWidgetHelpType;
GtkWidget*  gtk_widget_new                  (GType type,
                                             const gchar *first_property_name,
                                             ...);
GtkWidget*  gtk_widget_ref                  (GtkWidget *widget);
void        gtk_widget_unref                (GtkWidget *widget);
void        gtk_widget_destroy              (GtkWidget *widget);
void        gtk_widget_destroyed            (GtkWidget *widget,
                                             GtkWidget **widget_pointer);
void        gtk_widget_set                  (GtkWidget *widget,
                                             const gchar *first_property_name,
                                             ...);
void        gtk_widget_unparent             (GtkWidget *widget);
void        gtk_widget_show                 (GtkWidget *widget);
void        gtk_widget_show_now             (GtkWidget *widget);
void        gtk_widget_hide                 (GtkWidget *widget);
void        gtk_widget_show_all             (GtkWidget *widget);
void        gtk_widget_hide_all             (GtkWidget *widget);
void        gtk_widget_map                  (GtkWidget *widget);
void        gtk_widget_unmap                (GtkWidget *widget);
void        gtk_widget_realize              (GtkWidget *widget);
void        gtk_widget_unrealize            (GtkWidget *widget);
void        gtk_widget_queue_draw           (GtkWidget *widget);
void        gtk_widget_queue_resize         (GtkWidget *widget);
void        gtk_widget_queue_resize_no_redraw
                                            (GtkWidget *widget);
void        gtk_widget_draw                 (GtkWidget *widget,
                                             GdkRectangle *area);
void        gtk_widget_size_request         (GtkWidget *widget,
                                             GtkRequisition *requisition);
void        gtk_widget_get_child_requisition
                                            (GtkWidget *widget,
                                             GtkRequisition *requisition);
void        gtk_widget_size_allocate        (GtkWidget *widget,
                                             GtkAllocation *allocation);
void        gtk_widget_add_accelerator      (GtkWidget *widget,
                                             const gchar *accel_signal,
                                             GtkAccelGroup *accel_group,
                                             guint accel_key,
                                             GdkModifierType accel_mods,
                                             GtkAccelFlags accel_flags);
gboolean    gtk_widget_remove_accelerator   (GtkWidget *widget,
                                             GtkAccelGroup *accel_group,
                                             guint accel_key,
```

```
                                                GdkModifierType accel_mods);
void        gtk_widget_set_accel_path          (GtkWidget *widget,
                                                const gchar *accel_path,
                                                GtkAccelGroup *accel_group);
GList*      gtk_widget_list_accel_closures     (GtkWidget *widget);
gboolean    gtk_widget_can_activate_accel      (GtkWidget *widget,
                                                guint signal_id);
gboolean    gtk_widget_event                   (GtkWidget *widget,
                                                GdkEvent *event);
gboolean    gtk_widget_activate                (GtkWidget *widget);
void        gtk_widget_reparent                (GtkWidget *widget,
                                                GtkWidget *new_parent);
gboolean    gtk_widget_intersect               (GtkWidget *widget,
                                                GdkRectangle *area,
                                                GdkRectangle *intersection);
gboolean    gtk_widget_is_focus                (GtkWidget *widget);
void        gtk_widget_grab_focus              (GtkWidget *widget);
void        gtk_widget_grab_default            (GtkWidget *widget);
void        gtk_widget_set_name                (GtkWidget *widget,
                                                const gchar *name);
G_CONST_RETURN gchar* gtk_widget_get_name      (GtkWidget *widget);
void        gtk_widget_set_state               (GtkWidget *widget,
                                                GtkStateType state);
void        gtk_widget_set_sensitive           (GtkWidget *widget,
                                                gboolean sensitive);
void        gtk_widget_set_parent              (GtkWidget *widget,
                                                GtkWidget *parent);
void        gtk_widget_set_parent_window       (GtkWidget *widget,
                                                GdkWindow *parent_window);
GdkWindow*  gtk_widget_get_parent_window       (GtkWidget *widget);
void        gtk_widget_set_uposition           (GtkWidget *widget,
                                                gint x,
                                                gint y);
void        gtk_widget_set_usize               (GtkWidget *widget,
                                                gint width,
                                                gint height);
void        gtk_widget_set_events              (GtkWidget *widget,
                                                gint events);
void        gtk_widget_add_events              (GtkWidget *widget,
                                                gint events);
void        gtk_widget_set_extension_events    (GtkWidget *widget,
                                                GdkExtensionMode mode);
GdkExtensionMode gtk_widget_get_extension_events
                                               (GtkWidget *widget);
GtkWidget*  gtk_widget_get_toplevel            (GtkWidget *widget);
GtkWidget*  gtk_widget_get_ancestor            (GtkWidget *widget,
                                                GType widget_type);
```

```
GdkColormap* gtk_widget_get_colormap        (GtkWidget *widget);
void        gtk_widget_set_colormap         (GtkWidget *widget,
                                             GdkColormap *colormap);
GdkVisual*  gtk_widget_get_visual           (GtkWidget *widget);
gint        gtk_widget_get_events           (GtkWidget *widget);
void        gtk_widget_get_pointer          (GtkWidget *widget,
                                             gint *x,
                                             gint *y);
gboolean    gtk_widget_is_ancestor          (GtkWidget *widget,
                                             GtkWidget *ancestor);
gboolean    gtk_widget_translate_coordinates
                                            (GtkWidget *src_widget,
                                             GtkWidget *dest_widget,
                                             gint src_x,
                                             gint src_y,
                                             gint *dest_x,
                                             gint *dest_y);
gboolean    gtk_widget_hide_on_delete       (GtkWidget *widget);
void        gtk_widget_set_style            (GtkWidget *widget,
                                             GtkStyle *style);
#define     gtk_widget_set_rc_style         (widget)
void        gtk_widget_ensure_style         (GtkWidget *widget);
GtkStyle*   gtk_widget_get_style            (GtkWidget *widget);
#define     gtk_widget_restore_default_style(widget)
void        gtk_widget_reset_rc_styles      (GtkWidget *widget);
void        gtk_widget_push_colormap        (GdkColormap *cmap);
void        gtk_widget_pop_colormap         (void);
void        gtk_widget_set_default_colormap (GdkColormap *colormap);
GtkStyle*   gtk_widget_get_default_style    (void);
GdkColormap* gtk_widget_get_default_colormap
                                            (void);
GdkVisual*  gtk_widget_get_default_visual   (void);
void        gtk_widget_set_direction        (GtkWidget *widget,
                                             GtkTextDirection dir);
enum        GtkTextDirection;
GtkTextDirection gtk_widget_get_direction   (GtkWidget *widget);
void        gtk_widget_set_default_direction
                                            (GtkTextDirection dir);
GtkTextDirection gtk_widget_get_default_direction
                                            (void);
void        gtk_widget_shape_combine_mask   (GtkWidget *widget,
                                             GdkBitmap *shape_mask,
                                             gint offset_x,
                                             gint offset_y);
void        gtk_widget_path                 (GtkWidget *widget,
                                             guint *path_length,
                                             gchar **path,
```

```
                                            gchar **path_reversed);
void        gtk_widget_class_path           (GtkWidget *widget,
                                             guint *path_length,
                                             gchar **path,
                                             gchar **path_reversed);
gchar*      gtk_widget_get_composite_name   (GtkWidget *widget);
void        gtk_widget_modify_style         (GtkWidget *widget,
                                             GtkRcStyle *style);
GtkRcStyle* gtk_widget_get_modifier_style   (GtkWidget *widget);
void        gtk_widget_modify_fg            (GtkWidget *widget,
                                             GtkStateType state,
                                             const GdkColor *color);
void        gtk_widget_modify_bg            (GtkWidget *widget,
                                             GtkStateType state,
                                             const GdkColor *color);
void        gtk_widget_modify_text          (GtkWidget *widget,
                                             GtkStateType state,
                                             const GdkColor *color);
void        gtk_widget_modify_base          (GtkWidget *widget,
                                             GtkStateType state,
                                             const GdkColor *color);
void        gtk_widget_modify_font          (GtkWidget *widget,
                                             PangoFontDescription *font_desc);
PangoContext* gtk_widget_create_pango_context
                                            (GtkWidget *widget);
PangoContext* gtk_widget_get_pango_context  (GtkWidget *widget);
PangoLayout* gtk_widget_create_pango_layout (GtkWidget *widget,
                                             const gchar *text);
GdkPixbuf*  gtk_widget_render_icon          (GtkWidget *widget,
                                             const gchar *stock_id,
                                             GtkIconSize size,
                                             const gchar *detail);
void        gtk_widget_pop_composite_child  (void);
void        gtk_widget_push_composite_child (void);
void        gtk_widget_queue_clear          (GtkWidget *widget);
void        gtk_widget_queue_clear_area     (GtkWidget *widget,
                                             gint x,
                                             gint y,
                                             gint width,
                                             gint height);
void        gtk_widget_queue_draw_area      (GtkWidget *widget,
                                             gint x,
                                             gint y,
                                             gint width,
                                             gint height);
void        gtk_widget_reset_shapes         (GtkWidget *widget);
void        gtk_widget_set_app_paintable    (GtkWidget *widget,
```

```
                                                gboolean app_paintable);
void        gtk_widget_set_double_buffered  (GtkWidget *widget,
                                             gboolean double_buffered);
void        gtk_widget_set_redraw_on_allocate
                                            (GtkWidget *widget,
                                             gboolean redraw_on_allocate);
void        gtk_widget_set_composite_name   (GtkWidget *widget,
                                             const gchar *name);
gboolean    gtk_widget_set_scroll_adjustments
                                            (GtkWidget *widget,
                                             GtkAdjustment *hadjustment,
                                             GtkAdjustment *vadjustment);
gboolean    gtk_widget_mnemonic_activate    (GtkWidget *widget,
                                             gboolean group_cycling);
void        gtk_widget_class_install_style_property
                                            (GtkWidgetClass *klass,
                                             GParamSpec *pspec);
void        gtk_widget_class_install_style_property_parser
                                            (GtkWidgetClass *klass,
                                             GParamSpec *pspec,
                                             GtkRcPropertyParser parser);
GParamSpec* gtk_widget_class_find_style_property
                                            (GtkWidgetClass *klass,
                                             const gchar *property_name);
GParamSpec** gtk_widget_class_list_style_properties
                                            (GtkWidgetClass *klass,
                                             guint *n_properties);
GdkRegion*  gtk_widget_region_intersect     (GtkWidget *widget,
                                             GdkRegion *region);
gint        gtk_widget_send_expose          (GtkWidget *widget,
                                             GdkEvent *event);
void        gtk_widget_style_get            (GtkWidget *widget,
                                             const gchar *first_property_name,
                                             ...);
void        gtk_widget_style_get_property   (GtkWidget *widget,
                                             const gchar *property_name,
                                             GValue *value);
void        gtk_widget_style_get_valist     (GtkWidget *widget,
                                             const gchar *first_property_name,
                                             va_list var_args);
AtkObject*  gtk_widget_get_accessible       (GtkWidget *widget);
gboolean    gtk_widget_child_focus          (GtkWidget *widget,
                                             GtkDirectionType direction);
void        gtk_widget_child_notify         (GtkWidget *widget,
                                             const gchar *child_property);
void        gtk_widget_freeze_child_notify  (GtkWidget *widget);
gboolean    gtk_widget_get_child_visible    (GtkWidget *widget);
```

```
GtkWidget*  gtk_widget_get_parent           (GtkWidget *widget);
GtkSettings* gtk_widget_get_settings        (GtkWidget *widget);
GtkClipboard* gtk_widget_get_clipboard      (GtkWidget *widget,
                                             GdkAtom selection);
GdkDisplay* gtk_widget_get_display          (GtkWidget *widget);
GdkWindow*  gtk_widget_get_root_window      (GtkWidget *widget);
GdkScreen*  gtk_widget_get_screen           (GtkWidget *widget);
gboolean    gtk_widget_has_screen           (GtkWidget *widget);
void        gtk_widget_get_size_request     (GtkWidget *widget,
                                             gint *width,
                                             gint *height);
#define     gtk_widget_pop_visual           ()
#define     gtk_widget_push_visual          (visual)
void        gtk_widget_set_child_visible    (GtkWidget *widget,
                                             gboolean is_visible);
#define     gtk_widget_set_default_visual   (visual)
void        gtk_widget_set_size_request     (GtkWidget *widget,
                                             gint width,
                                             gint height);
#define     gtk_widget_set_visual           (widget,visual)
void        gtk_widget_thaw_child_notify    (GtkWidget *widget);
void        gtk_widget_set_no_show_all      (GtkWidget *widget,
                                             gboolean no_show_all);
gboolean    gtk_widget_get_no_show_all      (GtkWidget *widget);
GList*      gtk_widget_list_mnemonic_labels (GtkWidget *widget);
void        gtk_widget_add_mnemonic_label   (GtkWidget *widget,
                                             GtkWidget *label);
void        gtk_widget_remove_mnemonic_label
                                            (GtkWidget *widget,
                                             GtkWidget *label);

GtkRequisition* gtk_requisition_copy        (const GtkRequisition *requisition);
void        gtk_requisition_free            (GtkRequisition *requisition);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkWidget
               +----GtkContainer
               +----GtkMisc
               +----GtkCalendar
               +----GtkCellView
               +----GtkDrawingArea
```

```
                +----GtkEntry
                +----GtkRuler
                +----GtkRange
                +----GtkSeparator
                +----GtkInvisible
                +----GtkOldEditable
                +----GtkPreview
                +----GtkProgress
```

Known Derived Interfaces

GtkWidget is required by GtkFileChooser and GtkCellEditable.

Implemented Interfaces

GtkWidget implements AtkImplementorIface.

Properties

```
  "app-paintable"        gboolean             : Read / Write
  "can-default"          gboolean             : Read / Write
  "can-focus"            gboolean             : Read / Write
  "composite-child"      gboolean             : Read
  "events"               GdkEventMask         : Read / Write
  "extension-events"     GdkExtensionMode     : Read / Write
  "has-default"          gboolean             : Read / Write
  "has-focus"            gboolean             : Read / Write
  "height-request"       gint                 : Read / Write
  "is-focus"             gboolean             : Read / Write
  "name"                 gchararray           : Read / Write
  "no-show-all"          gboolean             : Read / Write
  "parent"               GtkContainer         : Read / Write
  "receives-default"     gboolean             : Read / Write
  "sensitive"            gboolean             : Read / Write
  "style"                GtkStyle             : Read / Write
  "visible"              gboolean             : Read / Write
  "width-request"        gint                 : Read / Write
```

Style Properties

```
  "cursor-aspect-ratio"  gfloat               : Read
```

```
  "cursor-color"         GdkColor              : Read
  "focus-line-pattern"   gchararray            : Read
  "focus-line-width"     gint                  : Read
  "focus-padding"        gint                  : Read
  "interior-focus"       gboolean              : Read
  "secondary-cursor-color" GdkColor              : Read
```

Signal Prototypes

```
"accel-closures-changed"
            void        user_function      (GtkWidget *widget,
                                            gpointer user_data);
"button-press-event"
            gboolean    user_function      (GtkWidget *widget,
                                            GdkEventButton *event,
                                            gpointer user_data);
"button-release-event"
            gboolean    user_function      (GtkWidget *widget,
                                            GdkEventButton *event,
                                            gpointer user_data);
"can-activate-accel"
            gboolean    user_function      (GtkWidget *widget,
                                            guint signal_id,
                                            gpointer user_data);
"child-notify"
            void        user_function      (GtkWidget *widget,
                                            GParamSpec *pspec,
                                            gpointer user_data);
"client-event"
            gboolean    user_function      (GtkWidget *widget,
                                            GdkEventClient *event,
                                            gpointer user_data);
"configure-event"
            gboolean    user_function      (GtkWidget *widget,
                                            GdkEventConfigure *event,
                                            gpointer user_data);
"delete-event"
            gboolean    user_function      (GtkWidget *widget,
                                            GdkEvent *event,
                                            gpointer user_data);
"destroy-event"
            gboolean    user_function      (GtkWidget *widget,
                                            GdkEvent *event,
                                            gpointer user_data);
"direction-changed"
```

```
            void        user_function      (GtkWidget *widget,
                                            GtkTextDirection arg1,
                                            gpointer user_data);
"drag-begin"
            void        user_function      (GtkWidget *widget,
                                            GdkDragContext *drag_context,
                                            gpointer user_data);
"drag-data-delete"
            void        user_function      (GtkWidget *widget,
                                            GdkDragContext *drag_context,
                                            gpointer user_data);
"drag-data-get"
            void        user_function      (GtkWidget *widget,
                                            GdkDragContext *drag_context,
                                            GtkSelectionData *data,
                                            guint info,
                                            guint time,
                                            gpointer user_data);
"drag-data-received"
            void        user_function      (GtkWidget *widget,
                                            GdkDragContext *drag_context,
                                            gint x,
                                            gint y,
                                            GtkSelectionData *data,
                                            guint info,
                                            guint time,
                                            gpointer user_data);
"drag-drop" gboolean    user_function      (GtkWidget *widget,
                                            GdkDragContext *drag_context,
                                            gint x,
                                            gint y,
                                            guint time,
                                            gpointer user_data);
"drag-end"  void        user_function      (GtkWidget *widget,
                                            GdkDragContext *drag_context,
                                            gpointer user_data);
"drag-leave"
            void        user_function      (GtkWidget *widget,
                                            GdkDragContext *drag_context,
                                            guint time,
                                            gpointer user_data);
"drag-motion"
            gboolean    user_function      (GtkWidget *widget,
                                            GdkDragContext *drag_context,
                                            gint x,
                                            gint y,
                                            guint time,
```

```
                                        gpointer user_data);
"enter-notify-event"
            gboolean    user_function   (GtkWidget *widget,
                                         GdkEventCrossing *event,
                                         gpointer user_data);
"event"     gboolean    user_function   (GtkWidget *widget,
                                         GdkEvent *event,
                                         gpointer user_data);
"event-after"
            void        user_function   (GtkWidget *widget,
                                         GdkEvent *event,
                                         gpointer user_data);
"expose-event"
            gboolean    user_function   (GtkWidget *widget,
                                         GdkEventExpose *event,
                                         gpointer user_data);
"focus"     gboolean    user_function   (GtkWidget *widget,
                                         GtkDirectionType arg1,
                                         gpointer user_data);
"focus-in-event"
            gboolean    user_function   (GtkWidget *widget,
                                         GdkEventFocus *event,
                                         gpointer user_data);
"focus-out-event"
            gboolean    user_function   (GtkWidget *widget,
                                         GdkEventFocus *event,
                                         gpointer user_data);
"grab-focus"
            void        user_function   (GtkWidget *widget,
                                         gpointer user_data);
"grab-notify"
            void        user_function   (GtkWidget *widget,
                                         gboolean arg1,
                                         gpointer user_data);
"hide"      void        user_function   (GtkWidget *widget,
                                         gpointer user_data);
"hierarchy-changed"
            void        user_function   (GtkWidget *widget,
                                         GtkWidget *widget2,
                                         gpointer user_data);
"key-press-event"
            gboolean    user_function   (GtkWidget *widget,
                                         GdkEventKey *event,
                                         gpointer user_data);
"key-release-event"
            gboolean    user_function   (GtkWidget *widget,
                                         GdkEventKey *event,
```

```
                                        gpointer user_data);
"leave-notify-event"
            gboolean    user_function   (GtkWidget *widget,
                                         GdkEventCrossing *event,
                                         gpointer user_data);
"map"       void        user_function   (GtkWidget *widget,
                                         gpointer user_data);
"map-event" gboolean    user_function   (GtkWidget *widget,
                                         GdkEvent *event,
                                         gpointer user_data);
"mnemonic-activate"
            gboolean    user_function   (GtkWidget *widget,
                                         gboolean arg1,
                                         gpointer user_data);
"motion-notify-event"
            gboolean    user_function   (GtkWidget *widget,
                                         GdkEventMotion *event,
                                         gpointer user_data);
"no-expose-event"
            gboolean    user_function   (GtkWidget *widget,
                                         GdkEventNoExpose *event,
                                         gpointer user_data);
"parent-set"
            void        user_function   (GtkWidget *widget,
                                         GtkObject *old_parent,
                                         gpointer user_data);
"popup-menu"
            gboolean    user_function   (GtkWidget *widget,
                                         gpointer user_data);
"property-notify-event"
            gboolean    user_function   (GtkWidget *widget,
                                         GdkEventProperty *event,
                                         gpointer user_data);
"proximity-in-event"
            gboolean    user_function   (GtkWidget *widget,
                                         GdkEventProximity *event,
                                         gpointer user_data);
"proximity-out-event"
            gboolean    user_function   (GtkWidget *widget,
                                         GdkEventProximity *event,
                                         gpointer user_data);
"realize"   void        user_function   (GtkWidget *widget,
                                         gpointer user_data);
"screen-changed"
            void        user_function   (GtkWidget *widget,
                                         GdkScreen *arg1,
                                         gpointer user_data);
```

```
"scroll-event"
            gboolean    user_function      (GtkWidget *widget,
                                            GdkEventScroll *event,
                                            gpointer user_data);
"selection-clear-event"
            gboolean    user_function      (GtkWidget *widget,
                                            GdkEventSelection *event,
                                            gpointer user_data);
"selection-get"
            void        user_function      (GtkWidget *widget,
                                            GtkSelectionData *data,
                                            guint info,
                                            guint time,
                                            gpointer user_data);
"selection-notify-event"
            gboolean    user_function      (GtkWidget *widget,
                                            GdkEventSelection *event,
                                            gpointer user_data);
"selection-received"
            void        user_function      (GtkWidget *widget,
                                            GtkSelectionData *data,
                                            guint time,
                                            gpointer user_data);
"selection-request-event"
            gboolean    user_function      (GtkWidget *widget,
                                            GdkEventSelection *event,
                                            gpointer user_data);
"show"      void        user_function      (GtkWidget *widget,
                                            gpointer user_data);
"show-help" gboolean    user_function      (GtkWidget *widget,
                                            GtkWidgetHelpType arg1,
                                            gpointer user_data);
"size-allocate"
            void        user_function      (GtkWidget *widget,
                                            GtkAllocation *allocation,
                                            gpointer user_data);
"size-request"
            void        user_function      (GtkWidget *widget,
                                            GtkRequisition *requisition,
                                            gpointer user_data);
"state-changed"
            void        user_function      (GtkWidget *widget,
                                            GtkStateType state,
                                            gpointer user_data);
"style-set" void        user_function      (GtkWidget *widget,
                                            GtkStyle *previous_style,
                                            gpointer user_data);
```

```
"unmap"     void        user_function      (GtkWidget *widget,
                                            gpointer user_data);
"unmap-event"
            gboolean    user_function      (GtkWidget *widget,
                                            GdkEvent *event,
                                            gpointer user_data);
"unrealize" void        user_function      (GtkWidget *widget,
                                            gpointer user_data);
"visibility-notify-event"
            gboolean    user_function      (GtkWidget *widget,
                                            GdkEventVisibility *event,
                                            gpointer user_data);
"window-state-event"
            gboolean    user_function      (GtkWidget *widget,
                                            GdkEventWindowState *event,
                                            gpointer user_data);
```

Description

GtkWidget introduces *style properties* - these are basically object properties that are stored not on the object, but in the style object associated to the widget. Style properties are set in resource files. This mechanism is used for configuring such things as the location of the scrollbar arrows through the theme, giving theme authors more control over the look of applications without the need to write a theme engine in C.

Use `gtk_widget_class_install_style_property()` to install style properties for a widget class, `gtk_widget_class_find_style_property()` or `gtk_widget_class_list_style_properties()` to get information about existing style properties and `gtk_widget_style_get_property()`, `gtk_widget_style_get()` or `gtk_widget_style_get_valist()` to obtain the value of a style property.

Details

GtkWidget

```
typedef struct {
  /* The style for the widget. The style contains the
   *  colors the widget should be drawn in for each state
   *  along with graphics contexts used to draw with and
   *  the font to use for text.
   */
  GtkStyle *style;

  /* The widgets desired size.
   */
  GtkRequisition requisition;

  /* The widgets allocated size.
   */
```

```
  GtkAllocation allocation;

  /* The widgets window or its parent window if it does
   *  not have a window. (Which will be indicated by the
   *  GTK_NO_WINDOW flag being set).
   */
  GdkWindow *window;

  /* The widgets parent.
   */
  GtkWidget *parent;
} GtkWidget;
```

GtkStyle *`style`;	The style for the widget. The style contains the colors the widget should be drawn in for each state along with graphics contexts used to draw with and the font to use for text.
GtkRequisition `requisition`;	The widgets desired size.
GtkAllocation `allocation`;	The widgets allocated size.
GdkWindow *`window`;	The widgets window or its parent window if it does not have a window. (Which will be indicated by the GTK_NO_WINDOW flag being set).
GtkWidget *`parent`;	

GtkWidgetClass

```
typedef struct {
  /* The object class structure needs to be the first
   *  element in the widget class structure in order for
   *  the class mechanism to work correctly. This allows a
   *  GtkWidgetClass pointer to be cast to a GtkObjectClass
   *  pointer.
   */
  GtkObjectClass parent_class;


  guint activate_signal;

  guint set_scroll_adjustments_signal;
} GtkWidgetClass;
```

`activate_signal` The signal to emit when a widget of this class is activated, `gtk_widget_activate()` handles the emission. Implementation of this signal is optional.

`set_scroll_adjustment_signal` This signal is emitted when a widget of this class is added to a scrolling aware parent, `gtk_widget_set_scroll_adjustments()` handles the emission. Implementation of this signal is optional.

---

enum GtkWidgetFlags

```
typedef enum
{
  GTK_TOPLEVEL         = 1 << 4,
  GTK_NO_WINDOW        = 1 << 5,
  GTK_REALIZED         = 1 << 6,
  GTK_MAPPED           = 1 << 7,
  GTK_VISIBLE          = 1 << 8,
  GTK_SENSITIVE        = 1 << 9,
  GTK_PARENT_SENSITIVE = 1 << 10,
  GTK_CAN_FOCUS        = 1 << 11,
  GTK_HAS_FOCUS        = 1 << 12,

  /* widget is allowed to receive the default via gtk_widget_grab_default
   * and will reserve space to draw the default if possible
   */
  GTK_CAN_DEFAULT      = 1 << 13,

  /* the widget currently is receiving the default action and should be drawn
   * appropriately if possible
   */
  GTK_HAS_DEFAULT      = 1 << 14,

  GTK_HAS_GRAB         = 1 << 15,
  GTK_RC_STYLE         = 1 << 16,
  GTK_COMPOSITE_CHILD  = 1 << 17,
  GTK_NO_REPARENT      = 1 << 18,
  GTK_APP_PAINTABLE    = 1 << 19,

  /* the widget when focused will receive the default action and have
   * HAS_DEFAULT set even if there is a different widget set as default
   */
  GTK_RECEIVES_DEFAULT = 1 << 20,

  GTK_DOUBLE_BUFFERED  = 1 << 21,
  GTK_NO_SHOW_ALL      = 1 << 22
} GtkWidgetFlags;
```

Tells about certain properties of the widget.

GTK_TOPLEVEL	widgets without a real parent, as there are GtkWindows and GtkMenus have this flag set throughout their lifetime. Toplevel widgets always contain their own GdkWindow.
GTK_NO_WINDOW	Indicative for a widget that does not provide its own GdkWindow. Visible action (e.g. drawing) is performed on the parent's GdkWindow.
GTK_REALIZED	Set by gtk_widget_realize(), unset by gtk_widget_unrealize(). A realized widget has an associated GdkWindow.

GTK_MAPPED	Set by gtk_widget_map(), unset by gtk_widget_unmap(). Only realized widgets can be mapped. It means that gdk_window_show() has been called on the widgets window(s).
GTK_VISIBLE	Set by gtk_widget_show(), unset by gtk_widget_hide(). Implies that a widget will be mapped as soon as its parent is mapped.
GTK_SENSITIVE	Set and unset by gtk_widget_set_sensitive(). The sensitivity of a widget determines whether it will receive certain events (e.g. button or key presses). One premise for the widgets sensitivity is to have this flag set.
GTK_PARENT_SENSITIVE	Set and unset by gtk_widget_set_sensitive() operations on the parents of the widget. This is the second premise for the widgets sensitivity. Once it has GTK_SENSITIVE and GTK_PARENT_SENSITIVE set, its state is effectively sensitive. This is expressed (and can be examined) by the GTK_WIDGET_IS_SENSITIVE macro.
GTK_CAN_FOCUS	Determines whether a widget is able to handle focus grabs.
GTK_HAS_FOCUS	Set by gtk_widget_grab_focus() for widgets that also have GTK_CAN_FOCUS set. The flag will be unset once another widget grabs the focus.
GTK_CAN_DEFAULT	The widget is allowed to receive the default action via gtk_widget_grab_default().
GTK_HAS_DEFAULT	The widget currently is receiving the default action.
GTK_HAS_GRAB	Set by gtk_grab_add(), unset by gtk_grab_remove(). It means that the widget is in the grab_widgets stack, and will be the preferred one for receiving events other than ones of cosmetic value.
GTK_RC_STYLE	Indicates that the widgets style has been looked up through the rc mechanism. It does not imply that the widget actually had a style defined through the rc mechanism.
GTK_COMPOSITE_CHILD	Indicates that the widget is a composite child of its parent; see gtk_widget_push_composite_child(), gtk_widget_pop_composite_child().
GTK_NO_REPARENT	Unused since before GTK+ 1.2, will be removed in a future version.
GTK_APP_PAINTABLE	Set and unset by gtk_widget_set_app_paintable(). Must be set on widgets whose window the application directly draws on, in order to keep GTK+ from overwriting the drawn stuff.
GTK_RECEIVES_DEFAULT	The widget when focused will receive the default action and have GTK_HAS_DEFAULT set even if there is a different widget set as default.
GTK_DOUBLE_BUFFERED	Set and unset by gtk_widget_set_double_buffered(). Indicates that exposes done on the widget should be double-buffered.
GTK_NO_SHOW_ALL	

GTK_WIDGET_TYPE()

```
#define GTK_WIDGET_TYPE(wid)            (GTK_OBJECT_TYPE (wid))
```

Gets the type of a widget.

wid : a GtkWidget.

GTK_WIDGET_STATE()

```
#define GTK_WIDGET_STATE(wid)             (GTK_WIDGET (wid)->state)
```

Returns the current state of the widget, as a GtkStateType.

wid : a GtkWidget.

GTK_WIDGET_SAVED_STATE()

```
#define GTK_WIDGET_SAVED_STATE(wid)       (GTK_WIDGET (wid)->saved_state)
```

Returns the saved state of the widget, as a GtkStateType.

The saved state will be restored when a widget gets sensitive again, after it has been made insensitive with `gtk_widget_set_state()` or `gtk_widget_set_sensitive()`.

wid : a GtkWidget.

GTK_WIDGET_FLAGS()

```
#define GTK_WIDGET_FLAGS(wid)             (GTK_OBJECT_FLAGS (wid))
```

Returns the widget flags from *wid*.

wid : a GtkWidget.

GTK_WIDGET_TOPLEVEL()

```
#define GTK_WIDGET_TOPLEVEL(wid)          ((GTK_WIDGET_FLAGS (wid) & GTK_TOPLEVEL) !=
0)
```

Evaluates to TRUE if the widget is a toplevel widget.

wid : a GtkWidget.

GTK_WIDGET_NO_WINDOW()

```
#define GTK_WIDGET_NO_WINDOW(wid)          ((GTK_WIDGET_FLAGS (wid) & GTK_NO_WINDOW)
!= 0)
```

Evaluates to TRUE if the widget doesn't have an own GdkWindow.

wid : a GtkWidget.

GTK_WIDGET_REALIZED()

```
#define GTK_WIDGET_REALIZED(wid)           ((GTK_WIDGET_FLAGS (wid) & GTK_REALIZED) !=
0)
```

Evaluates to TRUE if the widget is realized.

wid : a GtkWidget.

GTK_WIDGET_MAPPED()

```
#define GTK_WIDGET_MAPPED(wid)             ((GTK_WIDGET_FLAGS (wid) & GTK_MAPPED) !=
0)
```

Evaluates to TRUE if the widget is mapped.

wid : a GtkWidget.

GTK_WIDGET_VISIBLE()

```
#define GTK_WIDGET_VISIBLE(wid)            ((GTK_WIDGET_FLAGS (wid) & GTK_VISIBLE) !=
0)
```

Evaluates to TRUE if the widget is visible.

wid : a GtkWidget.

GTK_WIDGET_DRAWABLE()

```
#define GTK_WIDGET_DRAWABLE(wid)          (GTK_WIDGET_VISIBLE (wid) &&
GTK_WIDGET_MAPPED (wid))
```

Evaluates to TRUE if the widget is mapped and visible.

wid : a GtkWidget.

GTK_WIDGET_SENSITIVE()

```
#define GTK_WIDGET_SENSITIVE(wid)         ((GTK_WIDGET_FLAGS (wid) & GTK_SENSITIVE)
!= 0)
```

Evaluates to TRUE if the GTK_SENSITIVE flag has be set on the widget.

wid : a GtkWidget.

GTK_WIDGET_PARENT_SENSITIVE()

```
#define GTK_WIDGET_PARENT_SENSITIVE(wid)  ((GTK_WIDGET_FLAGS (wid) &
GTK_PARENT_SENSITIVE) != 0)
```

Evaluates to TRUE if the GTK_PARENT_SENSITIVE flag has be set on the widget.

wid : a GtkWidget.

GTK_WIDGET_IS_SENSITIVE()

```
#define     GTK_WIDGET_IS_SENSITIVE(wid)
```

Evaluates to TRUE if the widget is effectively sensitive.

wid : a GtkWidget.

GTK_WIDGET_CAN_FOCUS()

```
#define GTK_WIDGET_CAN_FOCUS(wid)         ((GTK_WIDGET_FLAGS (wid) & GTK_CAN_FOCUS)
!= 0)
```

Evaluates to TRUE if the widget is able to handle focus grabs.

wid : a GtkWidget.

GTK_WIDGET_HAS_FOCUS()

```
#define GTK_WIDGET_HAS_FOCUS(wid)         ((GTK_WIDGET_FLAGS (wid) & GTK_HAS_FOCUS)
!= 0)
```

Evaluates to TRUE if the widget has grabbed the focus and no other widget has done so more recently.

wid : a GtkWidget.

GTK_WIDGET_CAN_DEFAULT()

```
#define GTK_WIDGET_CAN_DEFAULT(wid)       ((GTK_WIDGET_FLAGS (wid) & GTK_CAN_DEFAULT)
!= 0)
```

Evaluates to TRUE if the widget is allowed to receive the default action via `gtk_widget_grab_default()`.

wid : a GtkWidget.

GTK_WIDGET_RECEIVES_DEFAULT()

```
#define GTK_WIDGET_RECEIVES_DEFAULT(wid)  ((GTK_WIDGET_FLAGS (wid) &
GTK_RECEIVES_DEFAULT) != 0)
```

Evaluates to TRUE if the widget when focused will receive the default action even if there is a different widget set as default.

wid : a GtkWidget.

GTK_WIDGET_HAS_DEFAULT()

```
#define GTK_WIDGET_HAS_DEFAULT(wid)       ((GTK_WIDGET_FLAGS (wid) & GTK_HAS_DEFAULT)
!= 0)
```

Evaluates to TRUE if the widget currently is receiving the default action.

wid : a GtkWidget.

GTK_WIDGET_HAS_GRAB()

```
#define GTK_WIDGET_HAS_GRAB(wid)          ((GTK_WIDGET_FLAGS (wid) & GTK_HAS_GRAB) !=
0)
```

Evaluates to TRUE if the widget is in the grab_widgets stack, and will be the preferred one for receiving events other than ones of cosmetic value.

wid : a GtkWidget.

GTK_WIDGET_RC_STYLE()

```
#define GTK_WIDGET_RC_STYLE(wid)          ((GTK_WIDGET_FLAGS (wid) & GTK_RC_STYLE) !=
0)
```

Evaluates to TRUE if the widgets style has been looked up through the rc mechanism.

wid : a GtkWidget.

GTK_WIDGET_COMPOSITE_CHILD()

```
#define GTK_WIDGET_COMPOSITE_CHILD(wid)   ((GTK_WIDGET_FLAGS (wid) &
GTK_COMPOSITE_CHILD) != 0)
```

Evaluates to TRUE if the widget is a composite child of its parent.

wid : a GtkWidget.

GTK_WIDGET_APP_PAINTABLE()

```
#define GTK_WIDGET_APP_PAINTABLE(wid)     ((GTK_WIDGET_FLAGS (wid) &
GTK_APP_PAINTABLE) != 0)
```

Evaluates to TRUE if the GTK_APP_PAINTABLE flag has been set on the widget.

wid : a GtkWidget.

GTK_WIDGET_DOUBLE_BUFFERED()

```
#define GTK_WIDGET_DOUBLE_BUFFERED(wid)   ((GTK_WIDGET_FLAGS (wid) &
GTK_DOUBLE_BUFFERED) != 0)
```

Evaluates to TRUE if the GTK_DOUBLE_BUFFERED flag has been set on the widget.

wid : a GtkWidget.

GTK_WIDGET_SET_FLAGS()

```
#define GTK_WIDGET_SET_FLAGS(wid,flag)    G_STMT_START{ (GTK_WIDGET_FLAGS (wid) |=
(flag)); }G_STMT_END
```

Turns on certain widget flags.

wid : a GtkWidget.
flag : the flags to set.

GTK_WIDGET_UNSET_FLAGS()

```
#define GTK_WIDGET_UNSET_FLAGS(wid,flag)  G_STMT_START{ (GTK_WIDGET_FLAGS (wid) &=
~(flag)); }G_STMT_END
```

Turns off certain widget flags.

wid : a GtkWidget.

flag : the flags to unset.

GtkCallback ()

```
void        (*GtkCallback)                  (GtkWidget *widget,
                                             gpointer data);
```

The type of the callback functions used for e.g. iterating over the children of a container, see `gtk_container_foreach()`.

widget : the widget to operate on
data : user-supplied data

GtkRequisition

```
typedef struct {
  gint width;
  gint height;
} GtkRequisition;
```

A GtkRequisition represents the desired size of a widget. See the section called “Size Requisition” for more information.

gint *width*; the widget's desired width
gint *height*; the widget's desired height

---

struct GtkAllocation

```
struct GtkAllocation {
  gint x;
  gint y;
  gint width;
  gint height;
};
```

A GtkAllocation of a widget represents region which has been allocated to the widget by its parent. It is a subregion of its parents allocation. See the section called “Size Allocation” for more information.

gint *x*; the X position of the widgets area relative to its parents allocation.
gint *y*; the Y position of the widgets area relative to its parents allocation.

gint *width*; the width of the widgets allocated area.

gint *height*; the height of the widgets allocated area.

GtkSelectionData

```
typedef struct {
  GdkAtom       selection;
  GdkAtom       target;
  GdkAtom       type;
  gint          format;
  guchar       *data;
  gint          length;
  GdkDisplay   *display;
} GtkSelectionData;
```

GtkWidgetAuxInfo

```
typedef struct {
  gint x;
  gint y;
  gint width;
  gint height;
  guint x_set : 1;
  guint y_set : 1;
} GtkWidgetAuxInfo;
```

GtkWidgetShapeInfo

```
typedef struct {
  gint16     offset_x;
  gint16     offset_y;
  GdkBitmap *shape_mask;
} GtkWidgetShapeInfo;
```

---

enum GtkWidgetHelpType

```
typedef enum
{
```

```
  GTK_WIDGET_HELP_TOOLTIP,
  GTK_WIDGET_HELP_WHATS_THIS
} GtkWidgetHelpType;
```

---

gtk_widget_new ()

```
GtkWidget*  gtk_widget_new                  (GType type,
                                             const gchar *first_property_name,
                                             ...);
```

This is a convenience function for creating a widget and setting its properties in one go. For example you might write: `gtk_widget_new (GTK_TYPE_LABEL, "label", "Hello World", "xalign", 0.0, NULL)` to create a left-aligned label. Equivalent to `g_object_new()`, but returns a widget so you don't have to cast the object yourself.

type :	type ID of the widget to create
first_property_name :	name of first property to set
... :	value of first property, followed by more properties, `NULL`-terminated
Returns :	a new GtkWidget of type *widget_type*

---

gtk_widget_ref ()

```
GtkWidget*  gtk_widget_ref                  (GtkWidget *widget);
```

Adds a reference to a widget. This function is exactly the same as calling `g_object_ref()`, and exists mostly for historical reasons. It can still be convenient to avoid casting a widget to a GObject, it saves a small amount of typing.

widget :	a GtkWidget
Returns :	the widget that was referenced

---

gtk_widget_unref ()

```
void        gtk_widget_unref                (GtkWidget *widget);
```

Inverse of `gtk_widget_ref()`. Equivalent to `g_object_unref()`.

widget :	a GtkWidget

---

gtk_widget_destroy ()

```
void        gtk_widget_destroy              (GtkWidget *widget);
```

Destroys a widget. Equivalent to `gtk_object_destroy()`, except that you don't have to cast the widget to GtkObject. When a widget is destroyed, it will break any references it holds to other objects. If the widget is inside a container, the widget will be removed from the container. If the widget is a toplevel (derived from GtkWindow), it will be removed from the list of toplevels, and the reference GTK+ holds to it will be removed. Removing a widget from its container or the list of toplevels results in the widget being finalized, unless you've added additional references to the widget with `g_object_ref()`.

In most cases, only toplevel widgets (windows) require explicit destruction, because when you destroy a toplevel its children will be destroyed as well.

widget : a GtkWidget

---

gtk_widget_destroyed ()

```
void        gtk_widget_destroyed            (GtkWidget *widget,
                                             GtkWidget **widget_pointer);
```

This function sets **widget_pointer* to `NULL` if *widget_pointer* != `NULL`. It's intended to be used as a callback connected to the "destroy" signal of a widget. You connect `gtk_widget_destroyed()` as a signal handler, and pass the address of your widget variable as user data. Then when the widget is destroyed, the variable will be set to `NULL`. Useful for example to avoid multiple copies of the same dialog.

widget : a GtkWidget

widget_pointer : address of a variable that contains *widget*

---

gtk_widget_set ()

```
void        gtk_widget_set                  (GtkWidget *widget,
                                             const gchar *first_property_name,
                                             ...);
```

Warning

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

Like `g_object_set()` - there's no reason to use this instead of `g_object_set()`.

widget : a GtkWidget
first_property_name : name of first property to set
... : value of first property, followed by more properties, NULL-terminated

---

gtk_widget_unparent ()

```
void        gtk_widget_unparent             (GtkWidget *widget);
```

This function is only for use in widget implementations. Should be called by implementations of the remove method on GtkContainer, to dissociate a child from the container.

widget : a GtkWidget

---

gtk_widget_show ()

```
void        gtk_widget_show                 (GtkWidget *widget);
```

Flags a widget to be displayed. Any widget that isn't shown will not appear on the screen. If you want to show all the widgets in a container, it's easier to call `gtk_widget_show_all()` on the container, instead of individually showing the widgets.

Remember that you have to show the containers containing a widget, in addition to the widget itself, before it will appear onscreen.

When a toplevel container is shown, it is immediately realized and mapped; other shown widgets are realized and mapped when their toplevel container is realized and mapped.

widget : a GtkWidget

---

gtk_widget_show_now ()

```
void        gtk_widget_show_now             (GtkWidget *widget);
```

Shows a widget. If the widget is an unmapped toplevel widget (i.e. a GtkWindow that has not yet been shown), enter the main loop and wait for the window to actually be mapped. Be careful; because the main loop is running, anything can happen during this function.

widget : a GtkWidget

---

gtk_widget_hide ()

```
void        gtk_widget_hide                 (GtkWidget *widget);
```

Reverses the effects of `gtk_widget_show()`, causing the widget to be hidden (invisible to the user).

widget : a GtkWidget

---

gtk_widget_show_all ()

```
void        gtk_widget_show_all             (GtkWidget *widget);
```

Recursively shows a widget, and any child widgets (if the widget is a container).

widget : a GtkWidget

---

gtk_widget_hide_all ()

```
void        gtk_widget_hide_all             (GtkWidget *widget);
```

Recursively hides a widget and any child widgets.

widget : a GtkWidget

---

gtk_widget_map ()

```
void        gtk_widget_map                  (GtkWidget *widget);
```

This function is only for use in widget implementations. Causes a widget to be mapped if it isn't already.

widget : a GtkWidget

---

gtk_widget_unmap ()

```
void        gtk_widget_unmap                (GtkWidget *widget);
```

This function is only for use in widget implementations. Causes a widget to be unmapped if it's currently mapped.

widget : a GtkWidget

---

gtk_widget_realize ()

```
void        gtk_widget_realize              (GtkWidget *widget);
```

Creates the GDK (windowing system) resources associated with a widget. For example, *widget->window* will be created when a widget is realized. Normally realization happens implicitly; if you show a widget and all its parent containers, then the widget will be realized and mapped automatically.

Realizing a widget requires all the widget's parent widgets to be realized; calling gtk_widget_realize() realizes the widget's parents in addition to *widget* itself. If a widget is not yet inside a toplevel window when you realize it, bad things will happen.

This function is primarily used in widget implementations, and isn't very useful otherwise. Many times when you think you might need it, a better approach is to connect to a signal that will be called after the widget is realized automatically, such as "expose_event". Or simply g_signal_connect_after() to the "realize" signal.

widget : a GtkWidget

---

gtk_widget_unrealize ()

```
void        gtk_widget_unrealize            (GtkWidget *widget);
```

This function is only useful in widget implementations. Causes a widget to be unrealized (frees all GDK resources associated with the widget, such as *widget->window*).

widget : a GtkWidget

---

gtk_widget_queue_draw ()

```
void        gtk_widget_queue_draw           (GtkWidget *widget);
```

Equivalent to calling gtk_widget_queue_draw_area() for the entire area of a widget.

widget : a GtkWidget

---

gtk_widget_queue_resize ()

```
void        gtk_widget_queue_resize         (GtkWidget *widget);
```

This function is only for use in widget implementations. Flags a widget to have its size renegotiated; should be called when a widget for some reason has a new size request. For example, when you change the text in a GtkLabel, GtkLabel queues a resize to ensure there's enough space for the new text.

widget : a GtkWidget

---

gtk_widget_queue_resize_no_redraw ()

```
void        gtk_widget_queue_resize_no_redraw
                                            (GtkWidget *widget);
```

This function works like `gtk_widget_queue_resize()`, except that the widget is not invalidated.

widget : a GtkWidget

Since 2.4

---

gtk_widget_draw ()

```
void        gtk_widget_draw                 (GtkWidget *widget,
                                             GdkRectangle *area);
```

Warning

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

In GTK+ 1.2, this function would immediately render the region *area* of a widget, by invoking the virtual draw method of a widget. In GTK+ 2.0, the draw method is gone, and instead `gtk_widget_draw()` simply invalidates the specified region of the widget, then updates the invalid region of the widget immediately. Usually you don't want to update the region immediately for performance reasons, so in general `gtk_widget_queue_draw_area()` is a better choice if you want to draw a region of a widget.

widget : a GtkWidget
area : area to draw

---

gtk_widget_size_request ()

```
void        gtk_widget_size_request         (GtkWidget *widget,
                                             GtkRequisition *requisition);
```

This function is typically used when implementing a GtkContainer subclass. Obtains the preferred size of a widget. The container uses this information to arrange its child widgets and decide what size allocations to give them with `gtk_widget_size_allocate()`.

You can also call this function from an application, with some caveats. Most notably, getting a size request requires the widget to be associated with a screen, because font information may be needed. Multihead-aware applications should keep this in mind.

Also remember that the size request is not necessarily the size a widget will actually be allocated.

See also `gtk_widget_get_child_requisition()`.

widget : a GtkWidget
requisition : a GtkRequisition to be filled in

---

gtk_widget_get_child_requisition ()

```
void        gtk_widget_get_child_requisition
                                            (GtkWidget *widget,
                                             GtkRequisition *requisition);
```

This function is only for use in widget implementations. Obtains *`widget->requisition`*, unless someone has forced a particular geometry on the widget (e.g. with `gtk_widget_set_usize()`), in which case it returns that geometry instead of the widget's requisition.

This function differs from `gtk_widget_size_request()` in that it retrieves the last size request value from *`widget->requisition`*, while `gtk_widget_size_request()` actually calls the "size_request" method on *`widget`* to compute the size request and fill in *`widget->requisition`*, and only then returns *`widget->requisition`*.

Because this function does not call the "size_request" method, it can only be used when you know that *`widget->requisition`* is up-to-date, that is, `gtk_widget_size_request()` has been called since the last time a resize was queued. In general, only container implementations have this information; applications should use `gtk_widget_size_request()`.

widget : a GtkWidget

requisition : a GtkRequisition to be filled in

---

gtk_widget_size_allocate ()

```
void        gtk_widget_size_allocate        (GtkWidget *widget,
                                             GtkAllocation *allocation);
```

This function is only used by GtkContainer subclasses, to assign a size and position to their child widgets.

widget : a GtkWidget
allocation : position and size to be allocated to *widget*

---

gtk_widget_add_accelerator ()

```
void        gtk_widget_add_accelerator      (GtkWidget *widget,
                                             const gchar *accel_signal,
                                             GtkAccelGroup *accel_group,
                                             guint accel_key,
                                             GdkModifierType accel_mods,
                                             GtkAccelFlags accel_flags);
```

Installs an accelerator for this *widget* in *accel_group* that causes *accel_signal* to be emitted if the accelerator is activated. The *accel_group* needs to be added to the widget's toplevel via gtk_window_add_accel_group(), and the signal must be of type G_RUN_ACTION. Accelerators added through this function are not user changeable during runtime. If you want to support accelerators that can be changed by the user, use gtk_accel_map_add_entry() and gtk_widget_set_accel_path() or gtk_menu_item_set_accel_path() instead.

widget : widget to install an accelerator on
accel_signal : widget signal to emit on accelerator activation
accel_group : accel group for this widget, added to its toplevel
accel_key : GDK keyval of the accelerator
accel_mods : modifier key combination of the accelerator
accel_flags : flag accelerators, e.g. GTK_ACCEL_VISIBLE

---

gtk_widget_remove_accelerator ()

```
gboolean    gtk_widget_remove_accelerator   (GtkWidget *widget,
                                             GtkAccelGroup *accel_group,
                                             guint accel_key,
```

```
                                                GdkModifierType accel_mods);
```

Removes an accelerator from *widget*, previously installed with gtk_widget_add_accelerator().

widget :	widget to install an accelerator on
accel_group :	accel group for this widget
accel_key :	GDK keyval of the accelerator
accel_mods :	modifier key combination of the accelerator
Returns :	whether an accelerator was installed and could be removed

---

gtk_widget_set_accel_path ()

```
void        gtk_widget_set_accel_path       (GtkWidget *widget,
                                             const gchar *accel_path,
                                             GtkAccelGroup *accel_group);
```

Given an accelerator group, *accel_group*, and an accelerator path, *accel_path*, sets up an accelerator in *accel_group* so whenever the key binding that is defined for *accel_path* is pressed, *widget* will be activated. This removes any accelerators (for any accelerator group) installed by previous calls to gtk_widget_set_accel_path(). Associating accelerators with paths allows them to be modified by the user and the modifications to be saved for future use. (See gtk_accel_map_save().)

This function is a low level function that would most likely be used by a menu creation system like GtkItemFactory. If you use GtkItemFactory, setting up accelerator paths will be done automatically.

Even when you you aren't using GtkItemFactory, if you only want to set up accelerators on menu items gtk_menu_item_set_accel_path() provides a somewhat more convenient interface.

widget :	a GtkWidget
accel_path :	path used to look up the the accelerator
accel_group :	a GtkAccelGroup.

---

gtk_widget_list_accel_closures ()

```
GList*      gtk_widget_list_accel_closures  (GtkWidget *widget);
```

Lists the closures used by *widget* for accelerator group connections with gtk_accel_group_connect_by_path() or gtk_accel_group_connect(). The closures can be used to monitor accelerator changes on *widget*, by connecting to the ::accel_changed signal of the GtkAccelGroup of a closure which can be found out with gtk_accel_group_from_accel_closure().

widget : widget to list accelerator closures for

Returns : a newly allocated GList of closures

---

gtk_widget_can_activate_accel ()

```
gboolean    gtk_widget_can_activate_accel   (GtkWidget *widget,
                                             guint signal_id);
```

Determines whether an accelerator that activates the signal identified by *signal_id* can currently be activated. This is done by emitting the GtkWidget::can-activate-accel signal on *widget*; if the signal isn't overridden by a handler or in a derived widget, then the default check is that the widget must be sensitive, and the widget and all its ancestors mapped.

widget : a GtkWidget

signal_id : the ID of a signal installed on *widget*

Returns : TRUE if the accelerator can be activated.

Since 2.4

---

gtk_widget_event ()

```
gboolean    gtk_widget_event                (GtkWidget *widget,
                                             GdkEvent *event);
```

Rarely-used function. This function is used to emit the event signals on a widget (those signals should never be emitted without using this function to do so). If you want to synthesize an event though, don't use this function; instead, use `gtk_main_do_event()` so the event will behave as if it were in the event queue. Don't synthesize expose events; instead, use `gdk_window_invalidate_rect()` to invalidate a region of the window.

widget : a GtkWidget

event : a GdkEvent

Returns : return from the event signal emission (TRUE if the event was handled)

---

gtk_widget_activate ()

```
gboolean    gtk_widget_activate             (GtkWidget *widget);
```

For widgets that can be "activated" (buttons, menu items, etc.) this function activates them. Activation is what happens when you

press Enter on a widget during key navigation. If *widget* isn't activatable, the function returns FALSE.

widget : a GtkWidget that's activatable
Returns : TRUE if the widget was activatable

---

gtk_widget_reparent ()

```
void        gtk_widget_reparent             (GtkWidget *widget,
                                             GtkWidget *new_parent);
```

Moves a widget from one GtkContainer to another, handling reference count issues to avoid destroying the widget.

widget : a GtkWidget
new_parent : a GtkContainer to move the widget into

---

gtk_widget_intersect ()

```
gboolean    gtk_widget_intersect            (GtkWidget *widget,
                                             GdkRectangle *area,
                                             GdkRectangle *intersection);
```

Computes the intersection of a *widget*'s area and *area*, storing the intersection in *intersection*, and returns TRUE if there was an intersection. *intersection* may be NULL if you're only interested in whether there was an intersection.

widget : a GtkWidget
area : a rectangle
intersection : rectangle to store intersection of *widget* and *area*
Returns : TRUE if there was an intersection

---

gtk_widget_is_focus ()

```
gboolean    gtk_widget_is_focus             (GtkWidget *widget);
```

Determines if the widget is the focus widget within its toplevel. (This does not mean that the HAS_FOCUS flag is necessarily set; HAS_FOCUS will only be set if the toplevel widget additionally has the global input focus.)

widget : a GtkWidget

Returns : TRUE if the widget is the focus widget.

---

gtk_widget_grab_focus ()

```
void        gtk_widget_grab_focus           (GtkWidget *widget);
```

Causes *widget* to have the keyboard focus for the GtkWindow it's inside. *widget* must be a focusable widget, such as a GtkEntry; something like GtkFrame won't work. (More precisely, it must have the GTK_CAN_FOCUS flag set.)

widget : a GtkWidget

---

gtk_widget_grab_default ()

```
void        gtk_widget_grab_default         (GtkWidget *widget);
```

Causes *widget* to become the default widget. *widget* must have the GTK_CAN_DEFAULT flag set; typically you have to set this flag yourself by calling GTK_WIDGET_SET_FLAGS (*widget*, GTK_CAN_DEFAULT). The default widget is activated when the user presses Enter in a window. Default widgets must be activatable, that is, gtk_widget_activate() should affect them.

widget : a GtkWidget

---

gtk_widget_set_name ()

```
void        gtk_widget_set_name             (GtkWidget *widget,
                                             const gchar *name);
```

Widgets can be named, which allows you to refer to them from a gtkrc file. You can apply a style to widgets with a particular name in the gtkrc file. See the documentation for gtkrc files (on the same page as the docs for GtkRcStyle).

Note that widget names are separated by periods in paths (see gtk_widget_path()), so names with embedded periods may cause confusion.

widget : a GtkWidget
name : name for the widget

---

gtk_widget_get_name ()

```
G_CONST_RETURN gchar* gtk_widget_get_name   (GtkWidget *widget);
```

Retrieves the name of a widget. See `gtk_widget_set_name()` for the significance of widget names.

widget : a GtkWidget

Returns : name of the widget. This string is owned by GTK+ and should not be modified or freed

---

gtk_widget_set_state ()

```
void        gtk_widget_set_state            (GtkWidget *widget,
                                             GtkStateType state);
```

This function is for use in widget implementations. Sets the state of a widget (insensitive, prelighted, etc.) Usually you should set the state using wrapper functions such as `gtk_widget_set_sensitive()`.

widget : a GtkWidget

state : new state for *widget*

---

gtk_widget_set_sensitive ()

```
void        gtk_widget_set_sensitive        (GtkWidget *widget,
                                             gboolean sensitive);
```

Sets the sensitivity of a widget. A widget is sensitive if the user can interact with it. Insensitive widgets are "grayed out" and the user can't interact with them. Insensitive widgets are known as "inactive", "disabled", or "ghosted" in some other toolkits.

widget : a GtkWidget

sensitive : TRUE to make the widget sensitive

---

gtk_widget_set_parent ()

```
void        gtk_widget_set_parent           (GtkWidget *widget,
                                             GtkWidget *parent);
```

This function is useful only when implementing subclasses of GtkContainer. Sets the container as the parent of *widget*, and takes care of some details such as updating the state and style of the child to reflect its new location. The opposite function is `gtk_widget_unparent()`.

widget : a GtkWidget
parent : parent container

---

gtk_widget_set_parent_window ()

```
void        gtk_widget_set_parent_window    (GtkWidget *widget,
                                             GdkWindow *parent_window);
```

Sets a non default parent window for *widget*.

widget : a GtkWidget.
parent_window : the new parent window.

---

gtk_widget_get_parent_window ()

```
GdkWindow*  gtk_widget_get_parent_window    (GtkWidget *widget);
```

Gets *widget*'s parent window.

widget : a GtkWidget.
Returns : the parent window of *widget*.

---

gtk_widget_set_uposition ()

```
void        gtk_widget_set_uposition        (GtkWidget *widget,
                                             gint x,
                                             gint y);
```

Warning

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

Sets the position of a widget. The funny "u" in the name comes from the "user position" hint specified by the X Window System, and exists for legacy reasons. This function doesn't work if a widget is inside a container; it's only really useful on GtkWindow.

Don't use this function to center dialogs over the main application window; most window managers will do the centering on your behalf if you call `gtk_window_set_transient_for()`, and it's really not possible to get the centering to work correctly

in all cases from application code. But if you insist, use `gtk_window_set_position()` to set GTK_WIN_POS_CENTER_ON_PARENT, don't do the centering manually.

Note that although *x* and *y* can be individually unset, the position is not honoured unless both *x* and *y* are set.

widget : a GtkWidget
x : x position; -1 to unset x; -2 to leave x unchanged
y : y position; -1 to unset y; -2 to leave y unchanged

---

gtk_widget_set_usize ()

```
void        gtk_widget_set_usize            (GtkWidget *widget,
                                             gint width,
                                             gint height);
```

Warning

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

Sets the minimum size of a widget; that is, the widget's size request will be *width* by *height*. You can use this function to force a widget to be either larger or smaller than it is. The strange "usize" name dates from the early days of GTK+, and derives from X Window System terminology. In many cases, `gtk_window_set_default_size()` is a better choice for toplevel windows than this function; setting the default size will still allow users to shrink the window. Setting the usize will force them to leave the window at least as large as the usize. When dealing with window sizes, `gtk_window_set_geometry_hints()` can be a useful function as well.

Note the inherent danger of setting any fixed size - themes, translations into other languages, different fonts, and user action can all change the appropriate size for a given widget. So, it's basically impossible to hardcode a size that will always be correct.

Deprecated: Use `gtk_widget_set_size_request()` instead.

widget : a GtkWidget
width : minimum width, or -1 to unset
height : minimum height, or -1 to unset

---

gtk_widget_set_events ()

```
void        gtk_widget_set_events           (GtkWidget *widget,
                                             gint events);
```

Sets the event mask (see GdkEventMask) for a widget. The event mask determines which events a widget will receive. Keep in mind that different widgets have different default event masks, and by changing the event mask you may disrupt a widget's

functionality, so be careful. This function must be called while a widget is unrealized. Consider `gtk_widget_add_events()` for widgets that are already realized, or if you want to preserve the existing event mask. This function can't be used with GTK_NO_WINDOW widgets; to get events on those widgets, place them inside a GtkEventBox and receive events on the event box.

widget : a GtkWidget
events : event mask

---

gtk_widget_add_events ()

```
void        gtk_widget_add_events           (GtkWidget *widget,
                                             gint events);
```

Adds the events in the bitfield *events* to the event mask for *widget*. See `gtk_widget_set_events()` for details.

widget : a GtkWidget
events : an event mask, see GdkEventMask

---

gtk_widget_set_extension_events ()

```
void        gtk_widget_set_extension_events (GtkWidget *widget,
                                             GdkExtensionMode mode);
```

Sets the extension events mask to *mode*. See GdkExtensionMode and `gdk_input_set_extension_events()`.

widget : a GtkWidget
mode : bitfield of extension events to receive

---

gtk_widget_get_extension_events ()

```
GdkExtensionMode gtk_widget_get_extension_events
                                             (GtkWidget *widget);
```

Retrieves the extension events the widget will receive; see `gdk_input_set_extension_events()`.

widget : a GtkWidget
Returns : extension events for *widget*

---

gtk_widget_get_toplevel ()

```
GtkWidget*  gtk_widget_get_toplevel         (GtkWidget *widget);
```

This function returns the topmost widget in the container hierarchy *widget* is a part of. If *widget* has no parent widgets, it will be returned as the topmost widget. No reference will be added to the returned widget; it should not be unreferenced.

Note the difference in behavior vs. gtk_widget_get_ancestor(); gtk_widget_get_ancestor (widget, GTK_TYPE_WINDOW) would return NULL if *widget* wasn't inside a toplevel window, and if the window was inside a GtkWindow-derived widget which was in turn inside the toplevel GtkWindow. While the second case may seem unlikely, it actually happens when a GtkPlug is embedded inside a GtkSocket within the same application.

To reliably find the toplevel GtkWindow, use gtk_widget_get_toplevel() and check if the TOPLEVEL flags is set on the result.

```
GtkWidget *toplevel = gtk_widget_get_toplevel (widget);
if (GTK_WIDGET_TOPLEVEL (toplevel))
  {
    [ Perform action on toplevel. ]
  }
```

widget : a GtkWidget

Returns : the topmost ancestor of *widget*, or *widget* itself if there's no ancestor.

---

gtk_widget_get_ancestor ()

```
GtkWidget*  gtk_widget_get_ancestor         (GtkWidget *widget,
                                             GType widget_type);
```

Gets the first ancestor of *widget* with type *widget_type*. For example, gtk_widget_get_ancestor (widget, GTK_TYPE_BOX) gets the first GtkBox that's an ancestor of *widget*. No reference will be added to the returned widget; it should not be unreferenced. See note about checking for a toplevel GtkWindow in the docs for gtk_widget_get_toplevel().

Note that unlike gtk_widget_is_ancestor(), gtk_widget_get_ancestor() considers *widget* to be an ancestor of itself.

widget : a GtkWidget

widget_type : ancestor type

Returns : the ancestor widget, or NULL if not found

---

gtk_widget_get_colormap ()

```
GdkColormap* gtk_widget_get_colormap        (GtkWidget *widget);
```

Gets the colormap that will be used to render *widget*. No reference will be added to the returned colormap; it should not be unreferenced.

widget : a GtkWidget

Returns : the colormap used by *widget*

---

gtk_widget_set_colormap ()

```
void        gtk_widget_set_colormap         (GtkWidget *widget,
                                             GdkColormap *colormap);
```

Sets the colormap for the widget to the given value. Widget must not have been previously realized. This probably should only be used from an `init()` function (i.e. from the constructor for the widget).

widget : a GtkWidget

colormap : a colormap

---

gtk_widget_get_visual ()

```
GdkVisual*  gtk_widget_get_visual           (GtkWidget *widget);
```

Gets the visual that will be used to render *widget*.

widget : a GtkWidget

Returns : the visual for *widget*

---

gtk_widget_get_events ()

```
gint        gtk_widget_get_events           (GtkWidget *widget);
```

Returns the event mask for the widget (a bitfield containing flags from the GdkEventMask enumeration). These are the events that the widget will receive.

widget : a GtkWidget

Returns : event mask for *widget*

---

gtk_widget_get_pointer ()

```
void        gtk_widget_get_pointer          (GtkWidget *widget,
                                             gint *x,
                                             gint *y);
```

Obtains the location of the mouse pointer in widget coordinates. Widget coordinates are a bit odd; for historical reasons, they are defined as *widget->window* coordinates for widgets that are not GTK_NO_WINDOW widgets, and are relative to *widget->allocation.x*, *widget->allocation.y* for widgets that are GTK_NO_WINDOW widgets.

widget : a GtkWidget

x : return location for the X coordinate, or NULL

y : return location for the Y coordinate, or NULL

---

gtk_widget_is_ancestor ()

```
gboolean    gtk_widget_is_ancestor          (GtkWidget *widget,
                                             GtkWidget *ancestor);
```

Determines whether *widget* is somewhere inside *ancestor*, possibly with intermediate containers.

widget : a GtkWidget

ancestor : another GtkWidget

Returns : TRUE if *ancestor* contains *widget* as a child, grandchild, great grandchild, etc.

---

gtk_widget_translate_coordinates ()

```
gboolean    gtk_widget_translate_coordinates
                                            (GtkWidget *src_widget,
                                             GtkWidget *dest_widget,
                                             gint src_x,
                                             gint src_y,
                                             gint *dest_x,
                                             gint *dest_y);
```

Translate coordinates relative to *src_widget*'s allocation to coordinates relative to *dest_widget*'s allocations. In order to perform this operation, both widgets must be realized, and must share a common toplevel.

src_widget :	a GtkWidget
dest_widget :	a GtkWidget
src_x :	X position relative to *src_widget*
src_y :	Y position relative to *src_widget*
dest_x :	location to store X position relative to *dest_widget*
dest_y :	location to store Y position relative to *dest_widget*
Returns :	FALSE if either widget was not realized, or there was no common ancestor. In this case, nothing is stored in **dest_x* and **dest_y*. Otherwise TRUE.

---

gtk_widget_hide_on_delete ()

```
gboolean    gtk_widget_hide_on_delete       (GtkWidget *widget);
```

Utility function; intended to be connected to the "delete_event" signal on a GtkWindow. The function calls gtk_widget_hide() on its argument, then returns TRUE. If connected to "delete_event", the result is that clicking the close button for a window (on the window frame, top right corner usually) will hide but not destroy the window. By default, GTK+ destroys windows when "delete_event" is received.

widget :	a GtkWidget
Returns :	TRUE

---

gtk_widget_set_style ()

```
void        gtk_widget_set_style            (GtkWidget *widget,
                                             GtkStyle *style);
```

Sets the GtkStyle for a widget (*widget->style*). You probably don't want to use this function; it interacts badly with themes, because themes work by replacing the GtkStyle. Instead, use gtk_widget_modify_style().

widget :	a GtkWidget
style :	a GtkStyle, or NULL to remove the effect of a previous gtk_widget_set_style() and go back to the default style

---

gtk_widget_set_rc_style()

```
#define gtk_widget_set_rc_style(widget)          (gtk_widget_set_style (widget,
NULL))
```

Warning

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

Equivalent to `gtk_widget_set_style (widget, NULL)`.

`widget` : a GtkWidget.

---

gtk_widget_ensure_style ()

```
void        gtk_widget_ensure_style         (GtkWidget *widget);
```

Ensures that *`widget`* has a style (*`widget->style`*). Not a very useful function; most of the time, if you want the style, the widget is realized, and realized widgets are guaranteed to have a style already.

`widget` : a GtkWidget

---

gtk_widget_get_style ()

```
GtkStyle*   gtk_widget_get_style            (GtkWidget *widget);
```

Simply an accessor function that returns *`widget->style`*.

`widget` : a GtkWidget

Returns : the widget's GtkStyle

---

gtk_widget_restore_default_style()

```
#define gtk_widget_restore_default_style(widget) (gtk_widget_set_style (widget,
NULL))
```

Warning

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

Equivalent to `gtk_widget_set_style (widget, NULL)`.

widget : a GtkWidget.

---

gtk_widget_reset_rc_styles ()

```
void        gtk_widget_reset_rc_styles      (GtkWidget *widget);
```

Reset the styles of *widget* and all descendents, so when they are looked up again, they get the correct values for the currently loaded RC file settings.

This function is not useful for applications.

widget : a GtkWidget.

---

gtk_widget_push_colormap ()

```
void        gtk_widget_push_colormap        (GdkColormap *cmap);
```

Pushes *cmap* onto a global stack of colormaps; the topmost colormap on the stack will be used to create all widgets. Remove *cmap* with `gtk_widget_pop_colormap()`. There's little reason to use this function.

cmap : a GdkColormap

---

gtk_widget_pop_colormap ()

```
void        gtk_widget_pop_colormap         (void);
```

Removes a colormap pushed with `gtk_widget_push_colormap()`.

---

gtk_widget_set_default_colormap ()

```
void        gtk_widget_set_default_colormap (GdkColormap *colormap);
```

Sets the default colormap to use when creating widgets. `gtk_widget_push_colormap()` is a better function to use if you only want to affect a few widgets, rather than all widgets.

colormap : a GdkColormap

---

gtk_widget_get_default_style ()

```
GtkStyle*   gtk_widget_get_default_style    (void);
```

Returns the default style used by all widgets initially.

Returns : the default style. This GtkStyle object is owned by GTK+ and should not be modified or freed.

---

gtk_widget_get_default_colormap ()

```
GdkColormap* gtk_widget_get_default_colormap
                                            (void);
```

Obtains the default colormap used to create widgets.

Returns : default widget colormap

---

gtk_widget_get_default_visual ()

```
GdkVisual*  gtk_widget_get_default_visual   (void);
```

Obtains the visual of the default colormap. Not really useful; used to be useful before gdk_colormap_get_visual() existed.

Returns : visual of the default colormap

---

gtk_widget_set_direction ()

```
void        gtk_widget_set_direction        (GtkWidget *widget,
                                             GtkTextDirection dir);
```

Sets the reading direction on a particular widget. This direction controls the primary direction for widgets containing text, and also the direction in which the children of a container are packed. The ability to set the direction is present in order so that correct

localization into languages with right-to-left reading directions can be done. Generally, applications will let the default reading direction present, except for containers where the containers are arranged in an order that is explicitely visual rather than logical (such as buttons for text justification).

If the direction is set to GTK_TEXT_DIR_NONE, then the value set by gtk_widget_set_default_direction() will be used.

widget : a GtkWidget
dir : the new direction

---

enum GtkTextDirection

```
typedef enum
{
  GTK_TEXT_DIR_NONE,
  GTK_TEXT_DIR_LTR,
  GTK_TEXT_DIR_RTL
} GtkTextDirection;
```

---

gtk_widget_get_direction ()

```
GtkTextDirection gtk_widget_get_direction   (GtkWidget *widget);
```

Gets the reading direction for a particular widget. See gtk_widget_set_direction().

widget : a GtkWidget
Returns : the reading direction for the widget.

---

gtk_widget_set_default_direction ()

```
void        gtk_widget_set_default_direction
                                            (GtkTextDirection dir);
```

Sets the default reading direction for widgets where the direction has not been explicitly set by gtk_widget_set_direction().

dir : the new default direction. This cannot be GTK_TEXT_DIR_NONE.

---

gtk_widget_get_default_direction ()

```
GtkTextDirection gtk_widget_get_default_direction
                                                (void);
```

Obtains the current default reading direction. See gtk_widget_set_default_direction().

Returns : the current default direction.

---

gtk_widget_shape_combine_mask ()

```
void        gtk_widget_shape_combine_mask   (GtkWidget *widget,
                                             GdkBitmap *shape_mask,
                                             gint offset_x,
                                             gint offset_y);
```

Sets a shape for this widget's GDK window. This allows for transparent windows etc., see gdk_window_shape_combine_mask() for more information.

widget : a GtkWidget.
shape_mask : shape to be added, or NULL to remove an existing shape.
offset_x : X position of shape mask with respect to *window*.
offset_y : Y position of shape mask with respect to *window*.

---

gtk_widget_path ()

```
void        gtk_widget_path                 (GtkWidget *widget,
                                             guint *path_length,
                                             gchar **path,
                                             gchar **path_reversed);
```

Obtains the full path to *widget*. The path is simply the name of a widget and all its parents in the container hierarchy, separated by periods. The name of a widget comes from gtk_widget_get_name(). Paths are used to apply styles to a widget in gtkrc configuration files. Widget names are the type of the widget by default (e.g. "GtkButton") or can be set to an application-specific value with gtk_widget_set_name(). By setting the name of a widget, you allow users or theme authors to apply styles to that specific widget in their gtkrc file. *path_reversed_p* fills in the path in reverse order, i.e. starting with *widget*'s name instead of starting with the name of *widget*'s outermost ancestor.

widget : a GtkWidget

path_length : location to store length of the path, or NULL
path : location to store allocated path string, or NULL
path_reversed : location to store allocated reverse path string, or NULL

---

gtk_widget_class_path ()

```
void        gtk_widget_class_path           (GtkWidget *widget,
                                             guint *path_length,
                                             gchar **path,
                                             gchar **path_reversed);
```

Same as gtk_widget_path(), but always uses the name of a widget's type, never uses a custom name set with gtk_widget_set_name().

widget : a GtkWidget
path_length : location to store the length of the class path, or NULL
path : location to store the class path as an allocated string, or NULL
path_reversed : location to store the reverse class path as an allocated string, or NULL

---

gtk_widget_get_composite_name ()

```
gchar*      gtk_widget_get_composite_name   (GtkWidget *widget);
```

Obtains the composite name of a widget.

widget : a GtkWidget.
Returns : the composite name of *widget*, or NULL if *widget* is not a composite child. The string should not be freed when it is no longer needed.

---

gtk_widget_modify_style ()

```
void        gtk_widget_modify_style         (GtkWidget *widget,
                                             GtkRcStyle *style);
```

Modifies style values on the widget. Modifications made using this technique take precedence over style values set via an RC file, however, they will be overriden if a style is explicitely set on the widget using gtk_widget_set_style(). The GtkRcStyle structure is designed so each field can either be set or unset, so it is possible, using this function, to modify some style values and leave the others unchanged.

Note that modifications made with this function are not cumulative with previous calls to `gtk_widget_modify_style()` or with such functions as `gtk_widget_modify_fg()`. If you wish to retain previous values, you must first call `gtk_widget_get_modifier_style()`, make your modifications to the returned style, then call `gtk_widget_modify_style()` with that style. On the other hand, if you first call `gtk_widget_modify_style()`, subsequent calls to such functions `gtk_widget_modify_fg()` will have a cumulative effect with the initial modifications.

widget : a GtkWidget

style : the GtkRcStyle holding the style modifications

---

gtk_widget_get_modifier_style ()

```
GtkRcStyle* gtk_widget_get_modifier_style   (GtkWidget *widget);
```

Returns the current modifier style for the widget. (As set by `gtk_widget_modify_style()`.) If no style has previously set, a new GtkRcStyle will be created with all values unset, and set as the modifier style for the widget. If you make changes to this rc style, you must call `gtk_widget_modify_style()`, passing in the returned rc style, to make sure that your changes take effect.

Caution: passing the style back to `gtk_widget_modify_style()` will normally end up destroying it, because `gtk_widget_modify_style()` copies the passed-in style and sets the copy as the new modifier style, thus dropping any reference to the old modifier style. Add a reference to the modifier style if you want to keep it alive.

widget : a GtkWidget

Returns : the modifier style for the widget. This rc style is owned by the widget. If you want to keep a pointer to value this around, you must add a refcount using `gtk_rc_style_ref()`.

---

gtk_widget_modify_fg ()

```
void        gtk_widget_modify_fg            (GtkWidget *widget,
                                             GtkStateType state,
                                             const GdkColor *color);
```

Sets the foreground color for a widget in a particular state. All other style values are left untouched. See also `gtk_widget_modify_style()`.

widget : a GtkWidget.

state : the state for which to set the foreground color.

color : the color to assign (does not need to be allocated), or `NULL` to undo the effect of previous calls to of `gtk_widget_modify_fg()`.

---

gtk_widget_modify_bg ()

```
void        gtk_widget_modify_bg            (GtkWidget *widget,
                                             GtkStateType state,
                                             const GdkColor *color);
```

Sets the background color for a widget in a particular state. All other style values are left untouched. See also `gtk_widget_modify_style()`.

widget : a GtkWidget.

state : the state for which to set the background color.

color : the color to assign (does not need to be allocated), or `NULL` to undo the effect of previous calls to of `gtk_widget_modify_bg()`.

---

gtk_widget_modify_text ()

```
void        gtk_widget_modify_text          (GtkWidget *widget,
                                             GtkStateType state,
                                             const GdkColor *color);
```

Sets the text color for a widget in a particular state. All other style values are left untouched. The text color is the foreground color used along with the base color (see `gtk_widget_modify_base()`) for widgets such as GtkEntry and GtkTextView. See also `gtk_widget_modify_style()`.

widget : a GtkWidget.

state : the state for which to set the text color.

color : the color to assign (does not need to be allocated), or `NULL` to undo the effect of previous calls to of `gtk_widget_modify_text()`.

---

gtk_widget_modify_base ()

```
void        gtk_widget_modify_base          (GtkWidget *widget,
                                             GtkStateType state,
                                             const GdkColor *color);
```

Sets the base color for a widget in a particular state. All other style values are left untouched. The base color is the background color used along with the text color (see `gtk_widget_modify_text()`) for widgets such as GtkEntry and GtkTextView. See also `gtk_widget_modify_style()`.

widget : a GtkWidget.

state : the state for which to set the base color.

color : the color to assign (does not need to be allocated), or NULL to undo the effect of previous calls to of gtk_widget_modify_base().

---

gtk_widget_modify_font ()

```
void        gtk_widget_modify_font          (GtkWidget *widget,
                                             PangoFontDescription *font_desc);
```

Sets the font to use for a widget. All other style values are left untouched. See also gtk_widget_modify_style().

widget : a GtkWidget

font_desc : the font description to use, or NULL to undo the effect of previous calls to gtk_widget_modify_font().

---

gtk_widget_create_pango_context ()

```
PangoContext* gtk_widget_create_pango_context
                                            (GtkWidget *widget);
```

Creates a new PangoContext with the appropriate colormap, font description, and base direction for drawing text for this widget. See also gtk_widget_get_pango_context().

widget : a GtkWidget

Returns : the new PangoContext

---

gtk_widget_get_pango_context ()

```
PangoContext* gtk_widget_get_pango_context  (GtkWidget *widget);
```

Gets a PangoContext with the appropriate colormap, font description and base direction for this widget. Unlike the context returned by gtk_widget_create_pango_context(), this context is owned by the widget (it can be used until the screen for the widget changes or the widget is removed from its toplevel), and will be updated to match any changes to the widget's attributes.

If you create and keep a PangoLayout using this context, you must deal with changes to the context by calling pango_layout_context_changed() on the layout in response to the ::style-set and ::direction-changed signals for the

widget.

widget : a GtkWidget

Returns : the PangoContext for the widget.

---

gtk_widget_create_pango_layout ()

```
PangoLayout* gtk_widget_create_pango_layout (GtkWidget *widget,
                                             const gchar *text);
```

Creates a new PangoLayout with the appropriate colormap, font description, and base direction for drawing text for this widget.

If you keep a PangoLayout created in this way around, in order notify the layout of changes to the base direction or font of this widget, you must call `pango_layout_context_changed()` in response to the ::style-set and ::direction-changed signals for the widget.

widget : a GtkWidget

text : text to set on the layout (can be NULL)

Returns : the new PangoLayout

---

gtk_widget_render_icon ()

```
GdkPixbuf*  gtk_widget_render_icon          (GtkWidget *widget,
                                             const gchar *stock_id,
                                             GtkIconSize size,
                                             const gchar *detail);
```

A convenience function that uses the theme engine and RC file settings for *widget* to look up *stock_id* and render it to a pixbuf. *stock_id* should be a stock icon ID such as GTK_STOCK_OPEN or GTK_STOCK_OK. *size* should be a size such as GTK_ICON_SIZE_MENU. *detail* should be a string that identifies the widget or code doing the rendering, so that theme engines can special-case rendering for that widget or code.

The pixels in the returned GdkPixbuf are shared with the rest of the application and should not be modified. The pixbuf should be freed after use with `g_object_unref()`.

widget : a GtkWidget

stock_id : a stock ID

size : a stock size. A size of (GtkIconSize)-1 means render at the size of the source and don't scale (if there are multiple source sizes, GTK+ picks one of the available sizes).

detail : render detail to pass to theme engine

Returns : a new pixbuf, or `NULL` if the stock ID wasn't known

---

gtk_widget_pop_composite_child ()

```
void        gtk_widget_pop_composite_child  (void);
```

Cancels the effect of a previous call to `gtk_widget_push_composite_child()`.

---

gtk_widget_push_composite_child ()

```
void        gtk_widget_push_composite_child (void);
```

Makes all newly-created widgets as composite children until the corresponding `gtk_widget_pop_composite_child()` call.

A composite child is a child that's an implementation detail of the container it's inside and should not be visible to people using the container. Composite children aren't treated differently by GTK (but see `gtk_container_foreach()` vs. `gtk_container_forall()`), but e.g. GUI builders might want to treat them in a different way.

Here is a simple example:

```
  gtk_widget_push_composite_child ();
  scrolled_window->hscrollbar = gtk_hscrollbar_new (hadjustment);
  gtk_widget_set_composite_name (scrolled_window->hscrollbar, "hscrollbar");
  gtk_widget_pop_composite_child ();
  gtk_widget_set_parent (scrolled_window->hscrollbar,
                         GTK_WIDGET (scrolled_window));
  g_object_ref (scrolled_window->hscrollbar);
```

---

gtk_widget_queue_clear ()

```
void        gtk_widget_queue_clear          (GtkWidget *widget);
```

Warning

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

This function does the same as `gtk_widget_queue_draw()`.

Deprecated: Use gtk_widget_queue_draw() instead.

widget : a GtkWidget

---

gtk_widget_queue_clear_area ()

```
void        gtk_widget_queue_clear_area     (GtkWidget *widget,
                                             gint x,
                                             gint y,
                                             gint width,
                                             gint height);
```

Warning

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

This function is no longer different from gtk_widget_queue_draw_area(), though it once was. Now it just calls gtk_widget_queue_draw_area(). Originally gtk_widget_queue_clear_area() would force a redraw of the background for GTK_NO_WINDOW widgets, and gtk_widget_queue_draw_area() would not. Now both functions ensure the background will be redrawn.

Deprecated: Use gtk_widget_queue_draw_area() instead.

widget : a GtkWidget
x : x coordinate of upper-left corner of rectangle to redraw
y : y coordinate of upper-left corner of rectangle to redraw
width : width of region to draw
height : height of region to draw

---

gtk_widget_queue_draw_area ()

```
void        gtk_widget_queue_draw_area      (GtkWidget *widget,
                                             gint x,
                                             gint y,
                                             gint width,
                                             gint height);
```

Invalidates the rectangular area of *widget* defined by *x*, *y*, *width* and *height* by calling gdk_window_invalidate_rect() on the widget's window and all its child windows. Once the main loop becomes idle (after the current batch of events has been processed, roughly), the window will receive expose events for the union of all regions

that have been invalidated.

Normally you would only use this function in widget implementations. You might also use it, or `gdk_window_invalidate_rect()` directly, to schedule a redraw of a GtkDrawingArea or some portion thereof.

Frequently you can just call `gdk_window_invalidate_rect()` or `gdk_window_invalidate_region()` instead of this function. Those functions will invalidate only a single window, instead of the widget and all its children.

The advantage of adding to the invalidated region compared to simply drawing immediately is efficiency; using an invalid region ensures that you only have to redraw one time.

widget : a GtkWidget
x : x coordinate of upper-left corner of rectangle to redraw
y : y coordinate of upper-left corner of rectangle to redraw
width : width of region to draw
height : height of region to draw

---

gtk_widget_reset_shapes ()

```
void        gtk_widget_reset_shapes         (GtkWidget *widget);
```

Recursively resets the shape on this widget and its descendants.

widget : a GtkWidget.

---

gtk_widget_set_app_paintable ()

```
void        gtk_widget_set_app_paintable    (GtkWidget *widget,
                                             gboolean app_paintable);
```

widget :
app_paintable :

---

gtk_widget_set_double_buffered ()

```
void        gtk_widget_set_double_buffered  (GtkWidget *widget,
                                             gboolean double_buffered);
```

Widgets are double buffered by default; you can use this function to turn off the buffering. "Double buffered" simply means that `gdk_window_begin_paint_region()` and `gdk_window_end_paint()` are called automatically around expose events sent to the widget. `gdk_window_begin_paint()` diverts all drawing to a widget's window to an offscreen buffer, and `gdk_window_end_paint()` draws the buffer to the screen. The result is that users see the window update in one smooth step, and don't see individual graphics primitives being rendered.

In very simple terms, double buffered widgets don't flicker, so you would only use this function to turn off double buffering if you had special needs and really knew what you were doing.

widget : a GtkWidget

double_buffered : TRUE to double-buffer a widget

---

gtk_widget_set_redraw_on_allocate ()

```
void        gtk_widget_set_redraw_on_allocate
                                            (GtkWidget *widget,
                                             gboolean redraw_on_allocate);
```

Sets whether a when a widgets size allocation changes, the entire widget is queued for drawing. By default, this setting is TRUE and the entire widget is redrawn on every size change. If your widget leaves the upper left unchanged when made bigger, turning this setting on will improve performance.

Note that for NO_WINDOW widgets setting this flag to FALSE turns off all allocation on resizing: the widget will not even redraw if its position changes; this is to allow containers that don't draw anything to avoid excess invalidations. If you set this flag on a NO_WINDOW widget that *does* draw on *widget->window*, you are responsible for invalidating both the old and new allocation of the widget when the widget is moved and responsible for invalidating regions newly when the widget increases size.

widget : a GtkWidget

redraw_on_allocate : if TRUE, the entire widget will be redrawn when it is allocated to a new size. Otherwise, only the new portion of the widget will be redrawn.

---

gtk_widget_set_composite_name ()

```
void        gtk_widget_set_composite_name   (GtkWidget *widget,
                                             const gchar *name);
```

Sets a widgets composite name. The widget must be a composite child of its parent; see `gtk_widget_push_composite_child()`.

widget : a GtkWidget.

name : the name to set.

gtk_widget_set_scroll_adjustments ()

```
gboolean    gtk_widget_set_scroll_adjustments
                                            (GtkWidget *widget,
                                             GtkAdjustment *hadjustment,
                                             GtkAdjustment *vadjustment);
```

For widgets that support scrolling, sets the scroll adjustments and returns TRUE. For widgets that don't support scrolling, does nothing and returns FALSE. Widgets that don't support scrolling can be scrolled by placing them in a GtkViewport, which does support scrolling.

widget : a GtkWidget
hadjustment : an adjustment for horizontal scrolling, or NULL
vadjustment : an adjustment for vertical scrolling, or NULL
Returns : TRUE if the widget supports scrolling

gtk_widget_mnemonic_activate ()

```
gboolean    gtk_widget_mnemonic_activate    (GtkWidget *widget,
                                             gboolean group_cycling);
```

widget :
group_cycling :
Returns :

gtk_widget_class_install_style_property ()

```
void        gtk_widget_class_install_style_property
                                            (GtkWidgetClass *klass,
                                             GParamSpec *pspec);
```

Installs a style property on a widget class. The parser for the style property is determined by the value type of *pspec*.

klass : a GtkWidgetClass
pspec : the GParamSpec for the property

gtk_widget_class_install_style_property_parser ()

```
void        gtk_widget_class_install_style_property_parser
                                            (GtkWidgetClass *klass,
                                             GParamSpec *pspec,
                                             GtkRcPropertyParser parser);
```

Installs a style property on a widget class.

klass : a GtkWidgetClass

pspec : the GParamSpec for the style property

parser : the parser for the style property

---

gtk_widget_class_find_style_property ()

```
GParamSpec* gtk_widget_class_find_style_property
                                            (GtkWidgetClass *klass,
                                             const gchar *property_name);
```

Finds a style property of a widget class by name.

klass : a GtkWidgetClass

property_name : the name of the style property to find

Returns : the GParamSpec of the style property or `NULL` if *class* has no style property with that name.

Since 2.2

---

gtk_widget_class_list_style_properties ()

```
GParamSpec** gtk_widget_class_list_style_properties
                                            (GtkWidgetClass *klass,
                                             guint *n_properties);
```

Returns all style properties of a widget class.

klass : a GtkWidgetClass

n_properties : location to return the number of style properties found

Returns : an newly allocated array of GParamSpec*. The array must be freed with `g_free()`.

Since 2.2

---

gtk_widget_region_intersect ()

```
GdkRegion*  gtk_widget_region_intersect     (GtkWidget *widget,
                                             GdkRegion *region);
```

Computes the intersection of a *`widget`*'s area and *`region`*, returning the intersection. The result may be empty, use `gdk_region_empty()` to check.

`widget` : a GtkWidget

`region` : a GdkRegion, in the same coordinate system as *`widget->allocation`*. That is, relative to *`widget->window`* for `NO_WINDOW` widgets; relative to the parent window of *`widget->window`* for widgets with their own window.

Returns : A newly allocated region holding the intersection of *`widget`* and *`region`*. The coordinates of the return value are relative to *`widget->window`* for `NO_WINDOW` widgets, and relative to the parent window of *`widget->window`* for widgets with their own window.

---

gtk_widget_send_expose ()

```
gint        gtk_widget_send_expose          (GtkWidget *widget,
                                             GdkEvent *event);
```

Very rarely-used function. This function is used to emit an expose event signals on a widget. This function is not normally used directly. The only time it is used is when propagating an expose event to a child `NO_WINDOW` widget, and that is normally done using `gtk_container_propagate_expose()`.

If you want to force an area of a window to be redrawn, use `gdk_window_invalidate_rect()` or `gdk_window_invalidate_region()`. To cause the redraw to be done immediately, follow that call with a call to `gdk_window_process_updates()`.

`widget` : a GtkWidget

`event` : a expose GdkEvent

Returns : return from the event signal emission (`TRUE` if the event was handled)

---

gtk_widget_style_get ()

```
void        gtk_widget_style_get            (GtkWidget *widget,
                                             const gchar *first_property_name,
                                             ...);
```

Gets the values of a multiple style properties of *widget*.

widget :	a GtkWidget
first_property_name :	the name of the first property to get
... :	pairs of property names and locations to return the property values, starting with the location for *first_property_name*, terminated by NULL.

---

gtk_widget_style_get_property ()

```
void        gtk_widget_style_get_property   (GtkWidget *widget,
                                             const gchar *property_name,
                                             GValue *value);
```

Gets the value of a style property of *widget*.

widget :	a GtkWidget
property_name :	the name of a style property
value :	location to return the property value

---

gtk_widget_style_get_valist ()

```
void        gtk_widget_style_get_valist     (GtkWidget *widget,
                                             const gchar *first_property_name,
                                             va_list var_args);
```

Non-vararg variant of gtk_widget_style_get(). Used primarily by language bindings.

widget :	a GtkWidget
first_property_name :	the name of the first property to get
var_args :	a va_list of pairs of property names and locations to return the property values, starting with the location for *first_property_name*.

---

gtk_widget_get_accessible ()

```
AtkObject*  gtk_widget_get_accessible       (GtkWidget *widget);
```

widget :

Returns :

---

gtk_widget_child_focus ()

```
gboolean    gtk_widget_child_focus          (GtkWidget *widget,
                                             GtkDirectionType direction);
```

This function is used by custom widget implementations; if you're writing an app, you'd use `gtk_widget_grab_focus()` to move the focus to a particular widget, and `gtk_container_set_focus_chain()` to change the focus tab order. So you may want to investigate those functions instead.

`gtk_widget_child_focus()` is called by containers as the user moves around the window using keyboard shortcuts. *direction* indicates what kind of motion is taking place (up, down, left, right, tab forward, tab backward). `gtk_widget_child_focus()` invokes the "focus" signal on GtkWidget; widgets override the default handler for this signal in order to implement appropriate focus behavior.

The "focus" default handler for a widget should return TRUE if moving in *direction* left the focus on a focusable location inside that widget, and FALSE if moving in *direction* moved the focus outside the widget. If returning TRUE, widgets normally call `gtk_widget_grab_focus()` to place the focus accordingly; if returning FALSE, they don't modify the current focus location.

This function replaces `gtk_container_focus()` from GTK+ 1.2. It was necessary to check that the child was visible, sensitive, and focusable before calling `gtk_container_focus()`. `gtk_widget_child_focus()` returns FALSE if the widget is not currently in a focusable state, so there's no need for those checks.

widget : a GtkWidget

direction : direction of focus movement

Returns : TRUE if focus ended up inside *widget*

---

gtk_widget_child_notify ()

```
void        gtk_widget_child_notify         (GtkWidget *widget,
                                             const gchar *child_property);
```

Emits a "child-notify" signal for the child property *child_property* on *widget*.

This is the analogue of `g_object_notify()` for child properties.

widget : a GtkWidget
child_property : the name of a child property installed on the class of *widget*'s parent.

---

gtk_widget_freeze_child_notify ()

```
void        gtk_widget_freeze_child_notify  (GtkWidget *widget);
```

Stops emission of "child-notify" signals on *widget*. The signals are queued until gtk_widget_thaw_child_notify() is called on *widget*.

This is the analogue of g_object_freeze_notify() for child properties.

widget : a GtkWidget

---

gtk_widget_get_child_visible ()

```
gboolean    gtk_widget_get_child_visible    (GtkWidget *widget);
```

Gets the value set with gtk_widget_set_child_visible(). If you feel a need to use this function, your code probably needs reorganization.

This function is only useful for container implementations and never should be called by an application.

widget : a GtkWidget
Returns : TRUE if the widget is mapped with the parent.

---

gtk_widget_get_parent ()

```
GtkWidget*  gtk_widget_get_parent           (GtkWidget *widget);
```

Returns the parent container of *widget*.

widget : a GtkWidget
Returns : the parent container of *widget*, or NULL

---

gtk_widget_get_settings ()

```
GtkSettings* gtk_widget_get_settings        (GtkWidget *widget);
```

Gets the settings object holding the settings (global property settings, RC file information, etc) used for this widget.

Note that this function can only be called when the GtkWidget is attached to a toplevel, since the settings object is specific to a particular GdkScreen.

widget : a GtkWidget

Returns : the relevant GtkSettings object

---

gtk_widget_get_clipboard ()

```
GtkClipboard* gtk_widget_get_clipboard      (GtkWidget *widget,
                                             GdkAtom selection);
```

Returns the clipboard object for the given selection to be used with *widget*. *widget* must have a GdkDisplay associated with it, so must be attached to a toplevel window.

widget : a GtkWidget

selection : a GdkAtom which identifies the clipboard to use. GDK_SELECTION_CLIPBOARD gives the default clipboard. Another common value is GDK_SELECTION_PRIMARY, which gives the primary X selection.

Returns : the appropriate clipboard object. If no clipboard already exists, a new one will be created. Once a clipboard object has been created, it is persistent for all time.

Since 2.2

---

gtk_widget_get_display ()

```
GdkDisplay* gtk_widget_get_display          (GtkWidget *widget);
```

Get the GdkDisplay for the toplevel window associated with this widget. This function can only be called after the widget has been added to a widget hierarchy with a GtkWindow at the top.

In general, you should only create display specific resources when a widget has been realized, and you should free those resources when the widget is unrealized.

widget : a GtkWidget

Returns : the GdkDisplay for the toplevel for this widget.

Since 2.2

---

gtk_widget_get_root_window ()

```
GdkWindow*  gtk_widget_get_root_window      (GtkWidget *widget);
```

Get the root window where this widget is located. This function can only be called after the widget has been added to a widget heirarchy with GtkWindow at the top.

The root window is useful for such purposes as creating a popup GdkWindow associated with the window. In general, you should only create display specific resources when a widget has been realized, and you should free those resources when the widget is unrealized.

widget : a GtkWidget

Returns : the GdkWindow root window for the toplevel for this widget.

Since 2.2

---

gtk_widget_get_screen ()

```
GdkScreen*  gtk_widget_get_screen           (GtkWidget *widget);
```

Get the GdkScreen from the toplevel window associated with this widget. This function can only be called after the widget has been added to a widget hierarchy with a GtkWindow at the top.

In general, you should only create screen specific resources when a widget has been realized, and you should free those resources when the widget is unrealized.

widget : a GtkWidget

Returns : the GdkScreen for the toplevel for this widget.

Since 2.2

---

gtk_widget_has_screen ()

```
gboolean    gtk_widget_has_screen           (GtkWidget *widget);
```

Checks whether there is a GdkScreen is associated with this widget. All toplevel widgets have an associated screen, and all widgets added into a heirarchy with a toplevel window at the top.

widget : a GtkWidget

Returns : TRUE if there is a GdkScreen associcated with the widget.

Since 2.2

---

gtk_widget_get_size_request ()

```
void        gtk_widget_get_size_request     (GtkWidget *widget,
                                             gint *width,
                                             gint *height);
```

Gets the size request that was explicitly set for the widget using `gtk_widget_set_size_request()`. A value of -1 stored in *width* or *height* indicates that that dimension has not been set explicitly and the natural requisition of the widget will be used intead. See `gtk_widget_set_size_request()`. To get the size a widget will actually use, call `gtk_widget_size_request()` instead of this function.

widget : a GtkWidget

width : return location for width, or NULL

height : return location for height, or NULL

---

gtk_widget_pop_visual()

```
#define gtk_widget_pop_visual()              ((void) 0)
```

Warning

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

This function is deprecated; it does nothing.

---

gtk_widget_push_visual()

```
#define gtk_widget_push_visual(visual)       ((void) 0)
```

Warning

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

This function is deprecated; it does nothing.

visual :

---

gtk_widget_set_child_visible ()

```
void        gtk_widget_set_child_visible    (GtkWidget *widget,
                                             gboolean is_visible);
```

Sets whether *widget* should be mapped along with its when its parent is mapped and *widget* has been shown with `gtk_widget_show()`.

The child visibility can be set for widget before it is added to a container with `gtk_widget_set_parent()`, to avoid mapping children unnecessary before immediately unmapping them. However it will be reset to its default state of TRUE when the widget is removed from a container.

Note that changing the child visibility of a widget does not queue a resize on the widget. Most of the time, the size of a widget is computed from all visible children, whether or not they are mapped. If this is not the case, the container can queue a resize itself.

This function is only useful for container implementations and never should be called by an application.

widget : a GtkWidget

is_visible : if TRUE, *widget* should be mapped along with its parent.

---

gtk_widget_set_default_visual()

```
#define gtk_widget_set_default_visual(visual) ((void) 0)
```

Warning

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

This function is deprecated; it does nothing.

visual :

---

gtk_widget_set_size_request ()

```
void        gtk_widget_set_size_request     (GtkWidget *widget,
                                             gint width,
                                             gint height);
```

Sets the minimum size of a widget; that is, the widget's size request will be *width* by *height*. You can use this function to force a widget to be either larger or smaller than it normally would be.

In most cases, gtk_window_set_default_size() is a better choice for toplevel windows than this function; setting the default size will still allow users to shrink the window. Setting the size request will force them to leave the window at least as large as the size request. When dealing with window sizes, gtk_window_set_geometry_hints() can be a useful function as well.

Note the inherent danger of setting any fixed size - themes, translations into other languages, different fonts, and user action can all change the appropriate size for a given widget. So, it's basically impossible to hardcode a size that will always be correct.

The size request of a widget is the smallest size a widget can accept while still functioning well and drawing itself correctly. However in some strange cases a widget may be allocated less than its requested size, and in many cases a widget may be allocated more space than it requested.

If the size request in a given direction is -1 (unset), then the "natural" size request of the widget will be used instead.

Widgets can't actually be allocated a size less than 1 by 1, but you can pass 0,0 to this function to mean "as small as possible."

widget : a GtkWidget
width : width *widget* should request, or -1 to unset
height : height *widget* should request, or -1 to unset

---

gtk_widget_set_visual()

```
#define gtk_widget_set_visual(widget,visual)  ((void) 0)
```

Warning

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

This function is deprecated; it does nothing.

widget :
visual :

---

gtk_widget_thaw_child_notify ()

```
void        gtk_widget_thaw_child_notify    (GtkWidget *widget);
```

Reverts the effect of a previous call to `gtk_widget_freeze_child_notify()`. This causes all queued "child-notify" signals on *`widget`* to be emitted.

`widget` : a GtkWidget

---

gtk_widget_set_no_show_all ()

```
void        gtk_widget_set_no_show_all      (GtkWidget *widget,
                                             gboolean no_show_all);
```

Sets the "no_show_all" property, which determines whether calls to `gtk_widget_show_all()` and `gtk_widget_hide_all()` will affect this widget.

This is mostly for use in constructing widget hierarchies with externally controlled visibility, see GtkUIManager.

`widget` : a GtkWidget
`no_show_all` : the new value for the "no_show_all" property

Since 2.4

---

gtk_widget_get_no_show_all ()

```
gboolean    gtk_widget_get_no_show_all      (GtkWidget *widget);
```

Returns the current value of the "no_show_all" property, which determines whether calls to `gtk_widget_show_all()` and `gtk_widget_hide_all()` will affect this widget.

`widget` : a GtkWidget
Returns : the current value of the "no_show_all" property.

Since 2.4

---

gtk_widget_list_mnemonic_labels ()

```
GList*      gtk_widget_list_mnemonic_labels (GtkWidget *widget);
```

Returns a newly allocated list of the widgets, normally labels, for which this widget is a the target of a mnemonic (see for example, `gtk_label_set_mnemonic_widget()`).

The widgets in the list are not individually referenced. If you want to iterate through the list and perform actions involving callbacks that might destroy the widgets, you *must* call `g_list_foreach (result, (GFunc)g_object_ref, NULL)` first, and then unref all the widgets afterwards.

widget : a GtkWidget

Returns : the list of mnemonic labels; free this list with `g_list_free()` when you are done with it.

Since 2.4

---

gtk_widget_add_mnemonic_label ()

```
void        gtk_widget_add_mnemonic_label   (GtkWidget *widget,
                                             GtkWidget *label);
```

Adds a widget to the list of mnemonic labels for this widget. (See `gtk_widget_list_mnemonic_labels()`). Note the list of mnemonic labels for the widget is cleared when the widget is destroyed, so the caller must make sure to update it's internal state at this point as well, by using a connection to the ::destroy signal or a weak notifier.

widget : a GtkWidget

label : a GtkWidget that acts as a mnemonic label for *widget*.

Since 2.4

---

gtk_widget_remove_mnemonic_label ()

```
void        gtk_widget_remove_mnemonic_label
                                            (GtkWidget *widget,
                                             GtkWidget *label);
```

Removes a widget from the list of mnemonic labels for this widget. (See `gtk_widget_list_mnemonic_labels()`). The widget must have previously been added to the list with `gtk_widget_add_mnemonic_label()`.

widget : a GtkWidget

label : a GtkWidget that was previously set as a mnemnic label for *widget* with gtk_widget_add_mnemonic_label().

Since 2.4

---

gtk_requisition_copy ()

```
GtkRequisition* gtk_requisition_copy        (const GtkRequisition *requisition);
```

Copies a GtkRequisition.

requisition : a GtkRequisition.

Returns : a copy of *requisition*.

---

gtk_requisition_free ()

```
void        gtk_requisition_free            (GtkRequisition *requisition);
```

Frees a GtkRequisition.

requisition : a GtkRequisition.

Properties

The "app-paintable" property

```
  "app-paintable"        gboolean              : Read / Write
```

Whether the application will paint directly on the widget.

Default value: FALSE

The "can-default" property

```
"can-default"          gboolean              : Read / Write
```

Whether the widget can be the default widget.

Default value: FALSE

The "can-focus" property

```
"can-focus"            gboolean              : Read / Write
```

Whether the widget can accept the input focus.

Default value: FALSE

The "composite-child" property

```
"composite-child"      gboolean              : Read
```

Whether the widget is part of a composite widget.

Default value: FALSE

The "events" property

```
"events"               GdkEventMask          : Read / Write
```

The event mask that decides what kind of GdkEvents this widget gets.

Default value: GDK_STRUCTURE_MASK

The "extension-events" property

```
"extension-events"     GdkExtensionMode      : Read / Write
```

The mask that decides what kind of extension events this widget gets.

Default value: GDK_EXTENSION_EVENTS_NONE

The "has-default" property

```
"has-default"          gboolean              : Read / Write
```

Whether the widget is the default widget.

Default value: FALSE

The "has-focus" property

```
"has-focus"            gboolean              : Read / Write
```

Whether the widget has the input focus.

Default value: FALSE

The "height-request" property

```
"height-request"       gint                  : Read / Write
```

Override for height request of the widget, or -1 if natural request should be used.

Allowed values: >= -1

Default value: -1

The "is-focus" property

```
"is-focus"             gboolean              : Read / Write
```

Whether the widget is the focus widget within the toplevel.

Default value: FALSE

The "name" property

"name"	gchararray	: Read / Write

The name of the widget.

Default value: NULL

The "no-show-all" property

"no-show-all"	gboolean	: Read / Write

Whether gtk_widget_show_all() should not affect this widget.

Default value: FALSE

The "parent" property

"parent"	GtkContainer	: Read / Write

The parent widget of this widget. Must be a Container widget.

The "receives-default" property

"receives-default"	gboolean	: Read / Write

If TRUE, the widget will receive the default action when it is focused.

Default value: FALSE

The "sensitive" property

```
"sensitive"            gboolean              : Read / Write
```

Whether the widget responds to input.

Default value: TRUE

The "style" property

```
"style"                GtkStyle              : Read / Write
```

The style of the widget, which contains information about how it will look (colors etc).

The "visible" property

```
"visible"              gboolean              : Read / Write
```

Whether the widget is visible.

Default value: FALSE

The "width-request" property

```
"width-request"        gint                  : Read / Write
```

Override for width request of the widget, or -1 if natural request should be used.

Allowed values: >= -1

Default value: -1

Style Properties

The "cursor-aspect-ratio" style property

```
"cursor-aspect-ratio"  gfloat                : Read
```

Aspect ratio with which to draw insertion cursor.

Allowed values: [0,1]

Default value: 0.04

The "cursor-color" style property

```
"cursor-color"         GdkColor              : Read
```

Color with which to draw insertion cursor.

The "focus-line-pattern" style property

```
"focus-line-pattern"   gchararray            : Read
```

Dash pattern used to draw the focus indicator.

Default value: "\001\001"

The "focus-line-width" style property

```
"focus-line-width"     gint                  : Read
```

Width, in pixels, of the focus indicator line.

Allowed values: >= 0

Default value: 1

The "focus-padding" style property

```
"focus-padding"        gint                  : Read
```

Width, in pixels, between focus indicator and the widget 'box'.

Allowed values: >= 0

Default value: 1

The "interior-focus" style property

```
  "interior-focus"       gboolean              : Read
```

Whether to draw the focus indicator inside widgets.

Default value: TRUE

The "secondary-cursor-color" style property

```
  "secondary-cursor-color" GdkColor              : Read
```

Color with which to draw the secondary insertion cursor when editing mixed right-to-left and left-to-right text.

Signals

The "accel-closures-changed" signal

```
void        user_function                 (GtkWidget *widget,
                                           gpointer user_data);
```

widget : the object which received the signal.
user_data : user data set when the signal handler was connected.

The "button-press-event" signal

```
gboolean    user_function                 (GtkWidget *widget,
                                           GdkEventButton *event,
                                           gpointer user_data);
```

widget : the object which received the signal.
event :
user_data : user data set when the signal handler was connected.
Returns : TRUE to stop other handlers from being invoked for the event. FALSE to propagate the event further.

The "button-release-event" signal

```
gboolean    user_function                   (GtkWidget *widget,
                                             GdkEventButton *event,
                                             gpointer user_data);
```

widget : the object which received the signal.
event :
user_data : user data set when the signal handler was connected.
Returns : TRUE to stop other handlers from being invoked for the event. FALSE to propagate the event further.

The "can-activate-accel" signal

```
gboolean    user_function                   (GtkWidget *widget,
                                             guint signal_id,
                                             gpointer user_data);
```

Determines whether an accelerator that activates the signal identified by *signal_id* can currently be activated. This signal is present to allow applications and derived widgets to override the default GtkWidget handling for determining whether an accelerator can be activated.

widget : the object which received the signal
signal_id : the ID of a signal installed on *widget*
returns : TRUE if the signal can be activated.
user_data : user data set when the signal handler was connected.

The "child-notify" signal

```
void        user_function                   (GtkWidget *widget,
                                             GParamSpec *pspec,
                                             gpointer user_data);
```

The ::child-notify signal is emitted for each child property that has changed on an object. The signal's detail holds the property name.

widget : the object which received the signal.
pspec : the GParamSpec of the changed child property.
user_data : user data set when the signal handler was connected.

The "client-event" signal

```
gboolean    user_function                   (GtkWidget *widget,
                                             GdkEventClient *event,
                                             gpointer user_data);
```

widget : the object which received the signal.
event :
user_data : user data set when the signal handler was connected.
Returns : TRUE to stop other handlers from being invoked for the event. FALSE to propagate the event further.

The "configure-event" signal

```
gboolean    user_function                   (GtkWidget *widget,
                                             GdkEventConfigure *event,
                                             gpointer user_data);
```

widget : the object which received the signal.
event :
user_data : user data set when the signal handler was connected.
Returns : TRUE to stop other handlers from being invoked for the event. FALSE to propagate the event further.

The "delete-event" signal

```
gboolean    user_function                   (GtkWidget *widget,
                                             GdkEvent *event,
                                             gpointer user_data);
```

widget : the object which received the signal.
event :
user_data : user data set when the signal handler was connected.
Returns : TRUE to stop other handlers from being invoked for the event. FALSE to propagate the event further.

The "destroy-event" signal

```
gboolean    user_function                   (GtkWidget *widget,
                                             GdkEvent *event,
                                             gpointer user_data);
```

widget : the object which received the signal.
event :
user_data : user data set when the signal handler was connected.
Returns : TRUE to stop other handlers from being invoked for the event. FALSE to propagate the event further.

The "direction-changed" signal

```
void        user_function                   (GtkWidget *widget,
                                             GtkTextDirection arg1,
                                             gpointer user_data);
```

widget : the object which received the signal.
arg1 :
user_data : user data set when the signal handler was connected.

The "drag-begin" signal

```
void        user_function                   (GtkWidget *widget,
                                             GdkDragContext *drag_context,
                                             gpointer user_data);
```

The ::drag-begin signal is emitted on the drag source when a drag is started. A typical reason to connect to this signal is to set up a custom drag icon with gtk_drag_source_set_icon().

widget : the object which received the signal.
drag_context : the drag context
user_data : user data set when the signal handler was connected.

The "drag-data-delete" signal

```
void        user_function                   (GtkWidget *widget,
                                             GdkDragContext *drag_context,
                                             gpointer user_data);
```

The ::drag-data-delete signal is emitted on the drag source when a drag with the action GDK_ACTION_MOVE is successfully completed. The signal handler is responsible for deleting the data that has been dropped. What "delete" means, depends on the context of the drag operation.

widget : the object which received the signal.
drag_context : the drag context
user_data : user data set when the signal handler was connected.

The "drag-data-get" signal

```
void        user_function                   (GtkWidget *widget,
                                             GdkDragContext *drag_context,
                                             GtkSelectionData *data,
                                             guint info,
                                             guint time,
                                             gpointer user_data);
```

The ::drag-data-get signal is emitted on the drag source when the drop site requests the data which is dragged. It is the responsibility of the signal handler to fill *data* with the data in the format which is indicated by *info*. See gtk_selection_data_set() and gtk_selection_data_set_text().

widget : the object which received the signal.
drag_context : the drag context
data : the GtkSelectionData to be filled with the dragged data
info : the info that has been registered with the target in the GtkTargetList.
time : the timestamp at which the data was requested
user_data : user data set when the signal handler was connected.

The "drag-data-received" signal

```
void        user_function                  (GtkWidget *widget,
                                            GdkDragContext *drag_context,
                                            gint x,
                                            gint y,
                                            GtkSelectionData *data,
                                            guint info,
                                            guint time,
                                            gpointer user_data);
```

The ::drag-data-received signal is emitted on the drop site when the dragged data has been received. If the data was received in order to determine whether the drop will be accepted, the handler is expected to call `gdk_drag_status()` and *not* finish the drag. If the data was received in response to a ::drag-drop signal (and this is the last target to be received), the handler for this signal is expected to process the received data and then call `gtk_drag_finish()`, setting the *`success`* parameter depending on whether the data was processed successfully.

The handler may inspect and modify *`drag_context->action`* before calling `gtk_drag_finish()`, e.g. to implement `GDK_ACTION_ASK` as shown in the following example:

```
void
drag_data_received (GtkWidget          *widget,
                    GdkDragContext     *drag_context,
                    gint                x,
                    gint                y,
                    GtkSelectionData   *data,
                    guint               info,
                    guint               time)
{
  if ((data->length >= 0) && (data->format == 8))
    {
      if (drag_context->action == GDK_ACTION_ASK)
        {
          GtkWidget *dialog;
          gint response;

          dialog = gtk_message_dialog_new (NULL,
                                           GTK_DIALOG_MODAL |
                                           GTK_DIALOG_DESTROY_WITH_PARENT,
                                           GTK_MESSAGE_INFO,
                                           GTK_BUTTONS_YES_NO,
                                           "Move the data ?\n");
          response = gtk_dialog_run (GTK_DIALOG (dialog));
          gtk_widget_destroy (dialog);

          if (response == GTK_RESPONSE_YES)
            drag_context->action = GDK_ACTION_MOVE;
          else
            drag_context->action = GDK_ACTION_COPY;
        }
```

```
      gtk_drag_finish (drag_context, TRUE, FALSE, time);
      return;
    }

  gtk_drag_finish (drag_context, FALSE, FALSE, time);
}
```

widget :	the object which received the signal.
drag_context :	the drag context
x :	where the drop happened
y :	where the drop happened
data :	the received data
info :	the info that has been registered with the target in the GtkTargetList.
time :	the timestamp at which the data was received
user_data :	user data set when the signal handler was connected.

The "drag-drop" signal

```
gboolean    user_function                   (GtkWidget *widget,
                                             GdkDragContext *drag_context,
                                             gint x,
                                             gint y,
                                             guint time,
                                             gpointer user_data);
```

The ::drag-drop signal is emitted on the drop site when the user drops the data onto the widget. The signal handler must determine whether the cursor position is in a drop zone or not. If it is not in a drop zone, it returns FALSE and no further processing is necessary. Otherwise, the handler returns TRUE. In this case, the handler must ensure that `gtk_drag_finish()` is called to let the source know that the drop is done. The call to `gtk_drag_finish()` can be done either directly or in a ::drag-data-received handler which gets triggered by calling `gtk_drop_get_data()` to receive the data for one or more of the supported targets.

widget :	the object which received the signal.
drag_context :	the drag context
x :	the x coordinate of the current cursor position
y :	the y coordinate of the current cursor position
time :	the timestamp of the motion event
returns :	whether the cursor position is in a drop zone
user_data :	user data set when the signal handler was connected.

The "drag-end" signal

```
void        user_function                   (GtkWidget *widget,
                                             GdkDragContext *drag_context,
                                             gpointer user_data);
```

The ::drag-end signal is emitted on the drag source when a drag is finished. A typical reason to connect to this signal is to undo things done in ::drag-begin.

widget :	the object which received the signal.
drag_context :	the drag context
user_data :	user data set when the signal handler was connected.

The "drag-leave" signal

```
void        user_function                   (GtkWidget *widget,
                                             GdkDragContext *drag_context,
                                             guint time,
                                             gpointer user_data);
```

The ::drag-leave signal is emitted on the drop site when the cursor leaves the widget. A typical reason to connect to this signal is to undo things done in ::drag-motion, e.g. undo highlighting with gtk_drag_unhighlight()

widget :	the object which received the signal.
drag_context :	the drag context
time :	the timestamp of the motion event
user_data :	user data set when the signal handler was connected.

The "drag-motion" signal

```
gboolean    user_function                   (GtkWidget *widget,
                                             GdkDragContext *drag_context,
                                             gint x,
                                             gint y,
                                             guint time,
                                             gpointer user_data);
```

The ::drag-motion signal is emitted on the drop site when the user moves the cursor over the widget during a drag. The signal handler must determine whether the cursor position is in a drop zone or not. If it is not in a drop zone, it returns FALSE and no further processing is necessary. Otherwise, the handler returns TRUE. In this case, the handler is responsible for providing the necessary information for displaying feedback to the user, by calling gdk_drag_status(). If the decision whether the drop will be accepted or rejected can't be made based solely on the cursor position and the type of the data, the handler may inspect

the dragged data by calling gtk_drag_get_data() and defer the gdk_drag_status() call to the ::drag-data-received handler.

Note that there is no ::drag-enter signal. The drag receiver has to keep track of whether he has received any ::drag-motion signals since the last ::drag-leave and if not, treat the ::drag-motion signal as an "enter" signal. Upon an "enter", the handler will typically highlight the drop site with gtk_drag_highlight().

```
static void
drag_motion (GtkWidget *widget,
          GdkDragContext *context,
             gint x,
             gint y,
             guint time)
{
  GdkAtom target;

  PrivateData *private_data = GET_PRIVATE_DATA (widget);

  if (!private_data->drag_highlight)
   {
     private_data->drag_highlight = 1;
     gtk_drag_highlight (widget);
   }

  target = gtk_drag_dest_find_target (widget, context, NULL);
  if (target == GDK_NONE)
    gdk_drag_status (context, 0, time);
  else
   {
     private_data->pending_status = context->suggested_action;
     gtk_drag_get_data (widget, context, target, time);
   }

  return TRUE;
}

static void
drag_data_received (GtkWidget        *widget,
                    GdkDragContext   *context,
                    gint              x,
                    gint              y,
                    GtkSelectionData *selection_data,
                    guint             info,
                    guint             time)
{
  PrivateData *private_data = GET_PRIVATE_DATA (widget);

  if (private_data->suggested_action)
   {
     private_data->suggested_action = 0;

    /* We are getting this data due to a request in drag_motion,
```

```
     * rather than due to a request in drag_drop, so we are just
     * supposed to call gdk_drag_status(), not actually paste in the data.
     */
     str = gtk_selection_data_get_text (selection_data);
     if (!data_is_acceptable (str))
       gdk_drag_status (context, 0, time);
     else
       gdk_drag_status (context, private_data->suggested_action, time);
   }
  else
   {
     /* accept the drop */
   }
}
```

widget :	the object which received the signal.
drag_context :	the drag context
x :	the x coordinate of the current cursor position
y :	the y coordinate of the current cursor position
time :	the timestamp of the motion event
returns :	whether the cursor position is in a drop zone
user_data :	user data set when the signal handler was connected.

The "enter-notify-event" signal

```
gboolean    user_function                  (GtkWidget *widget,
                                            GdkEventCrossing *event,
                                            gpointer user_data);
```

widget :	the object which received the signal.
event :	
user_data :	user data set when the signal handler was connected.
Returns :	TRUE to stop other handlers from being invoked for the event. FALSE to propagate the event further.

The "event" signal

```
gboolean    user_function                  (GtkWidget *widget,
                                            GdkEvent *event,
                                            gpointer user_data);
```

widget : the object which received the signal.
event :
user_data : user data set when the signal handler was connected.
Returns : TRUE to stop other handlers from being invoked for the event. FALSE to propagate the event further.

The "event-after" signal

```
void        user_function                   (GtkWidget *widget,
                                             GdkEvent *event,
                                             gpointer user_data);
```

widget : the object which received the signal.
event :
user_data : user data set when the signal handler was connected.

The "expose-event" signal

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

widget : the object which received the signal.
event :
user_data : user data set when the signal handler was connected.
Returns : TRUE to stop other handlers from being invoked for the event. FALSE to propagate the event further.

The "focus" signal

```
gboolean    user_function                   (GtkWidget *widget,
                                             GtkDirectionType arg1,
                                             gpointer user_data);
```

widget : the object which received the signal.
arg1 :
user_data : user data set when the signal handler was connected.

Returns : TRUE to stop other handlers from being invoked for the event. FALSE to propagate the event further.

The "focus-in-event" signal

```
gboolean    user_function                   (GtkWidget *widget,
                                             GdkEventFocus *event,
                                             gpointer user_data);
```

widget : the object which received the signal.
event :
user_data : user data set when the signal handler was connected.
Returns : TRUE to stop other handlers from being invoked for the event. FALSE to propagate the event further.

The "focus-out-event" signal

```
gboolean    user_function                   (GtkWidget *widget,
                                             GdkEventFocus *event,
                                             gpointer user_data);
```

widget : the object which received the signal.
event :
user_data : user data set when the signal handler was connected.
Returns : TRUE to stop other handlers from being invoked for the event. FALSE to propagate the event further.

The "grab-focus" signal

```
void        user_function                   (GtkWidget *widget,
                                             gpointer user_data);
```

widget : the object which received the signal.
user_data : user data set when the signal handler was connected.

The "grab-notify" signal

```
void        user_function                   (GtkWidget *widget,
                                             gboolean arg1,
                                             gpointer user_data);
```

widget : the object which received the signal.
arg1 :
user_data : user data set when the signal handler was connected.

The "hide" signal

```
void        user_function                   (GtkWidget *widget,
                                             gpointer user_data);
```

widget : the object which received the signal.
user_data : user data set when the signal handler was connected.

The "hierarchy-changed" signal

```
void        user_function                   (GtkWidget *widget,
                                             GtkWidget *widget2,
                                             gpointer user_data);
```

Emitted when there is a chance in the hierarchy to which a widget belong. More precisely, a widget is *anchored* when its toplevel ancestor is a GtkWindow. This signal is emitted when a widget changes from un-anchored to anchored or vice-versa.

widget : the object which received the signal.
widget2 :
user_data : user data set when the signal handler was connected.

The "key-press-event" signal

```
gboolean    user_function                   (GtkWidget *widget,
                                             GdkEventKey *event,
                                             gpointer user_data);
```

widget : the object which received the signal.

event :

user_data : user data set when the signal handler was connected.

Returns : TRUE to stop other handlers from being invoked for the event. FALSE to propagate the event further.

The "key-release-event" signal

```
gboolean    user_function                   (GtkWidget *widget,
                                             GdkEventKey *event,
                                             gpointer user_data);
```

widget : the object which received the signal.

event :

user_data : user data set when the signal handler was connected.

Returns : TRUE to stop other handlers from being invoked for the event. FALSE to propagate the event further.

The "leave-notify-event" signal

```
gboolean    user_function                   (GtkWidget *widget,
                                             GdkEventCrossing *event,
                                             gpointer user_data);
```

widget : the object which received the signal.

event :

user_data : user data set when the signal handler was connected.

Returns : TRUE to stop other handlers from being invoked for the event. FALSE to propagate the event further.

The "map" signal

```
void        user_function                   (GtkWidget *widget,
                                             gpointer user_data);
```

widget : the object which received the signal.

user_data : user data set when the signal handler was connected.

The "map-event" signal

```
gboolean    user_function                   (GtkWidget *widget,
                                             GdkEvent *event,
                                             gpointer user_data);
```

widget : the object which received the signal.
event :
user_data : user data set when the signal handler was connected.
Returns : TRUE to stop other handlers from being invoked for the event. FALSE to propagate the event further.

The "mnemonic-activate" signal

```
gboolean    user_function                   (GtkWidget *widget,
                                             gboolean arg1,
                                             gpointer user_data);
```

widget : the object which received the signal.
arg1 :
user_data : user data set when the signal handler was connected.
Returns :

The "motion-notify-event" signal

```
gboolean    user_function                   (GtkWidget *widget,
                                             GdkEventMotion *event,
                                             gpointer user_data);
```

widget : the object which received the signal.
event :
user_data : user data set when the signal handler was connected.
Returns : TRUE to stop other handlers from being invoked for the event. FALSE to propagate the event further.

The "no-expose-event" signal

```
gboolean    user_function                  (GtkWidget *widget,
                                            GdkEventNoExpose *event,
                                            gpointer user_data);
```

widget : the object which received the signal.
event :
user_data : user data set when the signal handler was connected.
Returns : TRUE to stop other handlers from being invoked for the event. FALSE to propagate the event further.

The "parent-set" signal

```
void        user_function                  (GtkWidget *widget,
                                            GtkObject *old_parent,
                                            gpointer user_data);
```

widget : the object which received the signal.
old_parent :
user_data : user data set when the signal handler was connected.

The "popup-menu" signal

```
gboolean    user_function                  (GtkWidget *widget,
                                            gpointer user_data);
```

This signal gets emitted whenever a widget should pop up a context-sensitive menu. This usually happens through the standard key binding mechanism; by pressing a certain key while a widget is focused, the user can cause the widget to pop up a menu. For example, the GtkEntry widget creates a menu with clipboard commands. See the section called “Implement GtkWidget::popup_menu” for an example of how to use this signal.

widget : the object which received the signal
returns : TRUE if a menu was activated
user_data : user data set when the signal handler was connected.

The "property-notify-event" signal

```
gboolean    user_function                  (GtkWidget *widget,
                                            GdkEventProperty *event,
```

```
                                        gpointer user_data);
```

widget : the object which received the signal.
event :
user_data : user data set when the signal handler was connected.
Returns : TRUE to stop other handlers from being invoked for the event. FALSE to propagate the event further.

The "proximity-in-event" signal

```
gboolean    user_function               (GtkWidget *widget,
                                         GdkEventProximity *event,
                                         gpointer user_data);
```

widget : the object which received the signal.
event :
user_data : user data set when the signal handler was connected.
Returns : TRUE to stop other handlers from being invoked for the event. FALSE to propagate the event further.

The "proximity-out-event" signal

```
gboolean    user_function               (GtkWidget *widget,
                                         GdkEventProximity *event,
                                         gpointer user_data);
```

widget : the object which received the signal.
event :
user_data : user data set when the signal handler was connected.
Returns : TRUE to stop other handlers from being invoked for the event. FALSE to propagate the event further.

The "realize" signal

```
void        user_function               (GtkWidget *widget,
                                         gpointer user_data);
```

widget : the object which received the signal.
user_data : user data set when the signal handler was connected.

The "screen-changed" signal

```
void        user_function                   (GtkWidget *widget,
                                             GdkScreen *arg1,
                                             gpointer user_data);
```

widget : the object which received the signal.
arg1 :
user_data : user data set when the signal handler was connected.

The "scroll-event" signal

```
gboolean    user_function                   (GtkWidget *widget,
                                             GdkEventScroll *event,
                                             gpointer user_data);
```

widget : the object which received the signal.
event :
user_data : user data set when the signal handler was connected.
Returns : TRUE to stop other handlers from being invoked for the event. FALSE to propagate the event further.

The "selection-clear-event" signal

```
gboolean    user_function                   (GtkWidget *widget,
                                             GdkEventSelection *event,
                                             gpointer user_data);
```

widget : the object which received the signal.
event :
user_data : user data set when the signal handler was connected.
Returns : TRUE to stop other handlers from being invoked for the event. FALSE to propagate the event further.

The "selection-get" signal

```
void        user_function                   (GtkWidget *widget,
                                             GtkSelectionData *data,
                                             guint info,
                                             guint time,
                                             gpointer user_data);
```

widget : the object which received the signal.
data :
info :
time :
user_data : user data set when the signal handler was connected.

The "selection-notify-event" signal

```
gboolean    user_function                   (GtkWidget *widget,
                                             GdkEventSelection *event,
                                             gpointer user_data);
```

widget : the object which received the signal.
event :
user_data : user data set when the signal handler was connected.
Returns : TRUE to stop other handlers from being invoked for the event. FALSE to propagate the event further.

The "selection-received" signal

```
void        user_function                   (GtkWidget *widget,
                                             GtkSelectionData *data,
                                             guint time,
                                             gpointer user_data);
```

widget : the object which received the signal.
data :
time :
user_data : user data set when the signal handler was connected.

The "selection-request-event" signal

```
gboolean    user_function                   (GtkWidget *widget,
                                             GdkEventSelection *event,
                                             gpointer user_data);
```

widget : the object which received the signal.
event :
user_data : user data set when the signal handler was connected.
Returns : TRUE to stop other handlers from being invoked for the event. FALSE to propagate the event further.

The "show" signal

```
void        user_function                   (GtkWidget *widget,
                                             gpointer user_data);
```

widget : the object which received the signal.
user_data : user data set when the signal handler was connected.

The "show-help" signal

```
gboolean    user_function                   (GtkWidget *widget,
                                             GtkWidgetHelpType arg1,
                                             gpointer user_data);
```

widget : the object which received the signal.
arg1 :
user_data : user data set when the signal handler was connected.
Returns :

The "size-allocate" signal

```
void        user_function                   (GtkWidget *widget,
                                             GtkAllocation *allocation,
                                             gpointer user_data);
```

widget : the object which received the signal.
allocation :
user_data : user data set when the signal handler was connected.

The "size-request" signal

```
void        user_function                   (GtkWidget *widget,
                                             GtkRequisition *requisition,
                                             gpointer user_data);
```

widget : the object which received the signal.
requisition :
user_data : user data set when the signal handler was connected.

The "state-changed" signal

```
void        user_function                   (GtkWidget *widget,
                                             GtkStateType state,
                                             gpointer user_data);
```

widget : the object which received the signal.
state :
user_data : user data set when the signal handler was connected.

The "style-set" signal

```
void        user_function                   (GtkWidget *widget,
                                             GtkStyle *previous_style,
                                             gpointer user_data);
```

widget : the object which received the signal.
previous_style :
user_data : user data set when the signal handler was connected.

The "unmap" signal

```
void        user_function                   (GtkWidget *widget,
                                             gpointer user_data);
```

widget : the object which received the signal.
user_data : user data set when the signal handler was connected.

The "unmap-event" signal

```
gboolean    user_function                   (GtkWidget *widget,
                                             GdkEvent *event,
                                             gpointer user_data);
```

widget : the object which received the signal.
event :
user_data : user data set when the signal handler was connected.
Returns : TRUE to stop other handlers from being invoked for the event. FALSE to propagate the event further.

The "unrealize" signal

```
void        user_function                   (GtkWidget *widget,
                                             gpointer user_data);
```

widget : the object which received the signal.
user_data : user data set when the signal handler was connected.

The "visibility-notify-event" signal

```
gboolean    user_function                   (GtkWidget *widget,
                                             GdkEventVisibility *event,
                                             gpointer user_data);
```

widget : the object which received the signal.
event :
user_data : user data set when the signal handler was connected.

Returns : TRUE to stop other handlers from being invoked for the event. FALSE to propagate the event further.

The "window-state-event" signal

```
gboolean    user_function                   (GtkWidget *widget,
                                             GdkEventWindowState *event,
                                             gpointer user_data);
```

widget : the object which received the signal.

event :

user_data : user data set when the signal handler was connected.

Returns : TRUE to stop other handlers from being invoked for the event. FALSE to propagate the event further.

<< GtkSeparator **GtkIMContext >>**

GtkIMContext

GtkIMContext — Base class for input method contexts

Synopsis

```
#include <gtk/gtk.h>

            GtkIMContext;
void        gtk_im_context_set_client_window
                                            (GtkIMContext *context,
                                             GdkWindow *window);
void        gtk_im_context_get_preedit_string
                                            (GtkIMContext *context,
                                             gchar **str,
                                             PangoAttrList **attrs,
                                             gint *cursor_pos);
gboolean    gtk_im_context_filter_keypress  (GtkIMContext *context,
                                             GdkEventKey *event);
void        gtk_im_context_focus_in         (GtkIMContext *context);
void        gtk_im_context_focus_out        (GtkIMContext *context);
void        gtk_im_context_reset            (GtkIMContext *context);
void        gtk_im_context_set_cursor_location
                                            (GtkIMContext *context,
                                             GdkRectangle *area);
void        gtk_im_context_set_use_preedit  (GtkIMContext *context,
                                             gboolean use_preedit);
void        gtk_im_context_set_surrounding  (GtkIMContext *context,
                                             const gchar *text,
                                             gint len,
                                             gint cursor_index);
```

```
gboolean    gtk_im_context_get_surrounding  (GtkIMContext *context,
                                             gchar **text,
                                             gint *cursor_index);
gboolean    gtk_im_context_delete_surrounding
                                            (GtkIMContext *context,
                                             gint offset,
                                             gint n_chars);
```

Object Hierarchy

```
  GObject
   +----GtkIMContext
         +----GtkIMContextSimple
         +----GtkIMMulticontext
```

Signal Prototypes

```
"commit"    void        user_function      (GtkIMContext *imcontext,
                                            gchar *arg1,
                                            gpointer user_data);
"delete-surrounding"
            gboolean    user_function      (GtkIMContext *imcontext,
                                            gint arg1,
                                            gint arg2,
                                            gpointer user_data);
"preedit-changed"
            void        user_function      (GtkIMContext *imcontext,
                                            gpointer user_data);
"preedit-end"
            void        user_function      (GtkIMContext *imcontext,
                                            gpointer user_data);
"preedit-start"
```

```
            void        user_function      (GtkIMContext *imcontext,
                                            gpointer user_data);
"retrieve-surrounding"
            gboolean    user_function      (GtkIMContext *imcontext,
                                            gpointer user_data);
```

Description

Details

GtkIMContext

```
typedef struct _GtkIMContext GtkIMContext;
```

---

gtk_im_context_set_client_window ()

```
void        gtk_im_context_set_client_window
                                           (GtkIMContext *context,
                                            GdkWindow *window);
```

Set the client window for the input context; this is the GdkWindow in which the input appears. This window is used in order to correctly position status windows, and may also be used for purposes internal to the input method.

context : a GtkIMContext

window : the client window. This may be NULL to indicate that the previous client window no longer exists.

---

gtk_im_context_get_preedit_string ()

```
```

```
void        gtk_im_context_get_preedit_string
                                            (GtkIMContext *context,
                                             gchar **str,
                                             PangoAttrList **attrs,
                                             gint *cursor_pos);
```

Retrieve the current preedit string for the input context, and a list of attributes to apply to the string. This string should be displayed inserted at the insertion point.

context :	a GtkIMContext
str :	location to store the retrieved string. The string retrieved must be freed with g_free().
attrs :	location to store the retrieved attribute list. When you are done with this list, you must unreference it with pango_attr_list_unref().
cursor_pos :	location to store position of cursor (in characters) within the preedit string.

---

gtk_im_context_filter_keypress ()

```
gboolean    gtk_im_context_filter_keypress  (GtkIMContext *context,
                                             GdkEventKey *event);
```

Allow an input method to internally handle a key press event. If this function returns TRUE, then no further processing should be done for this keystroke.

context :	a GtkIMContext
event :	the key event
Returns :	TRUE if the input method handled the keystroke.

---

gtk_im_context_focus_in ()

```
void        gtk_im_context_focus_in         (GtkIMContext *context);
```

Notify the input method that the widget to which this input context corresponds has gained focus. The input method may, for example, change the displayed feedback to reflect this change.

context : a GtkIMContext

---

gtk_im_context_focus_out ()

```
void        gtk_im_context_focus_out        (GtkIMContext *context);
```

Notify the input method that the widget to which this input context corresponds has lost focus. The input method may, for example, change the displayed feedback or reset the contexts state to reflect this change.

context : a GtkIMContext

---

gtk_im_context_reset ()

```
void        gtk_im_context_reset            (GtkIMContext *context);
```

Notify the input method that a change such as a change in cursor position has been made. This will typically cause the input method to clear the preedit state.

context : a GtkIMContext

---

gtk_im_context_set_cursor_location ()

```
void        gtk_im_context_set_cursor_location
                                            (GtkIMContext *context,
                                             GdkRectangle *area);
```

Notify the input method that a change in cursor position has been made. The location is relative to the

client window.

context : a GtkIMContext
area : new location

---

gtk_im_context_set_use_preedit ()

```
void        gtk_im_context_set_use_preedit  (GtkIMContext *context,
                                             gboolean use_preedit);
```

Sets whether the IM context should use the preedit string to display feedback. If *use_preedit* is FALSE (default is TRUE), then the IM context may use some other method to display feedback, such as displaying it in a child of the root window.

context : a GtkIMContext
use_preedit : whether the IM context should use the preedit string.

---

gtk_im_context_set_surrounding ()

```
void        gtk_im_context_set_surrounding  (GtkIMContext *context,
                                             const gchar *text,
                                             gint len,
                                             gint cursor_index);
```

Sets surrounding context around the insertion point and preedit string. This function is expected to be called in response to the GtkIMContext::retrieve_surrounding signal, and will likely have no effect if called at other times.

context : a GtkIMContext
text : text surrounding the insertion point, as UTF-8. the preedit string should not be included within *text*.
len : the length of *text*, or -1 if *text* is nul-terminated

cursor_index : the byte index of the insertion cursor within *text*.

---

gtk_im_context_get_surrounding ()

```
gboolean    gtk_im_context_get_surrounding  (GtkIMContext *context,
                                             gchar **text,
                                             gint *cursor_index);
```

Retrieves context around the insertion point. Input methods typically want context in order to constrain input text based on existing text; this is important for languages such as Thai where only some sequences of characters are allowed.

This function is implemented by emitting the GtkIMContext::retrieve_surrounding signal on the input method; in response to this signal, a widget should provide as much context as is available, up to an entire paragraph, by calling gtk_im_context_set_surrounding(). Note that there is no obligation for a widget to respond to the ::retrieve_surrounding signal, so input methods must be prepared to function without context.

context :	a GtkIMContext
text :	location to store a UTF-8 encoded string of text holding context around the insertion point. If the function returns TRUE, then you must free the result stored in this location with g_free().
cursor_index :	location to store byte index of the insertion cursor within *text*.
Returns :	TRUE if surrounding text was provided; in this case you must free the result stored in *text.

---

gtk_im_context_delete_surrounding ()

```
gboolean    gtk_im_context_delete_surrounding
                                            (GtkIMContext *context,
                                             gint offset,
                                             gint n_chars);
```

Asks the widget that the input context is attached to to delete characters around the cursor position by

emitting the GtkIMContext::delete_surrounding signal. Note that *offset* and *n_chars* are in characters not in bytes which differs from the usage other places in GtkIMContext.

In order to use this function, you should first call gtk_im_context_get_surrounding() to get the current context, and call this function immediately afterwards to make sure that you know what you are deleting. You should also account for the fact that even if the signal was handled, the input context might not have deleted all the characters that were requested to be deleted.

This function is used by an input method that wants to make subsitutions in the existing text in response to new input. It is not useful for applications.

context : a GtkIMContext

offset : offset from cursor position in chars; a negative value means start before the cursor.

n_chars : number of characters to delete.

Returns : TRUE if the signal was handled.

Signals

The "commit" signal

```
void        user_function                   (GtkIMContext *imcontext,
                                             gchar *arg1,
                                             gpointer user_data);
```

imcontext : the object which received the signal.

arg1 :

user_data : user data set when the signal handler was connected.

The "delete-surrounding" signal

```
gboolean    user_function                   (GtkIMContext *imcontext,
                                             gint arg1,
                                             gint arg2,
```

```
                                                gpointer user_data);
```

imcontext : the object which received the signal.
arg1 :
arg2 :
user_data : user data set when the signal handler was connected.
Returns :

The "preedit-changed" signal

```
void        user_function                   (GtkIMContext *imcontext,
                                             gpointer user_data);
```

imcontext : the object which received the signal.
user_data : user data set when the signal handler was connected.

The "preedit-end" signal

```
void        user_function                   (GtkIMContext *imcontext,
                                             gpointer user_data);
```

imcontext : the object which received the signal.
user_data : user data set when the signal handler was connected.

The "preedit-start" signal

```
void        user_function                   (GtkIMContext *imcontext,
                                             gpointer user_data);
```

imcontext : the object which received the signal.

user_data : user data set when the signal handler was connected.

The "retrieve-surrounding" signal

```
gboolean    user_function                   (GtkIMContext *imcontext,
                                             gpointer user_data);
```

imcontext : the object which received the signal.

user_data : user data set when the signal handler was connected.

Returns :

<< GtkWidget **Cross-process Embedding >>**

Cross-process Embedding

GtkPlug - Toplevel for embedding into other processes
GtkSocket - Container for widgets from other processes

<< GtkIMContext **GtkPlug >>**

GtkPlug

GtkPlug — Toplevel for embedding into other processes

Synopsis

```
#include <gtk/gtk.h>

            GtkPlug;
void        gtk_plug_construct              (GtkPlug *plug,
                                             GdkNativeWindow socket_id);
void        gtk_plug_construct_for_display  (GtkPlug *plug,
                                             GdkDisplay *display,
                                             GdkNativeWindow socket_id);
GtkWidget*  gtk_plug_new                    (GdkNativeWindow socket_id);
GtkWidget*  gtk_plug_new_for_display        (GdkDisplay *display,
                                             GdkNativeWindow socket_id);
GdkNativeWindow gtk_plug_get_id             (GtkPlug *plug);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkWidget
               +----GtkContainer
                     +----GtkBin
                           +----GtkWindow
                                 +----GtkPlug
```

Implemented Interfaces

GtkPlug implements AtkImplementorIface.

Signal Prototypes

```
"embedded"  void        user_function      (GtkPlug *plug,
                                            gpointer user_data);
```

Description

Together with GtkSocket, GtkPlug provides the ability to embed widgets from one process into another process in a fashion that is transparent to the user. One process creates a GtkSocket widget and, passes the ID of that widgets window to the other process, which then creates a GtkPlug with that window ID. Any widgets contained in the GtkPlug then will appear inside the first applications window.

Details

GtkPlug

```
typedef struct _GtkPlug GtkPlug;
```

---

gtk_plug_construct ()

```
void        gtk_plug_construct                 (GtkPlug *plug,
                                                GdkNativeWindow socket_id);
```

Finish the initialization of *plug* for a given GtkSocket identified by *socket_id*. This function will generally only be used by classes deriving from GtkPlug.

plug : a GtkPlug.

socket_id : the XID of the socket's window.

---

gtk_plug_construct_for_display ()

```
void        gtk_plug_construct_for_display  (GtkPlug *plug,
                                             GdkDisplay *display,
                                             GdkNativeWindow socket_id);
```

Finish the initialization of *plug* for a given GtkSocket identified by *socket_id* which is currently displayed on *display*. This function will generally only be used by classes deriving from GtkPlug.

plug : a GtkPlug.
display : the GdkDisplay associated with *socket_id*'s GtkSocket.
socket_id : the XID of the socket's window.

Since 2.2

---

gtk_plug_new ()

```
GtkWidget*  gtk_plug_new                    (GdkNativeWindow socket_id);
```

Creates a new plug widget inside the GtkSocket identified by *socket_id*. If *socket_id* is 0, the plug is left "unplugged" and can later be plugged into a GtkSocket by `gtk_socket_add_id()`.

socket_id : the window ID of the socket, or 0.
Returns : the new GtkPlug widget.

---

gtk_plug_new_for_display ()

```
GtkWidget*  gtk_plug_new_for_display        (GdkDisplay *display,
                                             GdkNativeWindow socket_id);
```

Create a new plug widget inside the GtkSocket identified by socket_id.

display : the GdkDisplay on which *socket_id* is displayed
socket_id : the XID of the socket's window.
Returns : the new GtkPlug widget.

Since 2.2

---

gtk_plug_get_id ()

```
GdkNativeWindow gtk_plug_get_id             (GtkPlug *plug);
```

Gets the window ID of a GtkPlug widget, which can then be used to embed this window inside another window, for instance with `gtk_socket_add_id()`.

plug : a GtkPlug.
Returns : the window ID for the plug

Signals

The "embedded" signal

```
void        user_function                   (GtkPlug *plug,
                                             gpointer user_data);
```

plug : the object which received the signal.
user_data : user data set when the signal handler was connected.

See Also

GtkSocket the widget that a GtkPlug plugs into.

<< Cross-process Embedding **GtkSocket >>**

GtkSocket

GtkSocket — Container for widgets from other processes

Synopsis

```
#include <gtk/gtk.h>


            GtkSocket;
GtkWidget*  gtk_socket_new                  (void);
void        gtk_socket_steal                (GtkSocket *socket_,
                                             GdkNativeWindow wid);
void        gtk_socket_add_id               (GtkSocket *socket_,
                                             GdkNativeWindow window_id);
GdkNativeWindow gtk_socket_get_id           (GtkSocket *socket_);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkWidget
               +----GtkContainer
                     +----GtkSocket
```

Implemented Interfaces

GtkSocket implements AtkImplementorIface.

Signal Prototypes

```
"plug-added"
            void        user_function      (GtkSocket *socket,
                                            gpointer user_data);
"plug-removed"
            gboolean    user_function      (GtkSocket *socket,
                                            gpointer user_data);
```

Description

Together with GtkPlug, GtkSocket provides the ability to embed widgets from one process into another process in a fashion that is transparent to the user. One process creates a GtkSocket widget and, passes the that widget's window ID to the other process, which then creates a GtkPlug with that window ID. Any widgets contained in the GtkPlug then will appear inside the first applications window.

The socket's window ID is obtained by using `gtk_socket_get_id()`. Before using this function, the socket must have been realized, and for hence, have been added to its parent.

Example 1. Obtaining the window ID of a socket.

```
GtkWidget *socket = gtk_socket_new ();
gtk_widget_show (socket);
gtk_container_add (GTK_CONTAINER (parent), socket);

/* The following call is only necessary if one of
 * the ancestors of the socket is not yet visible.
 */
gtk_widget_realize (socket);
g_print ("The ID of the sockets window is %x\n",
         gtk_socket_get_id (socket));
```

Note that if you pass the window ID of the socket to another process that will create a plug in the socket, you must make sure that the socket widget is not destroyed until that plug is created. Violating this rule will cause unpredictable consequences, the most likely consequence being that the plug will appear as a separate toplevel window. You can check if the plug has been created by examining the *`plug_window`* field of the GtkSocket structure. If this field is non-`NULL`, then the plug has been successfully created inside of the socket.

When GTK+ is notified that the embedded window has been destroyed, then it will destroy the socket as well. You should always, therefore, be prepared for your sockets to be destroyed at any time when the main event loop is running.

The communication between a GtkSocket and a GtkPlug follows the XEmbed protocol. This protocol has also been implemented in other toolkits, e.g. Qt, allowing the same level of integration when embedding a Qt widget in GTK or vice versa.

A socket can also be used to swallow arbitrary pre-existing top-level windows using `gtk_socket_steal()`, though the integration when this is done will not be as close as between a GtkPlug and a GtkSocket.

Details

GtkSocket

```
typedef struct _GtkSocket GtkSocket;
```

The GtkSocket structure contains the *`plug_window`* field. (This field should be considered read-only. It should never be set by an application.)

---

gtk_socket_new ()

```
GtkWidget*  gtk_socket_new                  (void);
```

Create a new empty GtkSocket.

Returns : the new GtkSocket.

---

gtk_socket_steal ()

```
void        gtk_socket_steal                (GtkSocket *socket_,
```

```
                                                GdkNativeWindow wid);
```

Warning

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

Reparents a pre-existing toplevel window into a GtkSocket. This is meant to embed clients that do not know about embedding into a GtkSocket, however doing so is inherently unreliable, and using this function is not recommended.

The GtkSocket must have already be added into a toplevel window before you can make this call.

*socket_* : a GtkSocket
wid : the window ID of an existing toplevel window.

---

gtk_socket_add_id ()

```
void        gtk_socket_add_id               (GtkSocket *socket_,
                                             GdkNativeWindow window_id);
```

Adds an XEMBED client, such as a GtkPlug, to the GtkSocket. The client may be in the same process or in a different process.

To embed a GtkPlug in a GtkSocket, you can either create the GtkPlug with `gtk_plug_new (0)`, call `gtk_plug_get_id()` to get the window ID of the plug, and then pass that to the `gtk_socket_add_id()`, or you can call `gtk_socket_get_id()` to get the window ID for the socket, and call `gtk_plug_new()` passing in that ID.

The GtkSocket must have already be added into a toplevel window before you can make this call.

*socket_* : a GtkSocket
window_id : the window ID of a client participating in the XEMBED protocol.

---

gtk_socket_get_id ()

```
GdkNativeWindow gtk_socket_get_id           (GtkSocket *socket_);
```

Gets the window ID of a GtkSocket widget, which can then be used to create a client embedded inside the socket, for instance with `gtk_plug_new()`.

The GtkSocket must have already be added into a toplevel window before you can make this call.

*socket_* : a GtkSocket.
Returns : the window ID for the socket

Signals

The "plug-added" signal

```
void        user_function                   (GtkSocket *socket,
                                             gpointer user_data);
```

This signal is emitted when a client is successfully added to the socket.

socket : the object which received the signal.
user_data : user data set when the signal handler was connected.

The "plug-removed" signal

```
gboolean    user_function                   (GtkSocket *socket,
                                             gpointer user_data);
```

This signal is emitted when a client is removed from the socket. The default action is to destroy the GtkSocket widget, so if you want to reuse it you must add a signal handler that returns TRUE.

socket : the object which received the signal.
user_data : user data set when the signal handler was connected.

Returns :

See Also

GtkPlug the widget that plugs into a GtkSocket.

XEmbed the XEmbed Protocol Specification.

<< GtkPlug **Special-purpose features >>**

Special-purpose features

GtkCurve - Allows direct editing of a curve
GtkGammaCurve - a subclass of GtkCurve for editing gamma curves.
GtkRuler - Base class for horizontal or vertical rulers
GtkHRuler - A horizontal ruler.
GtkVRuler - A vertical ruler.

<< GtkSocket **GtkCurve >>**

GtkCurve

GtkCurve — Allows direct editing of a curve

Synopsis

```
#include <gtk/gtk.h>


            GtkCurve;
GtkWidget*  gtk_curve_new                   (void);
void        gtk_curve_reset                 (GtkCurve *curve);
void        gtk_curve_set_gamma             (GtkCurve *curve,
                                             gfloat gamma_);
void        gtk_curve_set_range             (GtkCurve *curve,
                                             gfloat min_x,
                                             gfloat max_x,
                                             gfloat min_y,
                                             gfloat max_y);
void        gtk_curve_get_vector            (GtkCurve *curve,
                                             int veclen,
                                             gfloat vector[]);
void        gtk_curve_set_vector            (GtkCurve *curve,
                                             int veclen,
                                             gfloat vector[]);
void        gtk_curve_set_curve_type        (GtkCurve *curve,
                                             GtkCurveType type);
```

Object Hierarchy

```
GObject
 +----GtkObject
       +----GtkWidget
             +----GtkDrawingArea
                   +----GtkCurve
```

Implemented Interfaces

GtkCurve implements AtkImplementorIface.

Properties

```
"curve-type"           GtkCurveType          : Read / Write
"max-x"                gfloat                : Read / Write
"max-y"                gfloat                : Read / Write
"min-x"                gfloat                : Read / Write
"min-y"                gfloat                : Read / Write
```

Signal Prototypes

```
"curve-type-changed"
            void        user_function      (GtkCurve *curve,
                                            gpointer user_data);
```

Description

Note

This widget is considered too specialized/little-used for GTK+, and will in the future be moved to some other package. If your application needs this widget, feel free to use it, as the widget does work and is useful in some applications; it's just not of general interest.

However, we are not accepting new features for the widget, and it will eventually move out of the GTK+ distribution.

The GtkCurve widget allows the user to edit a curve covering a range of values. It is typically used to fine-tune color balances in graphics applications like the Gimp.

The GtkCurve widget has 3 modes of operation - spline, linear and free. In spline mode the user places points on the curve which are automatically connected together into a smooth curve. In linear mode the user places points on the curve which are connected by straight lines. In free mode the user can draw the points of the curve freely, and they are not connected at all.

Details

GtkCurve

```
typedef struct _GtkCurve GtkCurve;
```

The GtkCurve-struct struct contains private data only, and should be accessed using the functions below.

---

gtk_curve_new ()

```
GtkWidget*  gtk_curve_new                   (void);
```

Creates a new GtkCurve.

Returns : a new GtkCurve.

---

gtk_curve_reset ()

```
void        gtk_curve_reset                 (GtkCurve *curve);
```

Resets the curve to a straight line from the minimum x and y values to the maximum x and y values (i.e. from the bottom-left to the top-right corners). The curve type is not changed.

curve : a GtkCurve.

---

gtk_curve_set_gamma ()

```
void        gtk_curve_set_gamma             (GtkCurve *curve,
                                             gfloat gamma_);
```

Recomputes the entire curve using the given gamma value. A gamma value of 1 results in a straight line. Values greater than 1 result in a curve above the straight line. Values less than 1 result in a curve below the straight line. The curve type is changed to GTK_CURVE_TYPE_FREE. FIXME: Needs a more precise definition of gamma.

curve : a GtkCurve.

*gamma_* : the gamma value.

---

gtk_curve_set_range ()

```
void        gtk_curve_set_range             (GtkCurve *curve,
                                             gfloat min_x,
                                             gfloat max_x,
                                             gfloat min_y,
                                             gfloat max_y);
```

Sets the minimum and maximum x and y values of the curve. The curve is also reset with a call to gtk_curve_reset().

curve : a GtkCurve.

min_x : the minimum x value.

max_x : the maximum x value.

min_y : the minimum y value.
max_y : the maximum y value.

---

gtk_curve_get_vector ()

```
void        gtk_curve_get_vector            (GtkCurve *curve,
                                             int veclen,
                                             gfloat vector[]);
```

Returns a vector of points representing the curve.

curve : a GtkCurve.
veclen : the number of points to calculate.
vector : returns the points.

---

gtk_curve_set_vector ()

```
void        gtk_curve_set_vector            (GtkCurve *curve,
                                             int veclen,
                                             gfloat vector[]);
```

Sets the vector of points on the curve. The curve type is set to GTK_CURVE_TYPE_FREE.

curve : a GtkCurve.
veclen : the number of points.
vector : the points on the curve.

---

gtk_curve_set_curve_type ()

```
void        gtk_curve_set_curve_type        (GtkCurve *curve,
```

```
                                                GtkCurveType type);
```

Sets the type of the curve. The curve will remain unchanged except when changing from a free curve to a linear or spline curve, in which case the curve will be changed as little as possible.

curve : a GtkCurve.

type : the type of the curve.

Properties

The "curve-type" property

```
  "curve-type"           GtkCurveType          : Read / Write
```

Is this curve linear, spline interpolated, or free-form.

Default value: GTK_CURVE_TYPE_LINEAR

The "max-x" property

```
  "max-x"                gfloat                : Read / Write
```

Maximum possible X value.

Default value: 1

The "max-y" property

```
  "max-y"                gfloat                : Read / Write
```

Maximum possible value for Y.

Default value: 1

The "min-x" property

```
  "min-x"                gfloat                : Read / Write
```

Minimum possible value for X.

Default value: 0

The "min-y" property

```
  "min-y"                gfloat                : Read / Write
```

Minimum possible value for Y.

Default value: 0

Signals

The "curve-type-changed" signal

```
void        user_function                   (GtkCurve *curve,
                                             gpointer user_data);
```

Emitted when the curve type has been changed. The curve type can be changed explicitly with a call to gtk_curve_set_curve_type(). It is also changed as a side-effect of calling gtk_curve_reset() or gtk_curve_set_gamma().

curve : the object which received the signal.
user_data : user data set when the signal handler was connected.

See Also

GtkGammaCurve a subclass for editing gamma curves.

<< Special-purpose features **GtkGammaCurve >>**

GtkGammaCurve

GtkGammaCurve — a subclass of GtkCurve for editing gamma curves.

Synopsis

```
#include <gtk/gtk.h>

            GtkGammaCurve;
GtkWidget*  gtk_gamma_curve_new             (void);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkWidget
               +----GtkContainer
                     +----GtkBox
                           +----GtkVBox
                                 +----GtkGammaCurve
```

Implemented Interfaces

GtkGammaCurve implements AtkImplementorIface.

Description

Note

This widget is considered too specialized/little-used for GTK+, and will in the future be moved to some other package. If your application needs this widget, feel free to use it, as the widget does work and is useful in some applications; it's just not of general interest. However, we are not accepting new features for the widget, and it will eventually move out of the GTK+ distribution.

The GtkGammaCurve widget is a variant of GtkCurve specifically for editing gamma curves, which are used in graphics applications such as the Gimp.

The GtkGammaCurve widget shows a curve which the user can edit with the mouse just like a GtkCurve widget. On the right of the curve it also displays 5 buttons, 3 of which change between the 3 curve modes (spline, linear and free), and the other 2 set the curve to a particular gamma value, or reset it to a straight line.

Details

GtkGammaCurve

```
typedef struct _GtkGammaCurve GtkGammaCurve;
```

The GtkGammaCurve struct contains private data only, and should be accessed using the functions below.

---

gtk_gamma_curve_new ()

```
GtkWidget*  gtk_gamma_curve_new             (void);
```

Creates a new GtkGammaCurve.

Returns : a new GtkGammaCurve.

<< GtkCurve **GtkRuler >>**

GtkRuler

GtkRuler — Base class for horizontal or vertical rulers

Synopsis

```
#include <gtk/gtk.h>

            GtkRuler;
            GtkRulerMetric;
void        gtk_ruler_set_metric            (GtkRuler *ruler,
                                             GtkMetricType metric);
void        gtk_ruler_set_range             (GtkRuler *ruler,
                                             gdouble lower,
                                             gdouble upper,
                                             gdouble position,
                                             gdouble max_size);
GtkMetricType gtk_ruler_get_metric          (GtkRuler *ruler);
void        gtk_ruler_get_range             (GtkRuler *ruler,
                                             gdouble *lower,
                                             gdouble *upper,
                                             gdouble *position,
                                             gdouble *max_size);
```

Object Hierarchy

```
  GObject
   +----GtkObject
```

```
          +----GtkWidget
                +----GtkRuler
                      +----GtkHRuler
                      +----GtkVRuler
```

Implemented Interfaces

GtkRuler implements AtkImplementorIface.

Properties

```
  "lower"                 gdouble               : Read / Write
  "max-size"              gdouble               : Read / Write
  "position"              gdouble               : Read / Write
  "upper"                 gdouble               : Read / Write
```

Description

Note

This widget is considered too specialized/little-used for GTK+, and will in the future be moved to some other package. If your application needs this widget, feel free to use it, as the widget does work and is useful in some applications; it's just not of general interest. However, we are not accepting new features for the widget, and it will eventually move out of the GTK+ distribution.

The GTKRuler widget is a base class for horizontal and vertical rulers. Rulers are used to show the mouse pointer's location in a window. The ruler can either be horizontal or vertical on the window. Within the ruler a small triangle indicates the location of the mouse relative to the horizontal or vertical ruler. See GtkHRuler to learn how to create a new horizontal ruler. See GtkVRuler to learn how to create a new vertical ruler.

Details

GtkRuler

```
typedef struct _GtkRuler GtkRuler;
```

All distances are in 1/72nd's of an inch. (According to Adobe thats a point, but points are really 1/72.27 in.)

GtkRulerMetric

```
typedef struct {
  gchar *metric_name;
  gchar *abbrev;
  /* This should be points_per_unit. This is the size of the unit
   * in 1/72nd's of an inch and has nothing to do with screen pixels */
  gdouble pixels_per_unit;
  gdouble ruler_scale[10];
  gint subdivide[5];        /* five possible modes of subdivision */
} GtkRulerMetric;
```

This should be points_per_unit. This is the size of the unit in 1/72nd's of an inch and has nothing to do with screen pixels.

---

gtk_ruler_set_metric ()

```
void        gtk_ruler_set_metric            (GtkRuler *ruler,
                                             GtkMetricType metric);
```

This calls the GTKMetricType to set the ruler to units defined. Available units are GTK_PIXELS, GTK_INCHES, or GTK_CENTIMETERS. The default unit of measurement is GTK_PIXELS.

ruler : the gtkruler
metric : the unit of measurement

---

gtk_ruler_set_range ()

```
void        gtk_ruler_set_range             (GtkRuler *ruler,
                                             gdouble lower,
                                             gdouble upper,
                                             gdouble position,
                                             gdouble max_size);
```

This sets the range of the ruler using gfloat lower, gfloat upper, gfloat position, and gfloat max_size.

ruler : the gtkruler
lower : the upper limit of the ruler
upper : the lower limit of the ruler
position : the mark on the ruler
max_size : the maximum size of the ruler

---

gtk_ruler_get_metric ()

```
GtkMetricType gtk_ruler_get_metric          (GtkRuler *ruler);
```

Gets the units used for a GtkRuler. See `gtk_ruler_set_metric()`.

ruler : a GtkRuler
Returns : the units currently used for *ruler*

---

gtk_ruler_get_range ()

```
void        gtk_ruler_get_range             (GtkRuler *ruler,
                                             gdouble *lower,
                                             gdouble *upper,
                                             gdouble *position,
                                             gdouble *max_size);
```

Retrieves values indicating the range and current position of a GtkRuler. See

gtk_ruler_set_range().

ruler :	a GtkRuler
lower :	location to store lower limit of the ruler, or NULL
upper :	location to store upper limit of the ruler, or NULL
position :	location to store the current position of the mark on the ruler, or NULL
max_size :	location to store the maximum size of the ruler used when calculating the space to leave for the text, or NULL.

Properties

The "lower" property

```
"lower"                gdouble               : Read / Write
```

Lower limit of ruler.

Default value: 0

The "max-size" property

```
"max-size"             gdouble               : Read / Write
```

Maximum size of the ruler.

Default value: 0

The "position" property

```
"position"             gdouble               : Read / Write
```

Position of mark on the ruler.

Default value: 0

The "upper" property

```
  "upper"                gdouble               : Read / Write
```

Upper limit of ruler.

Default value: 0

See Also

GtkHRuler, GtkVRuler

<< GtkGammaCurve **GtkHRuler >>**

GtkHRuler

GtkHRuler — A horizontal ruler.

Synopsis

```
#include <gtk/gtk.h>

            GtkHRuler;
GtkWidget*  gtk_hruler_new                  (void);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkWidget
               +----GtkRuler
                     +----GtkHRuler
```

Implemented Interfaces

GtkHRuler implements AtkImplementorIface.

Description

Note

This widget is considered too specialized/little-used for GTK+, and will in the future be moved to some other package. If your application needs this widget, feel free to use it, as the widget does work and is useful in some applications; it's just not of general interest. However, we are not accepting new features for the widget, and it will eventually move out of the GTK+ distribution.

The HRuler widget is a widget arranged horizontally creating a ruler that is utilized around other widgets such as a text widget. The ruler is used to show the location of the mouse on the window and to show the size of the window in specified units. The available units of measurement are GTK_PIXELS, GTK_INCHES and GTK_CENTIMETERS. GTK_PIXELS is the default. rulers.

Details

GtkHRuler

```
typedef struct _GtkHRuler GtkHRuler;
```

The GtkHRuler struct contains private data and should be accessed with the functions below.

---

gtk_hruler_new ()

```
GtkWidget*  gtk_hruler_new                      (void);
```

Creates a new horizontal ruler.

Returns : a new GtkHRuler.

GtkVRuler

GtkVRuler — A vertical ruler.

Synopsis

```
#include <gtk/gtk.h>

            GtkVRuler;
GtkWidget*  gtk_vruler_new                  (void);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkWidget
               +----GtkRuler
                     +----GtkVRuler
```

Implemented Interfaces

GtkVRuler implements AtkImplementorIface.

Description

Note

> This widget is considered too specialized/little-used for GTK+, and will in the future be moved to some other package. If your application needs this widget, feel free to use it, as the widget does work and is useful in some applications; it's just not of general interest. However, we are not accepting new features for the widget, and it will eventually move out of the GTK+ distribution.

The VRuler widget is a widget arranged vertically creating a ruler that is utilized around other widgets such as a text widget. The ruler is used to show the location of the mouse on the window and to show the size of the window in specified units. The available units of measurement are GTK_PIXELS, GTK_INCHES and GTK_CENTIMETERS. GTK_PIXELS is the default. rulers.

Details

GtkVRuler

```
typedef struct _GtkVRuler GtkVRuler;
```

The GtkVRuler struct contains private data and should be accessed using the functions below.

---

gtk_vruler_new ()

```
GtkWidget*  gtk_vruler_new                    (void);
```

Creates a new vertical ruler

Returns : a new GtkVRuler.

<< GtkHRuler **Deprecated >>**

Deprecated

GtkCList - A multi-columned scrolling list widget
GtkCTree - A widget displaying a hierarchical tree
GtkCombo - A text entry field with a dropdown list
GtkItemFactory - A factory for menus
GtkList - Widget for packing a list of selectable items
GtkListItem - An item in a GtkList
GtkOldEditable - Base class for text-editing widgets
GtkOptionMenu - A widget used to choose from a list of valid choices
GtkPixmap - A widget displaying a graphical image or icon
GtkPreview - A widget to display RGB or grayscale data
GtkProgress - Base class for GtkProgressBar
GtkText - A text widget
GtkTipsQuery - Displays help about widgets in the user interface
GtkTree - A tree widget
GtkTreeItem - The widget used for items in a GtkTree

GtkCList

GtkCList — A multi-columned scrolling list widget

Synopsis

```
#include <gtk/gtk.h>


            GtkCList;
enum        GtkCellType;
enum        GtkButtonAction;
#define     GTK_CLIST_FLAGS                 (clist)
#define     GTK_CLIST_SET_FLAG              (clist,flag)
#define     GTK_CLIST_UNSET_FLAG            (clist,flag)
#define     GTK_CLIST_IN_DRAG               (clist)
#define     GTK_CLIST_ROW_HEIGHT_SET        (clist)
#define     GTK_CLIST_SHOW_TITLES           (clist)
#define     GTK_CLIST_ADD_MODE              (clist)
#define     GTK_CLIST_AUTO_SORT             (clist)
#define     GTK_CLIST_AUTO_RESIZE_BLOCKED   (clist)
#define     GTK_CLIST_REORDERABLE           (clist)
#define     GTK_CLIST_USE_DRAG_ICONS        (clist)
#define     GTK_CLIST_DRAW_DRAG_LINE        (clist)
#define     GTK_CLIST_DRAW_DRAG_RECT        (clist)
#define     GTK_CLIST_ROW                   (_glist_)
#define     GTK_CELL_TEXT                   (cell)
#define     GTK_CELL_PIXMAP                 (cell)
#define     GTK_CELL_PIXTEXT                (cell)
#define     GTK_CELL_WIDGET                 (cell)
gint        (*GtkCListCompareFunc)          (GtkCList *clist,
                                             gconstpointer ptr1,
                                             gconstpointer ptr2);
            GtkCListColumn;
            GtkCListRow;
            GtkCellText;
            GtkCellPixmap;
            GtkCellPixText;
```

```
            GtkCellWidget;
            GtkCell;
            GtkCListCellInfo;
            GtkCListDestInfo;
enum        GtkCListDragPos;
GtkWidget*  gtk_clist_new                   (gint columns);
GtkWidget*  gtk_clist_new_with_titles       (gint columns,
                                             gchar *titles[]);
void        gtk_clist_set_shadow_type       (GtkCList *clist,
                                             GtkShadowType type);
void        gtk_clist_set_selection_mode    (GtkCList *clist,
                                             GtkSelectionMode mode);
void        gtk_clist_freeze                (GtkCList *clist);
void        gtk_clist_thaw                  (GtkCList *clist);
void        gtk_clist_column_titles_show    (GtkCList *clist);
void        gtk_clist_column_titles_hide    (GtkCList *clist);
void        gtk_clist_column_title_active   (GtkCList *clist,
                                             gint column);
void        gtk_clist_column_title_passive  (GtkCList *clist,
                                             gint column);
void        gtk_clist_column_titles_active  (GtkCList *clist);
void        gtk_clist_column_titles_passive (GtkCList *clist);
void        gtk_clist_set_column_title      (GtkCList *clist,
                                             gint column,
                                             const gchar *title);
void        gtk_clist_set_column_widget     (GtkCList *clist,
                                             gint column,
                                             GtkWidget *widget);
void        gtk_clist_set_column_justification
                                            (GtkCList *clist,
                                             gint column,
                                             GtkJustification justification);
void        gtk_clist_set_column_visibility (GtkCList *clist,
                                             gint column,
                                             gboolean visible);
void        gtk_clist_set_column_resizeable (GtkCList *clist,
                                             gint column,
                                             gboolean resizeable);
void        gtk_clist_set_column_auto_resize
                                            (GtkCList *clist,
                                             gint column,
                                             gboolean auto_resize);
gint        gtk_clist_optimal_column_width  (GtkCList *clist,
                                             gint column);
```

```
void        gtk_clist_set_column_width      (GtkCList *clist,
                                             gint column,
                                             gint width);
void        gtk_clist_set_column_min_width  (GtkCList *clist,
                                             gint column,
                                             gint min_width);
void        gtk_clist_set_column_max_width  (GtkCList *clist,
                                             gint column,
                                             gint max_width);
void        gtk_clist_set_row_height        (GtkCList *clist,
                                             guint height);
void        gtk_clist_moveto                (GtkCList *clist,
                                             gint row,
                                             gint column,
                                             gfloat row_align,
                                             gfloat col_align);
GtkVisibility gtk_clist_row_is_visible      (GtkCList *clist,
                                             gint row);
GtkCellType gtk_clist_get_cell_type         (GtkCList *clist,
                                             gint row,
                                             gint column);
void        gtk_clist_set_text              (GtkCList *clist,
                                             gint row,
                                             gint column,
                                             const gchar *text);
gint        gtk_clist_get_text              (GtkCList *clist,
                                             gint row,
                                             gint column,
                                             gchar **text);
void        gtk_clist_set_pixmap            (GtkCList *clist,
                                             gint row,
                                             gint column,
                                             GdkPixmap *pixmap,
                                             GdkBitmap *mask);
gint        gtk_clist_get_pixmap            (GtkCList *clist,
                                             gint row,
                                             gint column,
                                             GdkPixmap **pixmap,
                                             GdkBitmap **mask);
void        gtk_clist_set_pixtext           (GtkCList *clist,
                                             gint row,
                                             gint column,
                                             const gchar *text,
                                             guint8 spacing,
```

```
                                        GdkPixmap *pixmap,
                                        GdkBitmap *mask);
gint        gtk_clist_get_pixtext       (GtkCList *clist,
                                        gint row,
                                        gint column,
                                        gchar **text,
                                        guint8 *spacing,
                                        GdkPixmap **pixmap,
                                        GdkBitmap **mask);
void        gtk_clist_set_foreground    (GtkCList *clist,
                                        gint row,
                                        const GdkColor *color);
void        gtk_clist_set_background    (GtkCList *clist,
                                        gint row,
                                        const GdkColor *color);
void        gtk_clist_set_cell_style    (GtkCList *clist,
                                        gint row,
                                        gint column,
                                        GtkStyle *style);
GtkStyle*   gtk_clist_get_cell_style    (GtkCList *clist,
                                        gint row,
                                        gint column);
void        gtk_clist_set_row_style     (GtkCList *clist,
                                        gint row,
                                        GtkStyle *style);
GtkStyle*   gtk_clist_get_row_style     (GtkCList *clist,
                                        gint row);
void        gtk_clist_set_shift         (GtkCList *clist,
                                        gint row,
                                        gint column,
                                        gint vertical,
                                        gint horizontal);
void        gtk_clist_set_selectable    (GtkCList *clist,
                                        gint row,
                                        gboolean selectable);
gboolean    gtk_clist_get_selectable    (GtkCList *clist,
                                        gint row);
gint        gtk_clist_prepend           (GtkCList *clist,
                                        gchar *text[]);
gint        gtk_clist_append            (GtkCList *clist,
                                        gchar *text[]);
gint        gtk_clist_insert            (GtkCList *clist,
                                        gint row,
                                        gchar *text[]);
```

```
void        gtk_clist_remove                (GtkCList *clist,
                                             gint row);
void        gtk_clist_set_row_data          (GtkCList *clist,
                                             gint row,
                                             gpointer data);
void        gtk_clist_set_row_data_full     (GtkCList *clist,
                                             gint row,
                                             gpointer data,
                                             GtkDestroyNotify destroy);
gpointer    gtk_clist_get_row_data          (GtkCList *clist,
                                             gint row);
gint        gtk_clist_find_row_from_data    (GtkCList *clist,
                                             gpointer data);
void        gtk_clist_select_row            (GtkCList *clist,
                                             gint row,
                                             gint column);
void        gtk_clist_unselect_row          (GtkCList *clist,
                                             gint row,
                                             gint column);
void        gtk_clist_undo_selection        (GtkCList *clist);
void        gtk_clist_clear                 (GtkCList *clist);
gint        gtk_clist_get_selection_info    (GtkCList *clist,
                                             gint x,
                                             gint y,
                                             gint *row,
                                             gint *column);
void        gtk_clist_select_all            (GtkCList *clist);
void        gtk_clist_unselect_all          (GtkCList *clist);
void        gtk_clist_swap_rows             (GtkCList *clist,
                                             gint row1,
                                             gint row2);
void        gtk_clist_set_compare_func      (GtkCList *clist,
                                             GtkCListCompareFunc cmp_func);
void        gtk_clist_set_sort_column       (GtkCList *clist,
                                             gint column);
void        gtk_clist_set_sort_type         (GtkCList *clist,
                                             GtkSortType sort_type);
void        gtk_clist_sort                  (GtkCList *clist);
void        gtk_clist_set_auto_sort         (GtkCList *clist,
                                             gboolean auto_sort);
gint        gtk_clist_columns_autosize      (GtkCList *clist);
gchar*      gtk_clist_get_column_title      (GtkCList *clist,
                                             gint column);
GtkWidget*  gtk_clist_get_column_widget     (GtkCList *clist,
```

```
                                                gint column);
GtkAdjustment* gtk_clist_get_hadjustment    (GtkCList *clist);
GtkAdjustment* gtk_clist_get_vadjustment    (GtkCList *clist);
void        gtk_clist_row_move              (GtkCList *clist,
                                             gint source_row,
                                             gint dest_row);
void        gtk_clist_set_button_actions    (GtkCList *clist,
                                             guint button,
                                             guint8 button_actions);
void        gtk_clist_set_hadjustment       (GtkCList *clist,
                                             GtkAdjustment *adjustment);
void        gtk_clist_set_reorderable       (GtkCList *clist,
                                             gboolean reorderable);
void        gtk_clist_set_use_drag_icons    (GtkCList *clist,
                                             gboolean use_icons);
void        gtk_clist_set_vadjustment       (GtkCList *clist,
                                             GtkAdjustment *adjustment);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkWidget
               +----GtkContainer
                     +----GtkCList
                           +----GtkCTree
```

Implemented Interfaces

GtkCList implements AtkImplementorIface.

Properties

```
  "n-columns"           guint                : Read / Write / Construct Only
  "reorderable"         gboolean             : Read / Write
  "row-height"          guint                : Read / Write
  "selection-mode"      GtkSelectionMode     : Read / Write
  "shadow-type"         GtkShadowType        : Read / Write
```

```
  "sort-type"            GtkSortType          : Read / Write
  "titles-active"        gboolean             : Read / Write
  "use-drag-icons"       gboolean             : Read / Write
```

Signal Prototypes

```
"abort-column-resize"
            void        user_function      (GtkCList *clist,
                                            gpointer user_data);
"click-column"
            void        user_function      (GtkCList *clist,
                                            gint column,
                                            gpointer user_data);
"end-selection"
            void        user_function      (GtkCList *clist,
                                            gpointer user_data);
"extend-selection"
            void        user_function      (GtkCList *clist,
                                            GtkScrollType scroll_type,
                                            gfloat position,
                                            gboolean auto_start_selection,
                                            gpointer user_data);
"resize-column"
            void        user_function      (GtkCList *clist,
                                            gint column,
                                            gint width,
                                            gpointer user_data);
"row-move"  void        user_function      (GtkCList *clist,
                                            gint arg1,
                                            gint arg2,
                                            gpointer user_data);
"scroll-horizontal"
            void        user_function      (GtkCList *clist,
                                            GtkScrollType scroll_type,
                                            gfloat position,
                                            gpointer user_data);
"scroll-vertical"
            void        user_function      (GtkCList *clist,
                                            GtkScrollType scroll_type,
                                            gfloat position,
                                            gpointer user_data);
"select-all"
```

```
            void        user_function      (GtkCList *clist,
                                            gpointer user_data);
"select-row"
            void        user_function      (GtkCList *clist,
                                            gint row,
                                            gint column,
                                            GdkEventButton *event,
                                            gpointer user_data);
"set-scroll-adjustments"
            void        user_function      (GtkCList *clist,
                                            GtkAdjustment *arg1,
                                            GtkAdjustment *arg2,
                                            gpointer user_data);
"start-selection"
            void        user_function      (GtkCList *clist,
                                            gpointer user_data);
"toggle-add-mode"
            void        user_function      (GtkCList *clist,
                                            gpointer user_data);
"toggle-focus-row"
            void        user_function      (GtkCList *clist,
                                            gpointer user_data);
"undo-selection"
            void        user_function      (GtkCList *clist,
                                            gpointer user_data);
"unselect-all"
            void        user_function      (GtkCList *clist,
                                            gpointer user_data);
"unselect-row"
            void        user_function      (GtkCList *clist,
                                            gint row,
                                            gint column,
                                            GdkEventButton *event,
                                            gpointer user_data);
```

Description

The GtkCList widget is a very useful multi-columned scrolling list. It can display data in nicely aligned vertical columns, with titles at the top of the list.

GtkCList has been deprecated since GTK+ 2.0 and should not be used in newly written code. Use GtkTreeView instead.

Details

GtkCList

```
typedef struct _GtkCList GtkCList;
```

Warning

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

This is the embodiment of the GtkCList widget. This structure contains only private data, and should be accessed only via the CList API.

---

enum GtkCellType

```
typedef enum
{
  GTK_CELL_EMPTY,
  GTK_CELL_TEXT,
  GTK_CELL_PIXMAP,
  GTK_CELL_PIXTEXT,
  GTK_CELL_WIDGET
} GtkCellType;
```

Warning

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

Identifies the type of element in the current cell of the CList. Cells can contain text, pixmaps, or both. Unfortunately support for `GTK_CELL_WIDGET` was never completed.

---

enum GtkButtonAction

```
typedef enum
{
  GTK_BUTTON_IGNORED = 0,
  GTK_BUTTON_SELECTS = 1 << 0,
  GTK_BUTTON_DRAGS   = 1 << 1,
  GTK_BUTTON_EXPANDS = 1 << 2
} GtkButtonAction;
```

Warning

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

Values for specifying what mouse button events a CList will react to.

GTK_CLIST_FLAGS()

```
#define GTK_CLIST_FLAGS(clist)              (GTK_CLIST (clist)->flags)
```

Warning

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

Reads the current flags of the specified CList.

`clist` : The GtkCList widget from which to get the flags

GTK_CLIST_SET_FLAG()

```
#define GTK_CLIST_SET_FLAG(clist,flag)      (GTK_CLIST_FLAGS (clist) |= (GTK_ ##
flag))
```

Warning

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

A macro to set a particular flag for the specified CList.

`clist` : The GtkCList widget to affect.
`flag` : A single GtkCList flag to set. NOTE: Do not add the GTK_ prefix.

GTK_CLIST_UNSET_FLAG()

```
#define GTK_CLIST_UNSET_FLAG(clist,flag)    (GTK_CLIST_FLAGS (clist) &= ~(GTK_ ##
```

```
flag))
```

Warning

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

A macro to clear a particular flag for the specified CList.

clist : The GtkCList widget to affect.

flag : A single GtkCList flag to clear. NOTE: Do not add the GTK_ prefix.

GTK_CLIST_IN_DRAG()

```
#define GTK_CLIST_IN_DRAG(clist)          (GTK_CLIST_FLAGS (clist) &
GTK_CLIST_IN_DRAG)
```

Warning

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

A macro to check whether the GtkCList is in "drag mode."

clist : The GtkCList to check.

GTK_CLIST_ROW_HEIGHT_SET()

```
#define GTK_CLIST_ROW_HEIGHT_SET(clist)    (GTK_CLIST_FLAGS (clist) &
GTK_CLIST_ROW_HEIGHT_SET)
```

Warning

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

A macro to check whether the GtkCList's row height is set.

clist : The GtkCList to check.

GTK_CLIST_SHOW_TITLES()

```
#define GTK_CLIST_SHOW_TITLES(clist)       (GTK_CLIST_FLAGS (clist) &
GTK_CLIST_SHOW_TITLES)
```

Warning

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

A macro to check whether the flag for showing the widget's column titles is set.

`clist` : The GtkCList widget to check.

GTK_CLIST_ADD_MODE()

```
#define GTK_CLIST_ADD_MODE(clist)          (GTK_CLIST_FLAGS (clist) &
GTK_CLIST_ADD_MODE)
```

Warning

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

A macro to test whether the CList is in "add mode."

`clist` : The GtkCList widget to check.

GTK_CLIST_AUTO_SORT()

```
#define GTK_CLIST_AUTO_SORT(clist)         (GTK_CLIST_FLAGS (clist) &
GTK_CLIST_AUTO_SORT)
```

Warning

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

A macro to test whether the CList has automatic sorting switched on.

clist : The GtkCList widget to check.

GTK_CLIST_AUTO_RESIZE_BLOCKED()

```
#define GTK_CLIST_AUTO_RESIZE_BLOCKED(clist) (GTK_CLIST_FLAGS (clist) &
GTK_CLIST_AUTO_RESIZE_BLOCKED)
```

Warning

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

A macro to check if automatic resizing of columns is blocked.

clist : The GtkCList widget to check.

GTK_CLIST_REORDERABLE()

```
#define GTK_CLIST_REORDERABLE(clist)       (GTK_CLIST_FLAGS (clist) &
GTK_CLIST_REORDERABLE)
```

Warning

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

A macro to test if the CList's columns are re-orderable

clist : The GtkCList widget to check.

GTK_CLIST_USE_DRAG_ICONS()

```
#define GTK_CLIST_USE_DRAG_ICONS(clist)    (GTK_CLIST_FLAGS (clist) &
GTK_CLIST_USE_DRAG_ICONS)
```

Warning

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

A macro to check if the USE_DRAG_ICONS property is enabled.

clist : The GtkCList widget to check.

GTK_CLIST_DRAW_DRAG_LINE()

```
#define GTK_CLIST_DRAW_DRAG_LINE(clist)    (GTK_CLIST_FLAGS (clist) &
GTK_CLIST_DRAW_DRAG_LINE)
```

Warning

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

A macro to check if the DRAW_DRAG_LINE property is enabled.

clist : The GtkCList widget to check.

GTK_CLIST_DRAW_DRAG_RECT()

```
#define GTK_CLIST_DRAW_DRAG_RECT(clist)    (GTK_CLIST_FLAGS (clist) &
GTK_CLIST_DRAW_DRAG_RECT)
```

Warning

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

A macro to check if the DRAW_DRAG_RECT property is enabled.

clist : The GtkCList widget to check.

GTK_CLIST_ROW()

```
#define GTK_CLIST_ROW(_glist_) ((GtkCListRow *)((_glist_)->data))
```

Warning

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

A macro to cast a GList element to a CListRow pointer.

*_glist_* : The GList element to convert.

GTK_CELL_TEXT()

```
#define GTK_CELL_TEXT(cell)     (((GtkCellText *) &(cell)))
```

Warning

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

A macro to cast a generic GtkCList cell item to a GtkCellText pointer.

cell : The GtkCList cell item to convert.

GTK_CELL_PIXMAP()

```
#define GTK_CELL_PIXMAP(cell)   (((GtkCellPixmap *) &(cell)))
```

Warning

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

A macro to cast a generic GtkCList cell item to a GtkCellPixmap pointer.

cell : The GtkCList cell item to convert.

GTK_CELL_PIXTEXT()

```
#define GTK_CELL_PIXTEXT(cell)  (((GtkCellPixText *) &(cell)))
```

Warning

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

A macro to cast a generic GtkCList cell item to a GtkCellPixText pointer.

cell : The GtkCList cell item to convert.

GTK_CELL_WIDGET()

```
#define GTK_CELL_WIDGET(cell)   (((GtkCellWidget *) &(cell)))
```

Warning

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

A macro to cast a generic GtkCList cell item to a GtkCellWidget pointer.

cell : The GtkCList cell item to convert.

GtkCListCompareFunc ()

```
gint        (*GtkCListCompareFunc)          (GtkCList *clist,
                                             gconstpointer ptr1,
                                             gconstpointer ptr2);
```

Warning

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

Function prototype for the compare function callback.

clist : The GtkCList that is affected.

ptr1 : A gconstpointer to the first node to compare.

ptr2 : A gconstpointer to the second node to compare.

Returns : 0 if the nodes are equal, less than 0 if the first node should come before the second, and greater than 1 if the second come before the first.

GtkCListColumn

```
typedef struct {
  gchar *title;
  GdkRectangle area;

  GtkWidget *button;
  GdkWindow *window;

  gint width;
  gint min_width;
  gint max_width;
  GtkJustification justification;

  guint visible        : 1;
  guint width_set      : 1;
  guint resizeable     : 1;
  guint auto_resize    : 1;
  guint button_passive : 1;
} GtkCListColumn;
```

Warning

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

A structure that the GtkCList widget uses to keep track of information about its columns.

GtkCListRow

```
typedef struct {
  GtkCell *cell;
  GtkStateType state;

  GdkColor foreground;
  GdkColor background;

  GtkStyle *style;

  gpointer data;
  GtkDestroyNotify destroy;

  guint fg_set     : 1;
  guint bg_set     : 1;
  guint selectable : 1;
} GtkCListRow;
```

Warning

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

A structure that the GtkCList widget uses to keep track of information about its rows.

GtkCellText

```
typedef struct {
  GtkCellType type;

  gint16 vertical;
  gint16 horizontal;

  GtkStyle *style;

  gchar *text;
} GtkCellText;
```

Warning

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

A structure that the GtkCList widget uses to keep track of GtkCList cells that contain text.

GtkCellPixmap

```
typedef struct {
  GtkCellType type;

  gint16 vertical;
  gint16 horizontal;

  GtkStyle *style;

  GdkPixmap *pixmap;
  GdkBitmap *mask;
} GtkCellPixmap;
```

Warning

Warning

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

A structure that the GtkCList widget uses to keep track of GtkCList cells that contain a GdkPixmap.

GtkCellPixText

```
typedef struct {
  GtkCellType type;

  gint16 vertical;
  gint16 horizontal;

  GtkStyle *style;

  gchar *text;
  guint8 spacing;
  GdkPixmap *pixmap;
  GdkBitmap *mask;
} GtkCellPixText;
```

Warning

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

A structure that the GtkCList widget uses to keep track of GtkCList cells that contain a combination of text and a GdkPixmap.

GtkCellWidget

```
typedef struct {
  GtkCellType type;

  gint16 vertical;
  gint16 horizontal;

  GtkStyle *style;

  GtkWidget *widget;
} GtkCellWidget;
```

Warning

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

A structure that the GtkCList widget uses to keep track of GtkCList cells that contain another widget.

GtkCell

```
typedef struct {
  GtkCellType type;

  gint16 vertical;
  gint16 horizontal;

  GtkStyle *style;

  union {
    gchar *text;

    struct {
      GdkPixmap *pixmap;
      GdkBitmap *mask;
    } pm;

    struct {
      gchar *text;
      guint8 spacing;
      GdkPixmap *pixmap;
      GdkBitmap *mask;
    } pt;

    GtkWidget *widget;
  } u;
} GtkCell;
```

Warning

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

A generic structure that the GtkCList widget uses to keep track of the contents of each of its cells.

GtkCListCellInfo

```
typedef struct {
  gint row;
  gint column;
} GtkCListCellInfo;
```

Warning

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

A simple structure that the GtkCList widget uses to keep track of the location of a cell.

GtkCListDestInfo

```
typedef struct {
  GtkCListCellInfo cell;
  GtkCListDragPos  insert_pos;
} GtkCListDestInfo;
```

Warning

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

A simple structure that the GtkCList widget uses to track a cell for a drag operation.

---

enum GtkCListDragPos

```
typedef enum
{
  GTK_CLIST_DRAG_NONE,
  GTK_CLIST_DRAG_BEFORE,
  GTK_CLIST_DRAG_INTO,
  GTK_CLIST_DRAG_AFTER
} GtkCListDragPos;
```

Warning

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

An enumeration for drag operations.

---

gtk_clist_new ()

```
GtkWidget*  gtk_clist_new                   (gint columns);
```

Warning

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

Creates a new GtkCList widget for use.

columns : The number of columns the GtkCList should have.

Returns : A pointer to a new GtkCList object.

---

gtk_clist_new_with_titles ()

```
GtkWidget*  gtk_clist_new_with_titles       (gint columns,
                                             gchar *titles[]);
```

Warning

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

Creates a new GtkCList widget with column titles for use.

columns : The number of columns the GtkCList should have.

titles : A string array of titles for the widget. There should be enough strings in the array for the specified number of columns.

Returns : A pointer to a new GtkCList object.

---

gtk_clist_set_shadow_type ()

```
void        gtk_clist_set_shadow_type       (GtkCList *clist,
                                             GtkShadowType type);
```

Warning

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

Sets the shadow type for the specified CList. Changing this value will cause the GtkCList to update its visuals.

clist : The GtkCList to affect.
type : The GtkShadowType desired.

---

gtk_clist_set_selection_mode ()

```
void        gtk_clist_set_selection_mode    (GtkCList *clist,
                                             GtkSelectionMode mode);
```

Warning

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

Sets the selection mode for the specified CList. This allows you to set whether only one or more than one item can be selected at a time in the widget. Note that setting the widget's selection mode to one of GTK_SELECTION_BROWSE or GTK_SELECTION_SINGLE will cause all the items in the GtkCList to become deselected.

clist : The GtkCList to affect.
mode : The GtkSelectionMode type to set for this CList.

---

gtk_clist_freeze ()

```
void        gtk_clist_freeze                (GtkCList *clist);
```

Warning

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

Causes the GtkCList to stop updating its visuals until a matching call to `gtk_clist_thaw()` is made. This function is useful if a lot of changes will be made to the widget that may cause a lot of visual updating to occur. Note that calls to `gtk_clist_freeze()` can be nested.

clist : The GtkCList to freeze.

---

gtk_clist_thaw ()

```
void        gtk_clist_thaw                  (GtkCList *clist);
```

Warning

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

Causes the specified GtkCList to allow visual updates.

clist : The GtkCList to thaw.

---

gtk_clist_column_titles_show ()

```
void        gtk_clist_column_titles_show    (GtkCList *clist);
```

Warning

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

This function causes the GtkCList to show its column titles, if they are not already showing.

clist : The GtkCList to affect.

---

gtk_clist_column_titles_hide ()

```
void        gtk_clist_column_titles_hide    (GtkCList *clist);
```

Warning

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

Causes the GtkCList to hide its column titles, if they are currently showing.

clist : The GtkCList to affect.

---

gtk_clist_column_title_active ()

```
void        gtk_clist_column_title_active   (GtkCList *clist,
                                             gint column);
```

Warning

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

Sets the specified column in the GtkCList to become selectable. You can then respond to events from the user clicking on a title button, and take appropriate action.

clist : The GtkCList to affect.
column : The column to make active, counting from 0.

---

gtk_clist_column_title_passive ()

```
void        gtk_clist_column_title_passive  (GtkCList *clist,
                                             gint column);
```

Warning

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

Causes the specified column title button to become passive, i.e., does not respond to events, such as the user clicking on it.

clist : The GtkCList to affect.
column : The column to make passive, counting from 0.

---

gtk_clist_column_titles_active ()

```
void        gtk_clist_column_titles_active  (GtkCList *clist);
```

Warning

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

Causes all column title buttons to become active. This is the same as calling `gtk_clist_column_title_active()` for each column.

`clist` : The GtkCList to affect.

---

gtk_clist_column_titles_passive ()

```
void        gtk_clist_column_titles_passive (GtkCList *clist);
```

Warning

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

Causes all column title buttons to become passive. This is the same as calling `gtk_clist_column_title_passive()` for each column.

`clist` : The GtkCList to affect.

---

gtk_clist_set_column_title ()

```
void        gtk_clist_set_column_title      (GtkCList *clist,
                                             gint column,
                                             const gchar *title);
```

Warning

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

Sets the title for the specified column.

`clist` : The GtkCList to affect.
`column` : The column whose title should be changed.
`title` : A string to be the column's title.

---

gtk_clist_set_column_widget ()

```
void        gtk_clist_set_column_widget     (GtkCList *clist,
                                             gint column,
                                             GtkWidget *widget);
```

Warning

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

Sets a widget to be used as the specified column's title. This can be used to place a pixmap or something else as the column title, instead of the standard text.

`clist` : The GtkCList to affect.
`column` : The column whose title should be a widget.
`widget` : A pointer to a previously create widget.

---

gtk_clist_set_column_justification ()

```
void        gtk_clist_set_column_justification
                                            (GtkCList *clist,
                                             gint column,
                                             GtkJustification justification);
```

Warning

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

Sets the justification to be used for all text in the specified column.

`clist` : The GtkCList to affect.
`column` : The column which should be affected.
`justification` : A GtkJustification value for the column.

---

gtk_clist_set_column_visibility ()

```
void        gtk_clist_set_column_visibility (GtkCList *clist,
                                             gint column,
                                             gboolean visible);
```

Warning

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

Allows you to set whether a specified column in the GtkCList should be hidden or shown. Note that at least one column must always be showing, so attempting to hide the last visible column will be ignored.

clist : The GtkCList to affect.
column : The column to set visibility.
visible : TRUE or FALSE.

---

gtk_clist_set_column_resizeable ()

```
void        gtk_clist_set_column_resizeable (GtkCList *clist,
                                             gint column,
                                             gboolean resizeable);
```

Warning

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

Lets you specify whether a specified column should be resizeable by the user. Note that turning on resizeability for the column will automatically shut off auto-resizing, but turning off resizeability will NOT turn on auto-resizing. This must be done manually via a call to `gtk_clist_set_column_auto_resize()`.

clist : The GtkCList to affect.
column : The column on which to set resizeability.
resizeable : TRUE or FALSE.

---

gtk_clist_set_column_auto_resize ()

```
void        gtk_clist_set_column_auto_resize
                                            (GtkCList *clist,
                                             gint column,
```

```
                                                gboolean auto_resize);
```

Warning

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

Lets you specify whether a column should be automatically resized by the widget when data is added or removed. Enabling auto-resize on a column explicity disallows user-resizing of the column.

clist : The GtkCList to affect.
column : The column on which to set auto-resizing.
auto_resize : TRUE or FALSE.

---

gtk_clist_optimal_column_width ()

```
gint        gtk_clist_optimal_column_width  (GtkCList *clist,
                                             gint column);
```

Warning

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

Gets the required width in pixels that is needed to show everything in the specified column.

clist : The GtkCList to check.
column : The column to check.
Returns : The required width in pixels for the column.

---

gtk_clist_set_column_width ()

```
void        gtk_clist_set_column_width      (GtkCList *clist,
                                             gint column,
                                             gint width);
```

Warning

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

Causes the column specified for the GtkCList to be set to a specified width.

clist : The GtkCList to affect.
column : The column to set the width.
width : The width, in pixels.

---

gtk_clist_set_column_min_width ()

```
void        gtk_clist_set_column_min_width  (GtkCList *clist,
                                             gint column,
                                             gint min_width);
```

Warning

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

Causes the column specified to have a minimum width, preventing the user from resizing it smaller than that specified.

clist : The GtkCList to affect.
column : The column to set the minimum width.
min_width : The width, in pixels.

---

gtk_clist_set_column_max_width ()

```
void        gtk_clist_set_column_max_width  (GtkCList *clist,
                                             gint column,
                                             gint max_width);
```

Warning

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

Causes the column specified to have a maximum width, preventing the user from resizing it larger than that specified.

clist : The GtkCList to affect.
column : The column to set the maximum width.
max_width : The width, in pixels.

gtk_clist_set_row_height ()

```
void        gtk_clist_set_row_height        (GtkCList *clist,
                                             guint height);
```

Warning

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

Causes the GtkCList to have a specified height for its rows. Setting the row height to 0 allows the GtkCList to adjust automatically to data in the row.

clist : The GtkCList to affect.
height : The height, in pixels.

---

gtk_clist_moveto ()

```
void        gtk_clist_moveto                (GtkCList *clist,
                                             gint row,
                                             gint column,
                                             gfloat row_align,
                                             gfloat col_align);
```

Warning

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

Tells the CList widget to visually move to the specified row and column.

clist : The GtkCList to affect.
row : The row to which to move.
column : The column to which to move.
row_align : A value between 0 and 1 that describes the positioning of the row in relation to the viewable area of the CList's contents.
col_align : A value between 0 and 1 that describes the positioning of the column in relation to the viewable area of the CList's contents.

---

gtk_clist_row_is_visible ()

```
GtkVisibility gtk_clist_row_is_visible      (GtkCList *clist,
                                             gint row);
```

Warning

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

Checks how the specified row is visible.

clist : The GtkCList to affect.
row : The row to query.
Returns : A GtkVisibility value that tells you how the row is visible.

---

gtk_clist_get_cell_type ()

```
GtkCellType gtk_clist_get_cell_type         (GtkCList *clist,
                                             gint row,
                                             gint column);
```

Warning

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

Checks the type of cell at the location specified.

clist : The GtkCList to affect.
row : The row of the cell.
column : The column of the cell.
Returns : A GtkCellType value describing the cell.

---

gtk_clist_set_text ()

```
void        gtk_clist_set_text              (GtkCList *clist,
                                             gint row,
                                             gint column,
```

```
                                                const gchar *text);
```

Warning

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

Sets the displayed text in the specified cell.

clist : The GtkCList to affect.
row : The row of the cell.
column : The column of the cell.
text : The text to set in the cell.

---

gtk_clist_get_text ()

```
gint        gtk_clist_get_text              (GtkCList *clist,
                                             gint row,
                                             gint column,
                                             gchar **text);
```

Warning

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

Gets the text for the specified cell.

clist : The GtkCList to affect.
row : The row to query.
column : The column to query.
text : A pointer to a pointer to store the text.
Returns : 1 if the cell's text could be retrieved, 0 otherwise.

---

gtk_clist_set_pixmap ()

```
void        gtk_clist_set_pixmap            (GtkCList *clist,
                                             gint row,
                                             gint column,
                                             GdkPixmap *pixmap,
```

```
                                                GdkBitmap *mask);
```

Warning

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

Sets a pixmap for the specified cell.

clist : The GtkCList to affect.
row : The row of the cell.
column : The column of the cell.
pixmap : A pointer to a GdkPixmap to place in the cell.
mask : A pointer to a GdkBitmap mask for the cell.

---

gtk_clist_get_pixmap ()

```
gint        gtk_clist_get_pixmap            (GtkCList *clist,
                                             gint row,
                                             gint column,
                                             GdkPixmap **pixmap,
                                             GdkBitmap **mask);
```

Warning

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

Gets the pixmap and bitmap mask of the specified cell. The returned mask value can be NULL.

clist : The GtkCList to affect.
row : The row of the cell.
column : The column of the cell.
pixmap : A pointer to a pointer to store the cell's GdkPixmap.
mask : A pointer to a pointer to store the cell's GdkBitmap mask.
Returns : 1 if the cell's pixmap could be retrieved, 0 otherwise.

---

gtk_clist_set_pixtext ()

```

```

```
void        gtk_clist_set_pixtext           (GtkCList *clist,
                                             gint row,
                                             gint column,
                                             const gchar *text,
                                             guint8 spacing,
                                             GdkPixmap *pixmap,
                                             GdkBitmap *mask);
```

Warning

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

Sets text and a pixmap/bitmap on the specified cell.

clist :	The GtkCList to affect.
row :	The row of the cell.
column :	The column of the cell.
text :	The text to set in the cell.
spacing :	The spacing between the cell's text and pixmap.
pixmap :	A pointer to a GdkPixmap for the cell.
mask :	A pointer to a GdkBitmap mask for the cell.

---

gtk_clist_get_pixtext ()

```
gint        gtk_clist_get_pixtext           (GtkCList *clist,
                                             gint row,
                                             gint column,
                                             gchar **text,
                                             guint8 *spacing,
                                             GdkPixmap **pixmap,
                                             GdkBitmap **mask);
```

Warning

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

Gets the text, pixmap and bitmap mask for the specified cell.

clist :	The GtkCList to affect.

row : The row to query.
column : The column to query.
text : A pointer to a pointer to store the text.
spacing : A pointer to a guint8 to store the spacing.
pixmap : A pointer to a GdkPixmap pointer to store the cell's pixmap.
mask : A pointer to a GdkBitmap pointer to store the cell's bitmap mask.
Returns : 1 if the retrieval was successful, 0 otherwise.

---

gtk_clist_set_foreground ()

```
void        gtk_clist_set_foreground        (GtkCList *clist,
                                             gint row,
                                             const GdkColor *color);
```

Warning

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

Sets the foreground color for the specified row.

clist : The GtkCList to affect.
row : The row to affect.
color : A pointer to a GdkColor structure.

---

gtk_clist_set_background ()

```
void        gtk_clist_set_background        (GtkCList *clist,
                                             gint row,
                                             const GdkColor *color);
```

Warning

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

Sets the background color for the specified row.

clist : The GtkCList to affect.

row : The row to affect.
color : A pointer to a GdkColor structure.

---

gtk_clist_set_cell_style ()

```
void        gtk_clist_set_cell_style        (GtkCList *clist,
                                             gint row,
                                             gint column,
                                             GtkStyle *style);
```

Warning

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

Sets the style for the specified cell.

clist : The GtkCList to affect.
row : The row of the cell.
column : The column of the cell.
style : A pointer to a GtkStyle structure.

---

gtk_clist_get_cell_style ()

```
GtkStyle*   gtk_clist_get_cell_style        (GtkCList *clist,
                                             gint row,
                                             gint column);
```

Warning

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

Gets the current style of the specified cell.

clist : The GtkCList to affect.
row : The row of the cell.
column : The column of the cell.
Returns : A GtkStyle object.

gtk_clist_set_row_style ()

```
void        gtk_clist_set_row_style         (GtkCList *clist,
                                             gint row,
                                             GtkStyle *style);
```

Warning

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

Sets the style for all cells in the specified row.

clist : The GtkCList to affect.
row : The row to affect.
style : A pointer to a GtkStyle to set.

gtk_clist_get_row_style ()

```
GtkStyle*   gtk_clist_get_row_style         (GtkCList *clist,
                                             gint row);
```

Warning

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

Gets the style set for the specified row.

clist : The GtkCList to affect.
row : The row to query.
Returns : The GtkStyle of the row.

gtk_clist_set_shift ()

```
void        gtk_clist_set_shift             (GtkCList *clist,
                                             gint row,
                                             gint column,
```

```
                                                gint vertical,
                                                gint horizontal);
```

Warning

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

Sets the vertical and horizontal shift of the specified cell.

clist :	The GtkCList to affect.
row :	The row of the cell.
column :	The column of the cell.
vertical :	The value to set for the vertical shift.
horizontal :	The value to set for the vertical shift.

---

gtk_clist_set_selectable ()

```
void        gtk_clist_set_selectable        (GtkCList *clist,
                                             gint row,
                                             gboolean selectable);
```

Warning

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

Sets whether the specified row is selectable or not.

clist :	The GtkCList to affect.
row :	The row to affect.
selectable :	TRUE or FALSE.

---

gtk_clist_get_selectable ()

```
gboolean    gtk_clist_get_selectable        (GtkCList *clist,
                                             gint row);
```

Warning

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

Gets whether the specified row is selectable or not.

clist : The GtkCList to affect.
row : The row to query.
Returns : A gboolean value.

---

gtk_clist_prepend ()

```
gint        gtk_clist_prepend               (GtkCList *clist,
                                             gchar *text[]);
```

Warning

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

Adds a row to the CList at the top.

clist : The GtkCList to affect.
text : An array of strings to add.
Returns : The number of the row added.

---

gtk_clist_append ()

```
gint        gtk_clist_append                (GtkCList *clist,
                                             gchar *text[]);
```

Warning

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

Adds a row to the CList at the bottom.

clist : The GtkCList to affect.
text : An array of strings to add.
Returns : The number of the row added.

gtk_clist_insert ()

```
gint        gtk_clist_insert                (GtkCList *clist,
                                             gint row,
                                             gchar *text[]);
```

Warning

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

Adds a row of text to the CList at the specified position.

clist : The GtkCList to affect.
row : The row where the text should be inserted.
text : An array of string to add.
Returns : The number of the row added.

gtk_clist_remove ()

```
void        gtk_clist_remove                (GtkCList *clist,
                                             gint row);
```

Warning

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

Removes the specified row from the CList.

clist : The GtkCList to affect.
row : The row to remove.

gtk_clist_set_row_data ()

```
void        gtk_clist_set_row_data          (GtkCList *clist,
                                             gint row,
```

```
                                                gpointer data);
```

Warning

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

Sets data for the specified row. This is the same as calling gtk_clist_set_row_data_full(clist, row, data, NULL).

clist : The GtkCList to affect.
row : The row to affect.
data : The data to set for the row.

---

gtk_clist_set_row_data_full ()

```
void        gtk_clist_set_row_data_full     (GtkCList *clist,
                                             gint row,
                                             gpointer data,
                                             GtkDestroyNotify destroy);
```

Warning

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

Sets the data for specified row, with a callback when the row is destroyed.

clist : The GtkCList to affect.
row : The row to affect.
data : The data to set for the row.
destroy : A GtkDestroyNotify function to be called when the row is destroyed.

---

gtk_clist_get_row_data ()

```
gpointer    gtk_clist_get_row_data          (GtkCList *clist,
                                             gint row);
```

Warning

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

Gets the currently set data for the specified row.

clist : The GtkCList to affect.
row : The row to query.
Returns : The data set for the row.

---

gtk_clist_find_row_from_data ()

```
gint        gtk_clist_find_row_from_data    (GtkCList *clist,
                                             gpointer data);
```

Warning

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

Searches the CList for the row with the specified data.

clist : The GtkCList to search.
data : The data to search for a match.
Returns : The number of the matching row, or -1 if no match could be found.

---

gtk_clist_select_row ()

```
void        gtk_clist_select_row            (GtkCList *clist,
                                             gint row,
                                             gint column);
```

Warning

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

Selects the specified row. Causes the "select-row" signal to be emitted for the specified row and column.

clist : The GtkCList to affect.
row : The row to select.
column : The column to select.

---

gtk_clist_unselect_row ()

```
void        gtk_clist_unselect_row          (GtkCList *clist,
                                             gint row,
                                             gint column);
```

Warning

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

Unselects the specified row. Causes the "unselect-row" signal to be emitted for the specified row and column.

`clist` : The GtkCList to affect.
`row` : The row to select.
`column` : The column to select.

---

gtk_clist_undo_selection ()

```
void        gtk_clist_undo_selection        (GtkCList *clist);
```

Warning

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

Undoes the last selection for an "extended selection mode" CList.

`clist` : The GtkCList to affect.

---

gtk_clist_clear ()

```
void        gtk_clist_clear                 (GtkCList *clist);
```

Warning

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

Removes all the CList's rows.

clist : The GtkCList to affect.

---

gtk_clist_get_selection_info ()

```
gint        gtk_clist_get_selection_info    (GtkCList *clist,
                                             gint x,
                                             gint y,
                                             gint *row,
                                             gint *column);
```

Warning

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

Gets the row and column at the specified pixel position in the CList.

clist : The GtkCList to affect.
x : The horizontal pixel position to check.
y : The vertical pixel position to check..
row : Pointer to a gint to store the row value.
column : Pointer to a gint to store the column value.
Returns : 1 if row/column is returned and in range, 0 otherwise.

---

gtk_clist_select_all ()

```
void        gtk_clist_select_all            (GtkCList *clist);
```

Warning

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

Selects all rows in the CList. This function has no affect for a CList in "single" or "browse" selection mode.

clist : The GtkCList to affect.

---

gtk_clist_unselect_all ()

```
void        gtk_clist_unselect_all          (GtkCList *clist);
```

Warning

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

Unselects all rows in the CList.

clist : The GtkCList to affect.

---

gtk_clist_swap_rows ()

```
void        gtk_clist_swap_rows             (GtkCList *clist,
                                             gint row1,
                                             gint row2);
```

Warning

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

Swaps the two specified rows with each other.

clist : The GtkCList to affect.
row1 : Number of the first row.
row2 : Number of the second row.

---

gtk_clist_set_compare_func ()

```
void        gtk_clist_set_compare_func      (GtkCList *clist,
                                             GtkCListCompareFunc cmp_func);
```

Warning

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

Sets the compare function of the GtkClist to *cmp_func*. If *cmp_func* is NULL, then the default compare function is used. The default compare function sorts ascending or with the type set by gtk_clist_set_sort_type() by the column set by gtk_clist_set_sort_column().

clist : The GtkCList to affect.
cmp_func : The GtkCompareFunction to use.

---

gtk_clist_set_sort_column ()

```
void        gtk_clist_set_sort_column       (GtkCList *clist,
                                             gint column);
```

Warning

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

Sets the sort column of the clist. The sort column is used by the default compare function to determine which column to sort by.

clist : The GtkCList to affect.
column : The column to sort by

---

gtk_clist_set_sort_type ()

```
void        gtk_clist_set_sort_type         (GtkCList *clist,
                                             GtkSortType sort_type);
```

Warning

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

Sets the sort type of the GtkClist. This is either GTK_SORT_ASCENDING for ascening sort or GTK_SORT_DESCENDING for descending sort.

clist : The GtkCList to affect.
sort_type : the GtkSortType to use

---

gtk_clist_sort ()

```
void        gtk_clist_sort                  (GtkCList *clist);
```

Warning

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

Sorts the GtkClist according to the current compare function, which can be set with the `gtk_clist_set_compare_func()` function.

`clist` : The GtkCList to sort.

---

gtk_clist_set_auto_sort ()

```
void        gtk_clist_set_auto_sort         (GtkCList *clist,
                                             gboolean auto_sort);
```

Warning

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

Turns on or off auto sort of the GtkCList. If auto sort is on, then the CList will be resorted when a row is inserted into the CList.

`clist` : The GtkCList to affect.

`auto_sort` : whether auto sort should be on or off

---

gtk_clist_columns_autosize ()

```
gint        gtk_clist_columns_autosize      (GtkCList *clist);
```

Warning

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

Auto-sizes all columns in the CList and returns the total width of the CList.

clist : The GtkCList to affect.

Returns : The total width of the CList.

---

gtk_clist_get_column_title ()

```
gchar*      gtk_clist_get_column_title      (GtkCList *clist,
                                             gint column);
```

Warning

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

Gets the current title of the specified column

clist : The GtkCList to affect.

column : The column to query.

Returns : The title of the column.

---

gtk_clist_get_column_widget ()

```
GtkWidget*  gtk_clist_get_column_widget     (GtkCList *clist,
                                             gint column);
```

Warning

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

Gets the widget in the column header for the specified column.

clist : The GtkCList to affect.

column : The column to query.

Returns : Pointer to a GtkWidget for the column header.

---

gtk_clist_get_hadjustment ()

```
GtkAdjustment* gtk_clist_get_hadjustment    (GtkCList *clist);
```

Warning

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

Gets the GtkAdjustment currently being used for the horizontal aspect.

`clist` : The GtkCList to check.

Returns : A GtkAdjustment object, or NULL if none is currently being used.

---

gtk_clist_get_vadjustment ()

```
GtkAdjustment* gtk_clist_get_vadjustment    (GtkCList *clist);
```

Warning

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

Gets the GtkAdjustment currently being used for the vertical aspect.

`clist` : The GtkCList to check.

Returns : A GtkAdjustment object, or NULL if none is currently being used.

---

gtk_clist_row_move ()

```
void        gtk_clist_row_move              (GtkCList *clist,
                                             gint source_row,
                                             gint dest_row);
```

Warning

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

Allows you to move a row from one position to another in the list.

clist : The GtkCList to affect.

source_row : The original position of the row to move.

dest_row : The position to which the row should be moved.

---

gtk_clist_set_button_actions ()

```
void        gtk_clist_set_button_actions    (GtkCList *clist,
                                             guint button,
                                             guint8 button_actions);
```

Warning

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

Sets the action(s) that the specified mouse button will have on the CList.

clist : The GtkCList to affect.

button : The mouse button to set. The values here, unlike in the rest of GTK+ start from 0. For instance, the right mouse button, which is 3 elsewhere, should be given as 2 here.

button_actions : A logically OR'd value of GtkButtonAction values for the button.

---

gtk_clist_set_hadjustment ()

```
void        gtk_clist_set_hadjustment       (GtkCList *clist,
                                             GtkAdjustment *adjustment);
```

Warning

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

Allows you to set the GtkAdjustment to be used for the horizontal aspect of the GtkCList widget.

clist : The GtkCList to affect.

adjustment : A pointer to a GtkAdjustment widget, or NULL.

---

gtk_clist_set_reorderable ()

```
void        gtk_clist_set_reorderable       (GtkCList *clist,
                                             gboolean reorderable);
```

Warning

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

Sets whether the CList's rows are re-orderable using drag-and-drop.

`clist` : The GtkCList to affect.
`reorderable` : TRUE or FALSE.

---

gtk_clist_set_use_drag_icons ()

```
void        gtk_clist_set_use_drag_icons    (GtkCList *clist,
                                             gboolean use_icons);
```

Warning

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

Determines whether the GtkClist should use icons when doing drag-and-drop operations.

`clist` : The GtkCList to affect.
`use_icons` : TRUE or FALSE.

---

gtk_clist_set_vadjustment ()

```
void        gtk_clist_set_vadjustment       (GtkCList *clist,
                                             GtkAdjustment *adjustment);
```

Warning

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

Allows you to set the GtkAdjustment to be used for the vertical aspect of the GtkCList widget.

clist : The GtkCList to affect.
adjustment : A pointer to a GtkAdjustment widget, or NULL.

Properties

The "n-columns" property

```
"n-columns"           guint                 : Read / Write / Construct Only
```

An integer value for a column.

Default value: 0

The "reorderable" property

```
"reorderable"         gboolean              : Read / Write
```

A boolean value for determining if the user can re-order the CList's columns.

Default value: FALSE

The "row-height" property

```
"row-height"          guint                 : Read / Write
```

An integer value representing the height of a row in pixels.

Default value: 0

The "selection-mode" property

```
  "selection-mode"       GtkSelectionMode      : Read / Write
```

Sets the type of selection mode for the CList.

Default value: GTK_SELECTION_NONE

The "shadow-type" property

```
  "shadow-type"          GtkShadowType         : Read / Write
```

Sets the shadowing for the CList.

Default value: GTK_SHADOW_NONE

The "sort-type" property

```
  "sort-type"            GtkSortType           : Read / Write
```

Default value: GTK_SORT_ASCENDING

The "titles-active" property

```
  "titles-active"        gboolean              : Read / Write
```

A boolean value for setting whether the column titles can be clicked.

Default value: FALSE

The "use-drag-icons" property

```
  "use-drag-icons"       gboolean              : Read / Write
```

A boolean value for setting whether to use icons during drag operations.

Default value: FALSE

Signals

The "abort-column-resize" signal

```
void        user_function                   (GtkCList *clist,
                                             gpointer user_data);
```

This signal is emitted when a column resize is aborted.

clist : the object which received the signal.
user_data : user data set when the signal handler was connected.

The "click-column" signal

```
void        user_function                   (GtkCList *clist,
                                             gint column,
                                             gpointer user_data);
```

This signal is emitted when a column title is clicked.

clist : The object which received the signal.
column : The number of the column.
user_data : user data set when the signal handler was connected.

The "end-selection" signal

```
void        user_function                   (GtkCList *clist,
                                             gpointer user_data);
```

This signal is emitted when a selection ends in a multiple selection CList.

clist : the object which received the signal.

user_data : user data set when the signal handler was connected.

The "extend-selection" signal

```
void        user_function                   (GtkCList *clist,
                                             GtkScrollType scroll_type,
                                             gfloat position,
                                             gboolean auto_start_selection,
                                             gpointer user_data);
```

This signal is emitted when the selection is extended.

clist :	the object which received the signal.
scroll_type :	A GtkScrollType value of any scrolling operation the occured during the selection.
position :	A value between 0.0 and 1.0.
auto_start_selection :	TRUE or FALSE.
user_data :	user data set when the signal handler was connected.

The "resize-column" signal

```
void        user_function                   (GtkCList *clist,
                                             gint column,
                                             gint width,
                                             gpointer user_data);
```

This signal is emitted when a column is resized.

clist :	The object which received the signal.
column :	The number of the column
width :	The new width of the column.
user_data :	user data set when the signal handler was connected.

The "row-move" signal

```
void        user_function                   (GtkCList *clist,
```

```
                                        gint arg1,
                                        gint arg2,
                                        gpointer user_data);
```

This signal is emitted when a row is moved.

clist : The object which received the signal.
arg1 : The source position of the row.
arg2 : The destination position of the row.
user_data : user data set when the signal handler was connected.

The "scroll-horizontal" signal

```
void        user_function                   (GtkCList *clist,
                                             GtkScrollType scroll_type,
                                             gfloat position,
                                             gpointer user_data);
```

This signal is emitted when the CList is scrolled horizontally.

clist : the object which received the signal.
scroll_type : A GtkScrollType value of how the scroll operation occured.
position : a value between 0.0 and 1.0.
user_data : user data set when the signal handler was connected.

The "scroll-vertical" signal

```
void        user_function                   (GtkCList *clist,
                                             GtkScrollType scroll_type,
                                             gfloat position,
                                             gpointer user_data);
```

This signal is emitted when the CList is scrolled vertically.

clist : the object which received the signal.
scroll_type : A GtkScrollType value of how the scroll operation occured.
position : A value between 0.0 and 1.0.

user_data : user data set when the signal handler was connected.

The "select-all" signal

```
void        user_function                   (GtkCList *clist,
                                             gpointer user_data);
```

This signal is emitted when all the rows are selected in a CList.

clist : the object which received the signal.
user_data : user data set when the signal handler was connected.

The "select-row" signal

```
void        user_function                   (GtkCList *clist,
                                             gint row,
                                             gint column,
                                             GdkEventButton *event,
                                             gpointer user_data);
```

This signal is emitted when the user selects a row in the list. It is emitted for every row that is selected in a multi-selection or by calling `gtk_clist_select_all()`.

clist : The object which received the signal.
row : The row selected.
column : The column where the selection occured.
event : A GdkEvent structure for the selection.
user_data : user data set when the signal handler was connected.

The "set-scroll-adjustments" signal

```
void        user_function                   (GtkCList *clist,
                                             GtkAdjustment *arg1,
                                             GtkAdjustment *arg2,
                                             gpointer user_data);
```

clist : the object which received the signal.
arg1 :
arg2 :
user_data : user data set when the signal handler was connected.

The "start-selection" signal

```
void        user_function                   (GtkCList *clist,
                                             gpointer user_data);
```

This signal is emitted when a drag-selection is started in a multiple-selection CList.

clist : the object which received the signal.
user_data : user data set when the signal handler was connected.

The "toggle-add-mode" signal

```
void        user_function                   (GtkCList *clist,
                                             gpointer user_data);
```

This signal is emitted when "add mode" is toggled.

clist : the object which received the signal.
user_data : user data set when the signal handler was connected.

The "toggle-focus-row" signal

```
void        user_function                   (GtkCList *clist,
                                             gpointer user_data);
```

clist : The object which received the signal.
user_data : user data set when the signal handler was connected.

The "undo-selection" signal

```
void        user_function                   (GtkCList *clist,
                                             gpointer user_data);
```

This signal is emitted when an undo selection occurs in the CList, probably via calling `gtk_clist_undo_selection()`.

clist : the object which received the signal.
user_data : user data set when the signal handler was connected.

The "unselect-all" signal

```
void        user_function                   (GtkCList *clist,
                                             gpointer user_data);
```

This signal is emitted when all rows are unselected in a CList.

clist : the object which received the signal.
user_data : user data set when the signal handler was connected.

The "unselect-row" signal

```
void        user_function                   (GtkCList *clist,
                                             gint row,
                                             gint column,
                                             GdkEventButton *event,
                                             gpointer user_data);
```

This signal is emitted when the user unselects a row in the list. It is emitted for every row that is unselected in a multi-selection or by calling `gtk_clist_unselect_all()`. It is also emitted for the previously selected row in a "single" or "browse" mode CList.

clist : The object which received the signal.
row : The selected row
column : The column where the selection occured.
event :
user_data : user data set when the signal handler was connected.

<< Deprecated **GtkCTree >>**

GtkCTree

GtkCTree — A widget displaying a hierarchical tree

Synopsis

```
#include <gtk/gtk.h>


            GtkCTree;
#define     GTK_CTREE_ROW                   (_node_)
#define     GTK_CTREE_NODE                  (_node_)
#define     GTK_CTREE_NODE_NEXT             (_nnode_)
#define     GTK_CTREE_NODE_PREV             (_pnode_)
#define     GTK_CTREE_FUNC                  (_func_)
enum        GtkCTreePos;
enum        GtkCTreeLineStyle;
enum        GtkCTreeExpanderStyle;
enum        GtkCTreeExpansionType;
void        (*GtkCTreeFunc)                 (GtkCTree *ctree,
                                             GtkCTreeNode *node,
                                             gpointer data);
gboolean    (*GtkCTreeGNodeFunc)            (GtkCTree *ctree,
                                             guint depth,
                                             GNode *gnode,
                                             GtkCTreeNode *cnode,
                                             gpointer data);
gboolean    (*GtkCTreeCompareDragFunc)      (GtkCTree *ctree,
                                             GtkCTreeNode *source_node,
                                             GtkCTreeNode *new_parent,
                                             GtkCTreeNode *new_sibling);
            GtkCTreeRow;
            GtkCTreeNode;
#define     GTK_TYPE_CTREE_NODE
GtkWidget*  gtk_ctree_new_with_titles       (gint columns,
                                             gint tree_column,
                                             gchar *titles[]);
GtkWidget*  gtk_ctree_new                   (gint columns,
                                             gint tree_column);
GtkCTreeNode* gtk_ctree_insert_node         (GtkCTree *ctree,
```

```
                                        GtkCTreeNode *parent,
                                        GtkCTreeNode *sibling,
                                        gchar *text[],
                                        guint8 spacing,
                                        GdkPixmap *pixmap_closed,
                                        GdkBitmap *mask_closed,
                                        GdkPixmap *pixmap_opened,
                                        GdkBitmap *mask_opened,
                                        gboolean is_leaf,
                                        gboolean expanded);
void        gtk_ctree_remove_node       (GtkCTree *ctree,
                                        GtkCTreeNode *node);
GtkCTreeNode* gtk_ctree_insert_gnode    (GtkCTree *ctree,
                                        GtkCTreeNode *parent,
                                        GtkCTreeNode *sibling,
                                        GNode *gnode,
                                        GtkCTreeGNodeFunc func,
                                        gpointer data);
GNode*      gtk_ctree_export_to_gnode   (GtkCTree *ctree,
                                        GNode *parent,
                                        GNode *sibling,
                                        GtkCTreeNode *node,
                                        GtkCTreeGNodeFunc func,
                                        gpointer data);
void        gtk_ctree_post_recursive    (GtkCTree *ctree,
                                        GtkCTreeNode *node,
                                        GtkCTreeFunc func,
                                        gpointer data);
void        gtk_ctree_post_recursive_to_depth
                                       (GtkCTree *ctree,
                                        GtkCTreeNode *node,
                                        gint depth,
                                        GtkCTreeFunc func,
                                        gpointer data);
void        gtk_ctree_pre_recursive     (GtkCTree *ctree,
                                        GtkCTreeNode *node,
                                        GtkCTreeFunc func,
                                        gpointer data);
void        gtk_ctree_pre_recursive_to_depth
                                       (GtkCTree *ctree,
                                        GtkCTreeNode *node,
                                        gint depth,
                                        GtkCTreeFunc func,
                                        gpointer data);
gboolean    gtk_ctree_is_viewable       (GtkCTree *ctree,
                                        GtkCTreeNode *node);
```

```
GtkCTreeNode* gtk_ctree_last                (GtkCTree *ctree,
                                             GtkCTreeNode *node);
GtkCTreeNode* gtk_ctree_find_node_ptr       (GtkCTree *ctree,
                                             GtkCTreeRow *ctree_row);
gboolean    gtk_ctree_find                  (GtkCTree *ctree,
                                             GtkCTreeNode *node,
                                             GtkCTreeNode *child);
gboolean    gtk_ctree_is_ancestor           (GtkCTree *ctree,
                                             GtkCTreeNode *node,
                                             GtkCTreeNode *child);
GtkCTreeNode* gtk_ctree_find_by_row_data    (GtkCTree *ctree,
                                             GtkCTreeNode *node,
                                             gpointer data);
GList*      gtk_ctree_find_all_by_row_data  (GtkCTree *ctree,
                                             GtkCTreeNode *node,
                                             gpointer data);
GtkCTreeNode* gtk_ctree_find_by_row_data_custom
                                            (GtkCTree *ctree,
                                             GtkCTreeNode *node,
                                             gpointer data,
                                             GCompareFunc func);
GList*      gtk_ctree_find_all_by_row_data_custom
                                            (GtkCTree *ctree,
                                             GtkCTreeNode *node,
                                             gpointer data,
                                             GCompareFunc func);
gboolean    gtk_ctree_is_hot_spot           (GtkCTree *ctree,
                                             gint x,
                                             gint y);
void        gtk_ctree_move                  (GtkCTree *ctree,
                                             GtkCTreeNode *node,
                                             GtkCTreeNode *new_parent,
                                             GtkCTreeNode *new_sibling);
void        gtk_ctree_expand                (GtkCTree *ctree,
                                             GtkCTreeNode *node);
void        gtk_ctree_expand_recursive      (GtkCTree *ctree,
                                             GtkCTreeNode *node);
void        gtk_ctree_expand_to_depth       (GtkCTree *ctree,
                                             GtkCTreeNode *node,
                                             gint depth);
void        gtk_ctree_collapse              (GtkCTree *ctree,
                                             GtkCTreeNode *node);
void        gtk_ctree_collapse_recursive    (GtkCTree *ctree,
                                             GtkCTreeNode *node);
void        gtk_ctree_collapse_to_depth     (GtkCTree *ctree,
                                             GtkCTreeNode *node,
```

```
                                        gint depth);
void        gtk_ctree_toggle_expansion      (GtkCTree *ctree,
                                             GtkCTreeNode *node);
void        gtk_ctree_toggle_expansion_recursive
                                            (GtkCTree *ctree,
                                             GtkCTreeNode *node);
void        gtk_ctree_select                (GtkCTree *ctree,
                                             GtkCTreeNode *node);
void        gtk_ctree_select_recursive      (GtkCTree *ctree,
                                             GtkCTreeNode *node);
void        gtk_ctree_unselect              (GtkCTree *ctree,
                                             GtkCTreeNode *node);
void        gtk_ctree_unselect_recursive    (GtkCTree *ctree,
                                             GtkCTreeNode *node);
void        gtk_ctree_real_select_recursive (GtkCTree *ctree,
                                             GtkCTreeNode *node,
                                             gint state);
void        gtk_ctree_node_set_text         (GtkCTree *ctree,
                                             GtkCTreeNode *node,
                                             gint column,
                                             const gchar *text);
void        gtk_ctree_node_set_pixmap       (GtkCTree *ctree,
                                             GtkCTreeNode *node,
                                             gint column,
                                             GdkPixmap *pixmap,
                                             GdkBitmap *mask);
void        gtk_ctree_node_set_pixtext      (GtkCTree *ctree,
                                             GtkCTreeNode *node,
                                             gint column,
                                             const gchar *text,
                                             guint8 spacing,
                                             GdkPixmap *pixmap,
                                             GdkBitmap *mask);
void        gtk_ctree_set_node_info         (GtkCTree *ctree,
                                             GtkCTreeNode *node,
                                             const gchar *text,
                                             guint8 spacing,
                                             GdkPixmap *pixmap_closed,
                                             GdkBitmap *mask_closed,
                                             GdkPixmap *pixmap_opened,
                                             GdkBitmap *mask_opened,
                                             gboolean is_leaf,
                                             gboolean expanded);
void        gtk_ctree_node_set_shift        (GtkCTree *ctree,
                                             GtkCTreeNode *node,
                                             gint column,
```

```
                                        gint vertical,
                                        gint horizontal);
void        gtk_ctree_node_set_selectable   (GtkCTree *ctree,
                                        GtkCTreeNode *node,
                                        gboolean selectable);
gboolean    gtk_ctree_node_get_selectable   (GtkCTree *ctree,
                                        GtkCTreeNode *node);
GtkCellType gtk_ctree_node_get_cell_type    (GtkCTree *ctree,
                                        GtkCTreeNode *node,
                                        gint column);
gboolean    gtk_ctree_node_get_text         (GtkCTree *ctree,
                                        GtkCTreeNode *node,
                                        gint column,
                                        gchar **text);
gboolean    gtk_ctree_node_get_pixmap       (GtkCTree *ctree,
                                        GtkCTreeNode *node,
                                        gint column,
                                        GdkPixmap **pixmap,
                                        GdkBitmap **mask);
gboolean    gtk_ctree_node_get_pixtext      (GtkCTree *ctree,
                                        GtkCTreeNode *node,
                                        gint column,
                                        gchar **text,
                                        guint8 *spacing,
                                        GdkPixmap **pixmap,
                                        GdkBitmap **mask);
gboolean    gtk_ctree_get_node_info         (GtkCTree *ctree,
                                        GtkCTreeNode *node,
                                        gchar **text,
                                        guint8 *spacing,
                                        GdkPixmap **pixmap_closed,
                                        GdkBitmap **mask_closed,
                                        GdkPixmap **pixmap_opened,
                                        GdkBitmap **mask_opened,
                                        gboolean *is_leaf,
                                        gboolean *expanded);
void        gtk_ctree_node_set_row_style    (GtkCTree *ctree,
                                        GtkCTreeNode *node,
                                        GtkStyle *style);
GtkStyle*   gtk_ctree_node_get_row_style    (GtkCTree *ctree,
                                        GtkCTreeNode *node);
void        gtk_ctree_node_set_cell_style   (GtkCTree *ctree,
                                        GtkCTreeNode *node,
                                        gint column,
                                        GtkStyle *style);
GtkStyle*   gtk_ctree_node_get_cell_style   (GtkCTree *ctree,
```

```
                                            GtkCTreeNode *node,
                                            gint column);
void        gtk_ctree_node_set_foreground   (GtkCTree *ctree,
                                            GtkCTreeNode *node,
                                            const GdkColor *color);
void        gtk_ctree_node_set_background   (GtkCTree *ctree,
                                            GtkCTreeNode *node,
                                            const GdkColor *color);
void        gtk_ctree_node_set_row_data     (GtkCTree *ctree,
                                            GtkCTreeNode *node,
                                            gpointer data);
void        gtk_ctree_node_set_row_data_full
                                            (GtkCTree *ctree,
                                            GtkCTreeNode *node,
                                            gpointer data,
                                            GtkDestroyNotify destroy);
gpointer    gtk_ctree_node_get_row_data     (GtkCTree *ctree,
                                            GtkCTreeNode *node);
void        gtk_ctree_node_moveto           (GtkCTree *ctree,
                                            GtkCTreeNode *node,
                                            gint column,
                                            gfloat row_align,
                                            gfloat col_align);
GtkVisibility gtk_ctree_node_is_visible     (GtkCTree *ctree,
                                            GtkCTreeNode *node);
void        gtk_ctree_set_indent            (GtkCTree *ctree,
                                            gint indent);
void        gtk_ctree_set_spacing           (GtkCTree *ctree,
                                            gint spacing);
#define     gtk_ctree_set_reorderable       (t,r)
void        gtk_ctree_set_line_style        (GtkCTree *ctree,
                                            GtkCTreeLineStyle line_style);
void        gtk_ctree_set_expander_style    (GtkCTree *ctree,
                                            GtkCTreeExpanderStyle expander_style);
void        gtk_ctree_set_drag_compare_func (GtkCTree *ctree,
                                            GtkCTreeCompareDragFunc cmp_func);
void        gtk_ctree_sort_node             (GtkCTree *ctree,
                                            GtkCTreeNode *node);
void        gtk_ctree_sort_recursive        (GtkCTree *ctree,
                                            GtkCTreeNode *node);
GtkCTreeNode* gtk_ctree_node_nth            (GtkCTree *ctree,
                                            guint row);
void        gtk_ctree_set_show_stub         (GtkCTree *ctree,
                                            gboolean show_stub);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkWidget
               +----GtkContainer
                     +----GtkCList
                           +----GtkCTree
```

Implemented Interfaces

GtkCTree implements AtkImplementorIface.

Properties

```
  "expander-style"       GtkCTreeExpanderStyle  : Read / Write
  "indent"               guint                  : Read / Write
  "line-style"           GtkCTreeLineStyle      : Read / Write
  "n-columns"            guint                  : Read / Write / Construct Only
  "show-stub"            gboolean               : Read / Write
  "spacing"              guint                  : Read / Write
  "tree-column"          guint                  : Read / Write / Construct Only
```

Signal Prototypes

```
"change-focus-row-expansion"
            void        user_function      (GtkCTree *ctree,
                                            GtkCTreeExpansionType expansion,
                                            gpointer user_data);
"tree-collapse"
            void        user_function      (GtkCTree *ctree,
                                            GtkCTreeNode *node,
                                            gpointer user_data);
"tree-expand"
            void        user_function      (GtkCTree *ctree,
                                            GtkCTreeNode *node,
                                            gpointer user_data);
"tree-move" void        user_function      (GtkCTree *ctree,
```

```
                                        GtkCTreeNode *node,
                                        GtkCTreeNode *new_parent,
                                        GtkCTreeNode *new_sibling,
                                        gpointer user_data);
"tree-select-row"
            void        user_function      (GtkCTree *ctree,
                                        GtkCTreeNode *node,
                                        gint column,
                                        gpointer user_data);
"tree-unselect-row"
            void        user_function      (GtkCTree *ctree,
                                        GtkCTreeNode *node,
                                        gint column,
                                        gpointer user_data);
```

Description

The GtkCTree widget is used for showing a hierarchical tree to the user, for example a directory tree.

The tree is internally represented as a set of GtkCTreeNode structures.

The interface has much in common with the GtkCList widget: rows (nodes) can be selected by the user etc.

Positions in the tree are often indicated by two arguments, a parent and a sibling, both GtkCTreeNode pointers. If the parent is `NULL`, the position is at the root of the tree and if the sibling is `NULL`, it will be the last child of parent, otherwise it will be inserted just before the sibling.

GtkCTree has been deprecated since GTK+ 2.0 and should not be used in newly written code. Use GtkTreeView instead.

Details

GtkCTree

```
typedef struct _GtkCTree GtkCTree;
```

Warning

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

The GtkCTree-struct contains the following user-accessible fields. These fields should be considered read-only; to set the values, use the methods below.

gint *tree_indent*;	The number of pixels each successive level of the tree is indented in the display.
gint *tree_spacing*;	The space in pixels between the graphical tree and the text in the node.
gint *tree_column*;	The index of the column for which the tree graphics is drawn.
GtkCTreeLineStyle *line_style*;	The style in which the lines in the tree graphics are drawn.
GtkCTreeExpanderStyle *expander_style*;	The style in which the expander buttons are drawn.
GtkCTreeExpanderStyle *expander_style*;	FIXME.

GTK_CTREE_ROW()

```
#define GTK_CTREE_ROW(_node_) ((GtkCTreeRow *)(((GList *)(_node_))->data))
```

Warning

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

Used to get the GtkCTreeRow structure corresponding to the given GtkCTreeNode.

*_node_* :

GTK_CTREE_NODE()

```
#define GTK_CTREE_NODE(_node_) ((GtkCTreeNode *)((_node_)))
```

Warning

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

*_node_* :

GTK_CTREE_NODE_NEXT()

```
#define GTK_CTREE_NODE_NEXT(_nnode_) ((GtkCTreeNode *)(((GList *)(_nnode_))->next))
```

Warning

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

FIXME

*_nnode_* :

GTK_CTREE_NODE_PREV()

```
#define GTK_CTREE_NODE_PREV(_pnode_) ((GtkCTreeNode *)(((GList *)(_pnode_))->prev))
```

Warning

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

FIXME

*_pnode_* :

GTK_CTREE_FUNC()

```
#define GTK_CTREE_FUNC(_func_) ((GtkCTreeFunc)(_func_))
```

Warning

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

*_func_* :

---

enum GtkCTreePos

```
typedef enum
{
  GTK_CTREE_POS_BEFORE,
  GTK_CTREE_POS_AS_CHILD,
  GTK_CTREE_POS_AFTER
} GtkCTreePos;
```

Warning

Warning

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

A value specifying the position of a new node relative to an old one.

GTK_CTREE_POS_BEFORE	As a sibling, before the specified node.
GTK_CTREE_POS_AS_CHILD	As a child of the specified node.
GTK_CTREE_POS_AFTER	As a sibling, after the specified node.

---

enum GtkCTreeLineStyle

```
typedef enum
{
  GTK_CTREE_LINES_NONE,
  GTK_CTREE_LINES_SOLID,
  GTK_CTREE_LINES_DOTTED,
  GTK_CTREE_LINES_TABBED
} GtkCTreeLineStyle;
```

Warning

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

The appearance of the lines in the tree graphics.

GTK_CTREE_LINES_NONE	No lines.
GTK_CTREE_LINES_SOLID	Solid lines.
GTK_CTREE_LINES_DOTTED	Dotted lines.
GTK_CTREE_LINES_TABBED	FIXME.

---

enum GtkCTreeExpanderStyle

```
typedef enum
{
  GTK_CTREE_EXPANDER_NONE,
  GTK_CTREE_EXPANDER_SQUARE,
  GTK_CTREE_EXPANDER_TRIANGLE,
  GTK_CTREE_EXPANDER_CIRCULAR
} GtkCTreeExpanderStyle;
```

Warning

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

The appearance of the expander buttons, i.e. the small buttons which expand or contract parts of the tree when pressed.

GTK_CTREE_EXPANDER_NONE	No expanders.
GTK_CTREE_EXPANDER_SQUARE	Square expanders.
GTK_CTREE_EXPANDER_TRIANGLE	Triangular expanders.
GTK_CTREE_EXPANDER_CIRCULAR	Round expanders.

---

enum GtkCTreeExpansionType

```
typedef enum
{
  GTK_CTREE_EXPANSION_EXPAND,
  GTK_CTREE_EXPANSION_EXPAND_RECURSIVE,
  GTK_CTREE_EXPANSION_COLLAPSE,
  GTK_CTREE_EXPANSION_COLLAPSE_RECURSIVE,
  GTK_CTREE_EXPANSION_TOGGLE,
  GTK_CTREE_EXPANSION_TOGGLE_RECURSIVE
} GtkCTreeExpansionType;
```

Warning

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

How to expand or collapse a part of a tree.

GTK_CTREE_EXPANSION_EXPAND	Expand this node.
GTK_CTREE_EXPANSION_EXPAND_RECURSIVE	Expand this node and everything below it in the hierarchy.
GTK_CTREE_EXPANSION_COLLAPSE	Collapse this node.
GTK_CTREE_EXPANSION_COLLAPSE_RECURSIVE	Collapse this node and everything below it in the hierarchy.
GTK_CTREE_EXPANSION_TOGGLE	Toggle this node (i.e. expand if collapsed and vice versa).
GTK_CTREE_EXPANSION_TOGGLE_RECURSIVE	Toggle this node and everything below it in the hierarchy.

GtkCTreeFunc ()

```
void        (*GtkCTreeFunc)                 (GtkCTree *ctree,
                                             GtkCTreeNode *node,
                                             gpointer data);
```

Warning

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

A generic callback type to do something with a particular node.

ctree : The GtkCTree object.
node : The GtkCTreeNode in the tree.
data : The user data associated with the node.

GtkCTreeGNodeFunc ()

```
gboolean    (*GtkCTreeGNodeFunc)            (GtkCTree *ctree,
                                             guint depth,
                                             GNode *gnode,
                                             GtkCTreeNode *cnode,
                                             gpointer data);
```

Warning

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

FIXME

ctree :
depth :
gnode :
cnode :
data :
Returns :

GtkCTreeCompareDragFunc ()

```
gboolean    (*GtkCTreeCompareDragFunc)      (GtkCTree *ctree,
                                             GtkCTreeNode *source_node,
                                             GtkCTreeNode *new_parent,
                                             GtkCTreeNode *new_sibling);
```

Warning

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

FIXME

ctree :
source_node :
new_parent :
new_sibling :
Returns :

GtkCTreeRow

```
typedef struct {
  GtkCListRow row;

  GtkCTreeNode *parent;
  GtkCTreeNode *sibling;
  GtkCTreeNode *children;

  GdkPixmap *pixmap_closed;
  GdkBitmap *mask_closed;
  GdkPixmap *pixmap_opened;
  GdkBitmap *mask_opened;

  guint16 level;

  guint is_leaf  : 1;
  guint expanded : 1;
} GtkCTreeRow;
```

Warning

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

A structure representing a single row in the tree graph. The values inside the structure should be considered read-only. This structure is derived from the GtkCListRow structure.

GtkCTreeNode *`parent`;	The parent node of the node corresponding to this row.
GtkCTreeNode *`sibling`;	The next sibling node of the node corresponding to this row.
GtkCTreeNode *`children`;	The first child node corresponding to this row; to access the other children, just use the siblings of that node.
GdkPixmap *`pixmap_closed`;	The pixmap to be shown when the node is collapsed.

GdkBitmap *mask_closed;	The mask for the above pixmap.
GdkPixmap *pixmap_opened;	The pixmap to be shown when the node is expanded.
GdkBitmap *mask_opened;	The mask for the above pixmap.
guint16 level;	The level of this node in the tree.
guint is_leaf : 1;	Whether this row is a leaf.
guint expanded : 1;	Whether the children of this row are visible.

GtkCTreeNode

```
typedef struct {
  GList list;
} GtkCTreeNode;
```

Warning

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

This structure is opaque - you should use the macros GTK_CTREE_ROW, GTK_CTREE_NODE_NEXT etc. as well as the functions below to access it.

GTK_TYPE_CTREE_NODE

```
#define GTK_TYPE_CTREE_NODE (gtk_ctree_node_get_type ())
```

Warning

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

---

gtk_ctree_new_with_titles ()

```
GtkWidget*  gtk_ctree_new_with_titles       (gint columns,
                                             gint tree_column,
                                             gchar *titles[]);
```

Warning

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

Create a new GtkCTree widget with the given titles for the columns.

columns :	Number of columns.
tree_column :	Which column has the tree graphic; 0 = leftmost.
titles :	The titles for the columns.
Returns :	The GtkCTree widget.

---

gtk_ctree_new ()

```
GtkWidget*  gtk_ctree_new                   (gint columns,
                                             gint tree_column);
```

Warning

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

Create a new GtkCTree widget.

columns :	Number of columns.
tree_column :	Which columns has the tree graphic.
Returns :	The new GtkCTree widget.

---

gtk_ctree_insert_node ()

```
GtkCTreeNode* gtk_ctree_insert_node         (GtkCTree *ctree,
                                             GtkCTreeNode *parent,
                                             GtkCTreeNode *sibling,
                                             gchar *text[],
                                             guint8 spacing,
                                             GdkPixmap *pixmap_closed,
                                             GdkBitmap *mask_closed,
                                             GdkPixmap *pixmap_opened,
                                             GdkBitmap *mask_opened,
                                             gboolean is_leaf,
                                             gboolean expanded);
```

Warning

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

Insert a new node to the tree. The position is specified through the parent-sibling notation, as explained in the introduction above.

ctree :	The GtkCTree widget.
parent :	The parent node to be.
sibling :	The sibling node to be.
text :	The texts to be shown in each column.
spacing :	The extra space between the pixmap and the text.
pixmap_closed :	The pixmap to be used when the node is collapsed. Can be `NULL`.
mask_closed :	The mask for the above pixmap. Can be `NULL`.
pixmap_opened :	The pixmap to be used when the children are visible. Can be `NULL`.
mask_opened :	The mask for the above pixmap. Can be `NULL`.
is_leaf :	Whether this node is going to be a leaf.
expanded :	Whether this node should start out expanded or not.
Returns :	

---

gtk_ctree_remove_node ()

```
void        gtk_ctree_remove_node           (GtkCTree *ctree,
                                             GtkCTreeNode *node);
```

Warning

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

Remove the node and all nodes underneath it from the tree.

ctree :	The widget.
node :	The node to be removed.

---

gtk_ctree_insert_gnode ()

```
GtkCTreeNode* gtk_ctree_insert_gnode        (GtkCTree *ctree,
                                             GtkCTreeNode *parent,
                                             GtkCTreeNode *sibling,
                                             GNode *gnode,
                                             GtkCTreeGNodeFunc func,
```

```
                                                gpointer data);
```

Warning

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

FIXME

ctree :
parent :
sibling :
gnode :
func :
data :
Returns :

---

gtk_ctree_export_to_gnode ()

```
GNode*      gtk_ctree_export_to_gnode       (GtkCTree *ctree,
                                             GNode *parent,
                                             GNode *sibling,
                                             GtkCTreeNode *node,
                                             GtkCTreeGNodeFunc func,
                                             gpointer data);
```

Warning

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

FIXME

ctree :
parent :
sibling :
node :
func :
data :
Returns :

---

gtk_ctree_post_recursive ()

```
void        gtk_ctree_post_recursive        (GtkCTree *ctree,
                                             GtkCTreeNode *node,
                                             GtkCTreeFunc func,
                                             gpointer data);
```

Warning

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

Recursively apply a function to all nodes of the tree at or below a certain node. The function is called for each node after it has been called for that node's children.

ctree :

node : The node where to start. `NULL` means to start at the root.

func : The function to apply to each node.

data : A closure argument given to each invocation of the function.

---

gtk_ctree_post_recursive_to_depth ()

```
void        gtk_ctree_post_recursive_to_depth
                                            (GtkCTree *ctree,
                                             GtkCTreeNode *node,
                                             gint depth,
                                             GtkCTreeFunc func,
                                             gpointer data);
```

Warning

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

Recursively apply a function to nodes up to a certain depth. The function is called for each node after it has been called for that node's children.

ctree :

node : The node where to start.

depth : The maximum absolute depth for applying the function. If depth is negative, this function just calls gtk_ctree_post_recursive.

func : The function to apply to each node.

data : A closure argument given to each invocation of the function.

---

gtk_ctree_pre_recursive ()

```
void        gtk_ctree_pre_recursive         (GtkCTree *ctree,
                                             GtkCTreeNode *node,
                                             GtkCTreeFunc func,
                                             gpointer data);
```

Warning

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

Recursively apply a function to all nodes of the tree at or below a certain node. The function is called for each node after it has been called for its parent.

ctree :
node : The node where to start. `NULL` means to start at the root.
func : The function to apply to each node.
data : A closure argument given to each invocation of the function.

---

gtk_ctree_pre_recursive_to_depth ()

```
void        gtk_ctree_pre_recursive_to_depth
                                            (GtkCTree *ctree,
                                             GtkCTreeNode *node,
                                             gint depth,
                                             GtkCTreeFunc func,
                                             gpointer data);
```

Warning

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

Recursively apply a function to nodes up to a certain depth. The function is called for each node after it has been called for that node's children.

ctree :
node : The node where to start.
depth : The maximum absolute depth for applying the function. If depth is negative, this function just calls gtk_ctree_post_recursive.
func : The function to apply to each node.
data : A closure argument given to each invocation of the function.

---

gtk_ctree_is_viewable ()

```
gboolean    gtk_ctree_is_viewable           (GtkCTree *ctree,
                                             GtkCTreeNode *node);
```

Warning

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

This function checks whether the given node is viewable i.e. so that all of its parent nodes are expanded. This is different from being actually visible: the node can be viewable but outside the scrolling area of the window.

ctree :
node :
Returns : Whether the node is viewable.

---

gtk_ctree_last ()

```
GtkCTreeNode* gtk_ctree_last                (GtkCTree *ctree,
                                             GtkCTreeNode *node);
```

Warning

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

Returns the last child of the last child of the last child... of the given node.

ctree :
node :
Returns :

---

gtk_ctree_find_node_ptr ()

```
GtkCTreeNode* gtk_ctree_find_node_ptr       (GtkCTree *ctree,
                                             GtkCTreeRow *ctree_row);
```

Warning

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

Finds the node pointer given a GtkCTreeRow structure.

ctree :
ctree_row :
Returns : The node pointer.

---

gtk_ctree_find ()

```
gboolean    gtk_ctree_find                  (GtkCTree *ctree,
                                             GtkCTreeNode *node,
                                             GtkCTreeNode *child);
```

Warning

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

ctree :
node : The node to start searching from. May be `NULL`.
child :
Returns : True if *child* is on some level a child (grandchild...) of the *node*.

---

gtk_ctree_is_ancestor ()

```
gboolean    gtk_ctree_is_ancestor           (GtkCTree *ctree,
                                             GtkCTreeNode *node,
                                             GtkCTreeNode *child);
```

Warning

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

ctree :
node :
child :
Returns : True is *node* is an ancestor of *child*.

---

gtk_ctree_find_by_row_data ()

```
GtkCTreeNode* gtk_ctree_find_by_row_data    (GtkCTree *ctree,
                                             GtkCTreeNode *node,
                                             gpointer data);
```

Warning

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

Finds a node in the tree under *node* that has the given user data pointer.

ctree :
node :
data :
Returns : The node, or `NULL` if not found.

---

gtk_ctree_find_all_by_row_data ()

```
GList*      gtk_ctree_find_all_by_row_data  (GtkCTree *ctree,
                                             GtkCTreeNode *node,
                                             gpointer data);
```

Warning

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

Finds all nodes in the tree under *node* that have the given user data pointer.

ctree :
node :
data :
Returns : A list of nodes that have the given data pointer.

---

gtk_ctree_find_by_row_data_custom ()

```
GtkCTreeNode* gtk_ctree_find_by_row_data_custom
```

```
                                                (GtkCTree *ctree,
                                                 GtkCTreeNode *node,
                                                 gpointer data,
                                                 GCompareFunc func);
```

Warning

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

Find the first node under *node* whose row data pointer fulfills a custom criterion.

ctree :
node : The node where to start searching.
data : User data for the criterion function.
func : The criterion function.
Returns : The first node found.

---

gtk_ctree_find_all_by_row_data_custom ()

```
GList*      gtk_ctree_find_all_by_row_data_custom
                                                (GtkCTree *ctree,
                                                 GtkCTreeNode *node,
                                                 gpointer data,
                                                 GCompareFunc func);
```

Warning

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

Find all nodes under *node* whose row data pointer fulfills a custom criterion.

ctree :
node : The node where to start searching.
data : User data for the criterion function.
func : The criterion function.
Returns : A list of all nodes found.

---

gtk_ctree_is_hot_spot ()

```
gboolean    gtk_ctree_is_hot_spot           (GtkCTree *ctree,
                                             gint x,
                                             gint y);
```

Warning

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

ctree :
x :
y :
Returns : True if the given coordinates lie on an expander button.

---

gtk_ctree_move ()

```
void        gtk_ctree_move                  (GtkCTree *ctree,
                                             GtkCTreeNode *node,
                                             GtkCTreeNode *new_parent,
                                             GtkCTreeNode *new_sibling);
```

Warning

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

Move a node in the tree to another location.

ctree :
node : The node to be moved.
new_parent : The new parent-to-be of the node.
new_sibling : The new sibling-to-be of the node.

---

gtk_ctree_expand ()

```
void        gtk_ctree_expand                (GtkCTree *ctree,
                                             GtkCTreeNode *node);
```

Warning

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

Expand one node.

ctree :
node :

---

gtk_ctree_expand_recursive ()

```
void        gtk_ctree_expand_recursive      (GtkCTree *ctree,
                                             GtkCTreeNode *node);
```

Warning

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

Expand one node and all nodes underneath.

ctree :
node :

---

gtk_ctree_expand_to_depth ()

```
void        gtk_ctree_expand_to_depth       (GtkCTree *ctree,
                                             GtkCTreeNode *node,
                                             gint depth);
```

Warning

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

Expand a node and its children up to the depth given.

ctree :
node :
depth : The (absolute) depth up to which to expand nodes.

---

gtk_ctree_collapse ()

```
void        gtk_ctree_collapse              (GtkCTree *ctree,
                                             GtkCTreeNode *node);
```

Warning

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

Collapse one node.

ctree :
node :

---

gtk_ctree_collapse_recursive ()

```
void        gtk_ctree_collapse_recursive    (GtkCTree *ctree,
                                             GtkCTreeNode *node);
```

Warning

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

Collapse one node and all its subnodes.

ctree :
node :

---

gtk_ctree_collapse_to_depth ()

```
void        gtk_ctree_collapse_to_depth     (GtkCTree *ctree,
                                             GtkCTreeNode *node,
                                             gint depth);
```

Warning

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

Collapse a node and its children up to the depth given.

ctree :

node :
depth : The (absolute) depth up to which to collapse nodes.

---

gtk_ctree_toggle_expansion ()

```
void        gtk_ctree_toggle_expansion      (GtkCTree *ctree,
                                             GtkCTreeNode *node);
```

Warning

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

Toggle a node, i.e. if it is collapsed, expand it and vice versa.

ctree :
node :

---

gtk_ctree_toggle_expansion_recursive ()

```
void        gtk_ctree_toggle_expansion_recursive
                                            (GtkCTree *ctree,
                                             GtkCTreeNode *node);
```

Warning

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

Toggle the expansion of a node and all its children.

ctree :
node :

---

gtk_ctree_select ()

```
void        gtk_ctree_select                (GtkCTree *ctree,
                                             GtkCTreeNode *node);
```

Warning

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

Cause the given node to be selected and emit the appropriate signal.

ctree :
node :

---

gtk_ctree_select_recursive ()

```
void        gtk_ctree_select_recursive      (GtkCTree *ctree,
                                             GtkCTreeNode *node);
```

Warning

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

Cause the given node and its subnodes to be selected and emit the appropriate signal(s).

ctree :
node :

---

gtk_ctree_unselect ()

```
void        gtk_ctree_unselect              (GtkCTree *ctree,
                                             GtkCTreeNode *node);
```

Warning

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

Unselect the given node and emit the appropriate signal.

ctree :
node :

---

gtk_ctree_unselect_recursive ()

```
void        gtk_ctree_unselect_recursive    (GtkCTree *ctree,
                                             GtkCTreeNode *node);
```

Warning

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

Unselect the given node and its subnodes and emit the appropriate signal(s).

`ctree` :
`node` :

---

gtk_ctree_real_select_recursive ()

```
void        gtk_ctree_real_select_recursive (GtkCTree *ctree,
                                             GtkCTreeNode *node,
                                             gint state);
```

Warning

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

The function that implements both gtk_ctree_select_recursive and gtk_ctree_unselect_recursive.

`ctree` :
`node` :
`state` : True for selecting, false for unselecting.

---

gtk_ctree_node_set_text ()

```
void        gtk_ctree_node_set_text         (GtkCTree *ctree,
                                             GtkCTreeNode *node,
                                             gint column,
                                             const gchar *text);
```

Warning

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

Set the text in a node.

ctree :
node :
column : The column whose text to change.
text : The new text.

---

gtk_ctree_node_set_pixmap ()

```
void        gtk_ctree_node_set_pixmap       (GtkCTree *ctree,
                                             GtkCTreeNode *node,
                                             gint column,
                                             GdkPixmap *pixmap,
                                             GdkBitmap *mask);
```

Warning

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

FIXME

ctree :
node :
column :
pixmap :
mask :

---

gtk_ctree_node_set_pixtext ()

```
void        gtk_ctree_node_set_pixtext      (GtkCTree *ctree,
                                             GtkCTreeNode *node,
                                             gint column,
                                             const gchar *text,
                                             guint8 spacing,
                                             GdkPixmap *pixmap,
                                             GdkBitmap *mask);
```

Warning

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

FIXME

ctree:
node:
column:
text:
spacing:
pixmap:
mask:

---

gtk_ctree_set_node_info ()

```
void        gtk_ctree_set_node_info         (GtkCTree *ctree,
                                             GtkCTreeNode *node,
                                             const gchar *text,
                                             guint8 spacing,
                                             GdkPixmap *pixmap_closed,
                                             GdkBitmap *mask_closed,
                                             GdkPixmap *pixmap_opened,
                                             GdkBitmap *mask_opened,
                                             gboolean is_leaf,
                                             gboolean expanded);
```

Warning

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

Change the information. Most parameters correspond to the parameters of gtk_ctree_insert_node.

ctree:
node:
text: The text to be in the tree column.
spacing:
pixmap_closed:
mask_closed:
pixmap_opened:
mask_opened:
is_leaf:
expanded:

---

gtk_ctree_node_set_shift ()

```
void        gtk_ctree_node_set_shift        (GtkCTree *ctree,
                                             GtkCTreeNode *node,
                                             gint column,
                                             gint vertical,
                                             gint horizontal);
```

Warning

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

Shift the given cell the given amounts in pixels.

ctree :
node :
column :
vertical :
horizontal :

---

gtk_ctree_node_set_selectable ()

```
void        gtk_ctree_node_set_selectable   (GtkCTree *ctree,
                                             GtkCTreeNode *node,
                                             gboolean selectable);
```

Warning

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

ctree :
node :
selectable : Whether this node can be selected by the user.

---

gtk_ctree_node_get_selectable ()

```
gboolean    gtk_ctree_node_get_selectable   (GtkCTree *ctree,
```

```
                                                GtkCTreeNode *node);
```

Warning

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

ctree :
node :
Returns : Whether this node can be selected by the user.

---

gtk_ctree_node_get_cell_type ()

```
GtkCellType gtk_ctree_node_get_cell_type    (GtkCTree *ctree,
                                             GtkCTreeNode *node,
                                             gint column);
```

Warning

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

ctree :
node :
column :
Returns : The type of the given cell.

---

gtk_ctree_node_get_text ()

```
gboolean    gtk_ctree_node_get_text         (GtkCTree *ctree,
                                             GtkCTreeNode *node,
                                             gint column,
                                             gchar **text);
```

Warning

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

ctree :
node :

column :

text : If nonnull, the pointer to the text string is assigned to **text*.

Returns : True if the given cell has text in it.

---

gtk_ctree_node_get_pixmap ()

```
gboolean    gtk_ctree_node_get_pixmap       (GtkCTree *ctree,
                                             GtkCTreeNode *node,
                                             gint column,
                                             GdkPixmap **pixmap,
                                             GdkBitmap **mask);
```

Warning

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

ctree :

node :

column :

pixmap : If nonnull, the pointer to the pixmap is returned through this.

mask : If nonnull, the pointer to the mask is returned through this.

Returns : True if the given cell contains a pixmap.

---

gtk_ctree_node_get_pixtext ()

```
gboolean    gtk_ctree_node_get_pixtext      (GtkCTree *ctree,
                                             GtkCTreeNode *node,
                                             gint column,
                                             gchar **text,
                                             guint8 *spacing,
                                             GdkPixmap **pixmap,
                                             GdkBitmap **mask);
```

Warning

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

Get the parameters of a cell containing both a pixmap and text.

ctree :
node :
column :
text :
spacing :
pixmap :
mask :
Returns :

---

gtk_ctree_get_node_info ()

```
gboolean    gtk_ctree_get_node_info         (GtkCTree *ctree,
                                             GtkCTreeNode *node,
                                             gchar **text,
                                             guint8 *spacing,
                                             GdkPixmap **pixmap_closed,
                                             GdkBitmap **mask_closed,
                                             GdkPixmap **pixmap_opened,
                                             GdkBitmap **mask_opened,
                                             gboolean *is_leaf,
                                             gboolean *expanded);
```

Warning

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

Get information corresponding to a node. Any of the return parameters can be null.

ctree :
node :
text :
spacing :
pixmap_closed :
mask_closed :
pixmap_opened :
mask_opened :
is_leaf :
expanded :
Returns :

---

gtk_ctree_node_set_row_style ()

```
void        gtk_ctree_node_set_row_style    (GtkCTree *ctree,
                                             GtkCTreeNode *node,
                                             GtkStyle *style);
```

Warning

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

Set the style of a row.

ctree :
node :
style :

---

gtk_ctree_node_get_row_style ()

```
GtkStyle*   gtk_ctree_node_get_row_style    (GtkCTree *ctree,
                                             GtkCTreeNode *node);
```

Warning

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

Get the style of a row.

ctree :
node :
Returns :

---

gtk_ctree_node_set_cell_style ()

```
void        gtk_ctree_node_set_cell_style   (GtkCTree *ctree,
                                             GtkCTreeNode *node,
                                             gint column,
                                             GtkStyle *style);
```

Warning

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

Set the style of an individual cell.

ctree :
node :
column :
style :

---

gtk_ctree_node_get_cell_style ()

```
GtkStyle*   gtk_ctree_node_get_cell_style   (GtkCTree *ctree,
                                             GtkCTreeNode *node,
                                             gint column);
```

Warning

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

Get the style of an individual cell.

ctree :
node :
column :
Returns :

---

gtk_ctree_node_set_foreground ()

```
void        gtk_ctree_node_set_foreground   (GtkCTree *ctree,
                                             GtkCTreeNode *node,
                                             const GdkColor *color);
```

Warning

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

ctree :

node :
color :

---

gtk_ctree_node_set_background ()

```
void        gtk_ctree_node_set_background   (GtkCTree *ctree,
                                             GtkCTreeNode *node,
                                             const GdkColor *color);
```

Warning

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

ctree :
node :
color :

---

gtk_ctree_node_set_row_data ()

```
void        gtk_ctree_node_set_row_data     (GtkCTree *ctree,
                                             GtkCTreeNode *node,
                                             gpointer data);
```

Warning

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

Set the custom data associated with a node.

ctree :
node :
data :

---

gtk_ctree_node_set_row_data_full ()

```
void        gtk_ctree_node_set_row_data_full
                                            (GtkCTree *ctree,
                                             GtkCTreeNode *node,
```

```
                                                gpointer data,
                                                GtkDestroyNotify destroy);
```

Warning

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

This is the full interface to setting row data, so that a destructor can be given for the data.

ctree :
node :
data :
destroy : The routine to be called when *data* is no longer needed.

---

gtk_ctree_node_get_row_data ()

```
gpointer    gtk_ctree_node_get_row_data     (GtkCTree *ctree,
                                             GtkCTreeNode *node);
```

Warning

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

ctree :
node :
Returns :

---

gtk_ctree_node_moveto ()

```
void        gtk_ctree_node_moveto           (GtkCTree *ctree,
                                             GtkCTreeNode *node,
                                             gint column,
                                             gfloat row_align,
                                             gfloat col_align);
```

Warning

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

This function makes the given column of the given node visible by scrolling.

ctree :
node : The node to be made visible.
column : The column to be made visible.
row_align : Where in the window the row should appear.
col_align : Where in the window the column should appear.

---

gtk_ctree_node_is_visible ()

```
GtkVisibility gtk_ctree_node_is_visible     (GtkCTree *ctree,
                                             GtkCTreeNode *node);
```

Warning

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

ctree :
node :
Returns : True if the node is currently inside the bounds of the window. Note that this function can return true even if the node is not viewable, if the node's ancestor is visible.

---

gtk_ctree_set_indent ()

```
void        gtk_ctree_set_indent            (GtkCTree *ctree,
                                             gint indent);
```

Warning

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

ctree :
indent : The number of pixels to shift the levels of the tree.

---

gtk_ctree_set_spacing ()

```
void        gtk_ctree_set_spacing           (GtkCTree *ctree,
```

```
                                                gint spacing);
```

Warning

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

The spacing between the tree graphic and the actual node content.

ctree:
spacing:

---

gtk_ctree_set_reorderable()

```
#define gtk_ctree_set_reorderable(t,r)
gtk_clist_set_reorderable((GtkCList*) (t),(r))
```

Warning

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

t:
r:

---

gtk_ctree_set_line_style ()

```
void        gtk_ctree_set_line_style        (GtkCTree *ctree,
                                             GtkCTreeLineStyle line_style);
```

Warning

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

ctree:
line_style:

---

gtk_ctree_set_expander_style ()

```
```

```
void        gtk_ctree_set_expander_style    (GtkCTree *ctree,
                                             GtkCTreeExpanderStyle expander_style);
```

Warning

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

ctree:
expander_style:

---

gtk_ctree_set_drag_compare_func ()

```
void        gtk_ctree_set_drag_compare_func (GtkCTree *ctree,
                                             GtkCTreeCompareDragFunc cmp_func);
```

Warning

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

FIXME

ctree:
cmp_func:

---

gtk_ctree_sort_node ()

```
void        gtk_ctree_sort_node             (GtkCTree *ctree,
                                             GtkCTreeNode *node);
```

Warning

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

Sort the children of a node. See GtkCList for how to set the sorting criteria etc.

ctree:
node:

---

gtk_ctree_sort_recursive ()

```
void        gtk_ctree_sort_recursive        (GtkCTree *ctree,
                                             GtkCTreeNode *node);
```

Warning

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

Sort the descendants of a node. See GtkCList for how to set the sorting criteria etc.

ctree :
node :

---

gtk_ctree_node_nth ()

```
GtkCTreeNode* gtk_ctree_node_nth            (GtkCTree *ctree,
                                             guint row);
```

Warning

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

ctree :
row :
Returns : The node corresponding to the *row* th row.

---

gtk_ctree_set_show_stub ()

```
void        gtk_ctree_set_show_stub         (GtkCTree *ctree,
                                             gboolean show_stub);
```

Warning

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

ctree :
show_stub :

Properties

The "expander-style" property

```
"expander-style"       GtkCTreeExpanderStyle  : Read / Write
```

The style of the expander buttons.

Default value: GTK_CTREE_EXPANDER_NONE

The "indent" property

```
"indent"               guint                  : Read / Write
```

The number of pixels to indent the tree levels.

Default value: 0

The "line-style" property

```
"line-style"           GtkCTreeLineStyle      : Read / Write
```

The style of the lines in the tree graphic.

Default value: GTK_CTREE_LINES_NONE

The "n-columns" property

```
"n-columns"            guint                  : Read / Write / Construct Only
```

The number of columns in the tree.

Default value: 0

The "show-stub" property

```
  "show-stub"           gboolean              : Read / Write
```

Default value: FALSE

The "spacing" property

```
  "spacing"             guint                 : Read / Write
```

The number of pixels between the tree and the columns.

Default value: 0

The "tree-column" property

```
  "tree-column"         guint                 : Read / Write / Construct Only
```

The column in which the actual tree graphic appears.

Default value: 0

Signals

The "change-focus-row-expansion" signal

```
void        user_function                   (GtkCTree *ctree,
                                             GtkCTreeExpansionType expansion,
                                             gpointer user_data);
```

The row which has the focus is either collapsed or expanded or toggled.

ctree : the object which received the signal.
expansion : What is being done.
user_data : user data set when the signal handler was connected.

The "tree-collapse" signal

```
void        user_function                   (GtkCTree *ctree,
                                             GtkCTreeNode *node,
                                             gpointer user_data);
```

Emitted when a node is collapsed.

ctree : the object which received the signal.
node :
user_data : user data set when the signal handler was connected.

The "tree-expand" signal

```
void        user_function                   (GtkCTree *ctree,
                                             GtkCTreeNode *node,
                                             gpointer user_data);
```

Emitted when a node is expanded.

ctree : the object which received the signal.
node :
user_data : user data set when the signal handler was connected.

The "tree-move" signal

```
void        user_function                   (GtkCTree *ctree,
                                             GtkCTreeNode *node,
                                             GtkCTreeNode *new_parent,
                                             GtkCTreeNode *new_sibling,
                                             gpointer user_data);
```

Emitted when a node is moved.

ctree : the object which received the signal.
node : The node that is moved.

new_parent : The new parent of the node.
new_sibling : The new sibling of the node.
user_data : user data set when the signal handler was connected.

The "tree-select-row" signal

```
void        user_function                   (GtkCTree *ctree,
                                             GtkCTreeNode *node,
                                             gint column,
                                             gpointer user_data);
```

Emitted when a row is selected.

ctree : the object which received the signal.
node : The node corresponding to the selected row.
column : The column which was selected.
user_data : user data set when the signal handler was connected.

The "tree-unselect-row" signal

```
void        user_function                   (GtkCTree *ctree,
                                             GtkCTreeNode *node,
                                             gint column,
                                             gpointer user_data);
```

Emitted when a node is unselected.

ctree : the object which received the signal.
node : The node corresponding to the selected row.
column :
user_data : user data set when the signal handler was connected.

<< GtkCList **GtkCombo >>**

GtkCombo

GtkCombo — A text entry field with a dropdown list

Synopsis

```
#include <gtk/gtk.h>


            GtkCombo;
GtkWidget*  gtk_combo_new                   (void);
void        gtk_combo_set_popdown_strings   (GtkCombo *combo,
                                             GList *strings);
void        gtk_combo_set_value_in_list     (GtkCombo *combo,
                                             gboolean val,
                                             gboolean ok_if_empty);
void        gtk_combo_set_use_arrows        (GtkCombo *combo,
                                             gboolean val);
void        gtk_combo_set_use_arrows_always (GtkCombo *combo,
                                             gboolean val);
void        gtk_combo_set_case_sensitive    (GtkCombo *combo,
                                             gboolean val);
void        gtk_combo_set_item_string       (GtkCombo *combo,
                                             GtkItem *item,
                                             const gchar *item_value);
void        gtk_combo_disable_activate      (GtkCombo *combo);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkWidget
               +----GtkContainer
```

```
                    +----GtkBox
                          +----GtkHBox
                                +----GtkCombo
```

Implemented Interfaces

GtkCombo implements AtkImplementorIface.

Properties

```
"allow-empty"          gboolean              : Read / Write
"case-sensitive"       gboolean              : Read / Write
"enable-arrow-keys"    gboolean              : Read / Write
"enable-arrows-always" gboolean              : Read / Write
"value-in-list"        gboolean              : Read / Write
```

Description

The GtkCombo widget consists of a single-line text entry field and a drop-down list. The drop-down list is displayed when the user clicks on a small arrow button to the right of the entry field.

The drop-down list is a GtkList widget and can be accessed using the *list* member of the GtkCombo-struct. List elements can contain arbitrary widgets, but if an element is not a plain label, then you must use the `gtk_list_set_item_string()` function. This sets the string which will be placed in the text entry field when the item is selected.

By default, the user can step through the items in the list using the arrow (cursor) keys, though this behaviour can be turned off with `gtk_combo_set_use_arrows()`.

As of GTK+ 2.4, GtkCombo has been deprecated in favor of GtkComboBox.

Example 1. Creating a GtkCombo widget with simple text items.

```
GtkWidget *combo;
GList *items = NULL;

items = g_list_append (items, "First Item");
items = g_list_append (items, "Second Item");
items = g_list_append (items, "Third Item");
items = g_list_append (items, "Fourth Item");
```

```
  items = g_list_append (items, "Fifth Item");

  combo = gtk_combo_new ();
  gtk_combo_set_popdown_strings (GTK_COMBO (combo), items);
```

Example 2. Creating a GtkCombo widget with a complex item.

```
  GtkWidget *combo, *item, *hbox, *arrow, *label;

  combo = gtk_combo_new ();

  item = gtk_list_item_new ();
  gtk_widget_show (item);

  /* You can put almost anything into the GtkListItem widget. Here we will use
     a horizontal box with an arrow and a label in it. */
  hbox = gtk_hbox_new (FALSE, 3);
  gtk_container_add (GTK_CONTAINER (item), hbox);
  gtk_widget_show (hbox);

  arrow = gtk_arrow_new (GTK_ARROW_RIGHT, GTK_SHADOW_OUT);
  gtk_widget_show (arrow);
  gtk_box_pack_start (GTK_BOX (hbox), arrow, FALSE, FALSE, 0);

  label = gtk_label_new ("First Item");
  gtk_widget_show (label);
  gtk_box_pack_start (GTK_BOX (hbox), label, FALSE, FALSE, 0);

  /* You must set the string to display in the entry field when the item is
     selected. */
  gtk_combo_set_item_string (GTK_COMBO (combo), GTK_ITEM (item), "1st Item");

  /* Now we simply add the item to the combo's list. */
  gtk_container_add (GTK_CONTAINER (GTK_COMBO (combo)->list), item);
```

Details

GtkCombo

```
typedef struct {
        GtkWidget *entry;

        GtkWidget *list;
} GtkCombo;
```

Warning

`GtkCombo` is deprecated and should not be used in newly-written code. Use GtkComboBox instead.

The GtkFixedChild-struct struct contains the following fields. (These fields should be considered read-only. They should never be set by an application.)

GtkWidget *`entry`; the text entry field.

GtkWidget *`list`; the list shown in the drop-down window.

---

gtk_combo_new ()

```
GtkWidget*  gtk_combo_new                   (void);
```

Warning

`gtk_combo_new` is deprecated and should not be used in newly-written code. Use GtkComboBox instead.

Creates a new GtkCombo.

Returns : a new GtkCombo.

---

gtk_combo_set_popdown_strings ()

```
void        gtk_combo_set_popdown_strings   (GtkCombo *combo,
                                             GList *strings);
```

Warning

`gtk_combo_set_popdown_strings` is deprecated and should not be used in newly-written code. Use GtkComboBox instead.

Convenience function to set all of the items in the popup list. (See the example above.)

`combo` : a GtkCombo.

strings : a list of strings, or NULL to clear the popup list

---

gtk_combo_set_value_in_list ()

```
void        gtk_combo_set_value_in_list     (GtkCombo *combo,
                                             gboolean val,
                                             gboolean ok_if_empty);
```

Warning

gtk_combo_set_value_in_list is deprecated and should not be used in newly-written code. Use GtkComboBox instead.

Specifies whether the value entered in the text entry field must match one of the values in the list. If this is set then the user will not be able to perform any other action until a valid value has been entered.

If an empty field is acceptable, the *ok_if_empty* parameter should be TRUE.

combo : a GtkCombo.

val : TRUE if the value entered must match one of the values in the list.

ok_if_empty : TRUE if an empty value is considered valid.

---

gtk_combo_set_use_arrows ()

```
void        gtk_combo_set_use_arrows        (GtkCombo *combo,
                                             gboolean val);
```

Warning

gtk_combo_set_use_arrows is deprecated and should not be used in newly-written code. Use GtkComboBox instead.

Specifies if the arrow (cursor) keys can be used to step through the items in the list. This is on by default.

combo : a GtkCombo.

val : TRUE if the arrow keys can be used to step through the items in the list.

---

gtk_combo_set_use_arrows_always ()

```
void        gtk_combo_set_use_arrows_always (GtkCombo *combo,
                                             gboolean val);
```

Warning

`gtk_combo_set_use_arrows_always` is deprecated and should not be used in newly-written code. Use GtkComboBox instead.

Obsolete function, does nothing.

combo : a GtkCombo.

val : unused

---

gtk_combo_set_case_sensitive ()

```
void        gtk_combo_set_case_sensitive    (GtkCombo *combo,
                                             gboolean val);
```

Warning

`gtk_combo_set_case_sensitive` is deprecated and should not be used in newly-written code. Use GtkComboBox instead.

Specifies whether the text entered into the GtkEntry field and the text in the list items is case sensitive.

This may be useful, for example, when you have called `gtk_combo_set_value_in_list()` to limit the values entered, but you are not worried about differences in case.

combo : a GtkCombo.

val : TRUE if the text in the list items is case sensitive.

---

gtk_combo_set_item_string ()

```
void        gtk_combo_set_item_string       (GtkCombo *combo,
```

```
                                                GtkItem *item,
                                                const gchar *item_value);
```

Warning

`gtk_combo_set_item_string` is deprecated and should not be used in newly-written code. Use GtkComboBox instead.

Sets the string to place in the GtkEntry field when a particular list item is selected. This is needed if the list item is not a simple label.

combo : a GtkCombo.

item : a GtkItem.

item_value : the string to place in the GtkEntry when *item* is selected.

---

gtk_combo_disable_activate ()

```
void        gtk_combo_disable_activate      (GtkCombo *combo);
```

Warning

`gtk_combo_disable_activate` is deprecated and should not be used in newly-written code. Use GtkComboBox instead.

Stops the GtkCombo widget from showing the popup list when the GtkEntry emits the "activate" signal, i.e. when the Return key is pressed. This may be useful if, for example, you want the Return key to close a dialog instead.

combo : a GtkCombo.

Properties

The "allow-empty" property

```
  "allow-empty"          gboolean              : Read / Write
```

Whether an empty value may be entered in this field.

Default value: TRUE

The "case-sensitive" property

```
"case-sensitive"       gboolean              : Read / Write
```

Whether list item matching is case sensitive.

Default value: FALSE

The "enable-arrow-keys" property

```
"enable-arrow-keys"    gboolean              : Read / Write
```

Whether the arrow keys move through the list of items.

Default value: TRUE

The "enable-arrows-always" property

```
"enable-arrows-always" gboolean              : Read / Write
```

Obsolete property, ignored.

Default value: TRUE

The "value-in-list" property

```
"value-in-list"        gboolean              : Read / Write
```

Whether entered values must already be present in the list.

Default value: FALSE

<< GtkCTree **GtkItemFactory >>**

GtkItemFactory

GtkItemFactory — A factory for menus

Synopsis

```
#include <gtk/gtk.h>

            GtkItemFactory;
void        (*GtkPrintFunc)                 (gpointer func_data,
                                             const gchar *str);
gchar*      (*GtkTranslateFunc)             (const gchar *path,
                                             gpointer func_data);
void        (*GtkItemFactoryCallback)       ();
void        (*GtkItemFactoryCallback1)      (gpointer callback_data,
                                             guint callback_action,
                                             GtkWidget *widget);
void        (*GtkItemFactoryCallback2)      (GtkWidget *widget,
                                             gpointer callback_data,
                                             guint callback_action);
            GtkItemFactoryEntry;
            GtkItemFactoryItem;
GtkItemFactory* gtk_item_factory_new        (GType container_type,
                                             const gchar *path,
                                             GtkAccelGroup *accel_group);
void        gtk_item_factory_construct      (GtkItemFactory *ifactory,
                                             GType container_type,
                                             const gchar *path,
                                             GtkAccelGroup *accel_group);
void        gtk_item_factory_add_foreign    (GtkWidget *accel_widget,
                                             const gchar *full_path,
                                             GtkAccelGroup *accel_group,
                                             guint keyval,
                                             GdkModifierType modifiers);
GtkItemFactory* gtk_item_factory_from_widget
```

```
                                                    (GtkWidget *widget);
G_CONST_RETURN gchar* gtk_item_factory_path_from_widget
                                                    (GtkWidget *widget);
GtkWidget*  gtk_item_factory_get_item       (GtkItemFactory *ifactory,
                                             const gchar *path);
GtkWidget*  gtk_item_factory_get_widget     (GtkItemFactory *ifactory,
                                             const gchar *path);
GtkWidget*  gtk_item_factory_get_widget_by_action
                                            (GtkItemFactory *ifactory,
                                             guint action);
GtkWidget*  gtk_item_factory_get_item_by_action
                                            (GtkItemFactory *ifactory,
                                             guint action);
void        gtk_item_factory_create_item    (GtkItemFactory *ifactory,
                                             GtkItemFactoryEntry *entry,
                                             gpointer callback_data,
                                             guint callback_type);
void        gtk_item_factory_create_items   (GtkItemFactory *ifactory,
                                             guint n_entries,
                                             GtkItemFactoryEntry *entries,
                                             gpointer callback_data);
void        gtk_item_factory_create_items_ac
                                            (GtkItemFactory *ifactory,
                                             guint n_entries,
                                             GtkItemFactoryEntry *entries,
                                             gpointer callback_data,
                                             guint callback_type);
void        gtk_item_factory_delete_item    (GtkItemFactory *ifactory,
                                             const gchar *path);
void        gtk_item_factory_delete_entry   (GtkItemFactory *ifactory,
                                             GtkItemFactoryEntry *entry);
void        gtk_item_factory_delete_entries (GtkItemFactory *ifactory,
                                             guint n_entries,
                                             GtkItemFactoryEntry *entries);
void        gtk_item_factory_popup          (GtkItemFactory *ifactory,
                                             guint x,
                                             guint y,
                                             guint mouse_button,
                                             guint32 time_);
void        gtk_item_factory_popup_with_data
                                            (GtkItemFactory *ifactory,
                                             gpointer popup_data,
```

```
                                                GtkDestroyNotify destroy,
                                                guint x,
                                                guint y,
                                                guint mouse_button,
                                                guint32 time_);
gpointer    gtk_item_factory_popup_data     (GtkItemFactory *ifactory);
gpointer    gtk_item_factory_popup_data_from_widget
                                            (GtkWidget *widget);
GtkItemFactory* gtk_item_factory_from_path  (const gchar *path);
void        gtk_item_factory_create_menu_entries
                                            (guint n_entries,
                                             GtkMenuEntry *entries);
void        gtk_item_factories_path_delete  (const gchar *ifactory_path,
                                             const gchar *path);
void        gtk_item_factory_set_translate_func
                                            (GtkItemFactory *ifactory,
                                             GtkTranslateFunc func,
                                             gpointer data,
                                             GtkDestroyNotify notify);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkItemFactory
```

Description

As of GTK+ 2.4, GtkItemFactory has been deprecated in favour of GtkUIManager.

Details

GtkItemFactory

```
typedef struct _GtkItemFactory GtkItemFactory;
```

Warning

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

GtkPrintFunc ()

```
void        (*GtkPrintFunc)                 (gpointer func_data,
                                             const gchar *str);
```

Warning

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

`func_data` :
`str` :

GtkTranslateFunc ()

```
gchar*      (*GtkTranslateFunc)             (const gchar *path,
                                             gpointer func_data);
```

`path` :
`func_data` :
Returns :

GtkItemFactoryCallback ()

```
void        (*GtkItemFactoryCallback)       ();
```

Warning

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

GtkItemFactoryCallback1 ()

```
void        (*GtkItemFactoryCallback1)      (gpointer callback_data,
                                             guint callback_action,
                                             GtkWidget *widget);
```

Warning

`GtkItemFactoryCallback1` is deprecated and should not be used in newly-written code.

`callback_data`:
`callback_action`:
`widget`:

GtkItemFactoryCallback2 ()

```
void        (*GtkItemFactoryCallback2)      (GtkWidget *widget,
                                             gpointer callback_data,
                                             guint callback_action);
```

Warning

`GtkItemFactoryCallback2` is deprecated and should not be used in newly-written code.

`widget`:
`callback_data`:
`callback_action`:

GtkItemFactoryEntry

```
typedef struct {
  gchar *path;
```

```
  gchar *accelerator;

  GtkItemFactoryCallback callback;
  guint                  callback_action;

  /* possible values:
   * NULL               -> "<Item>"
   * ""                 -> "<Item>"
   * "<Title>"          -> create a title item
   * "<Item>"           -> create a simple item
   * "<ImageItem>"      -> create an item holding an image
   * "<StockItem>"      -> create an item holding a stock image
   * "<CheckItem>"      -> create a check item
   * "<ToggleItem>"     -> create a toggle item
   * "<RadioItem>"      -> create a radio item
   * <path>             -> path of a radio item to link against
   * "<Separator>"      -> create a separator
   * "<Tearoff>"        -> create a tearoff separator
   * "<Branch>"         -> create an item to hold sub items
   * "<LastBranch>"     -> create a right justified item to hold sub items
   */
  gchar          *item_type;

  /* Extra data for some item types:
   *  ImageItem  -> pointer to inlined pixbuf stream
   *  StockItem  -> name of stock item
   */
  gconstpointer extra_data;
} GtkItemFactoryEntry;
```

Warning

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

GtkItemFactoryItem

```
typedef struct {
  gchar *path;
  GSList *widgets;
} GtkItemFactoryItem;
```

Warning

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

---

gtk_item_factory_new ()

```
GtkItemFactory* gtk_item_factory_new        (GType container_type,
                                             const gchar *path,
                                             GtkAccelGroup *accel_group);
```

Warning

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

Creates a new GtkItemFactory.

Beware that the returned object does not have a floating reference.

container_type :	the kind of menu to create; can be GTK_TYPE_MENU_BAR, GTK_TYPE_MENU or GTK_TYPE_OPTION_MENU
path :	the factory path of the new item factory, a string of the form `"<name>"`
accel_group :	a GtkAccelGroup to which the accelerators for the menu items will be added, or `NULL` to create a new one
Returns :	a new GtkItemFactory

---

gtk_item_factory_construct ()

```
void        gtk_item_factory_construct      (GtkItemFactory *ifactory,
                                             GType container_type,
                                             const gchar *path,
                                             GtkAccelGroup *accel_group);
```

Warning

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

Initializes an item factory.

ifactory :	a GtkItemFactory
container_type :	the kind of menu to create; can be GTK_TYPE_MENU_BAR, GTK_TYPE_MENU or GTK_TYPE_OPTION_MENU
path :	the factory path of *ifactory*, a string of the form "<name>"
accel_group :	a GtkAccelGroup to which the accelerators for the menu items will be added, or NULL to create a new one

---

gtk_item_factory_add_foreign ()

```
void        gtk_item_factory_add_foreign    (GtkWidget *accel_widget,
                                             const gchar *full_path,
                                             GtkAccelGroup *accel_group,
                                             guint keyval,
                                             GdkModifierType modifiers);
```

Warning

gtk_item_factory_add_foreign is deprecated and should not be used in newly-written code. The recommended API for this purpose are the functions gtk_menu_item_set_accel_path() and gtk_widget_set_accel_path(); don't use gtk_item_factory_add_foreign() in new code, since it is likely to be removed in the future.

Installs an accelerator for *accel_widget* in *accel_group*, that causes the ::activate signal to be emitted if the accelerator is activated.

This function can be used to make widgets participate in the accel saving/restoring functionality provided by gtk_accel_map_save() and gtk_accel_map_load(), even if they haven't been created by an item factory.

accel_widget :	widget to install an accelerator on
full_path :	the full path for the *accel_widget*
accel_group :	the accelerator group to install the accelerator in
keyval :	key value of the accelerator
modifiers :	modifier combination of the accelerator

---

gtk_item_factory_from_widget ()

```
GtkItemFactory* gtk_item_factory_from_widget
                                                (GtkWidget *widget);
```

Warning

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

Obtains the item factory from which a widget was created.

widget : a widget

Returns : the item factory from which *widget* was created, or `NULL`

---

gtk_item_factory_path_from_widget ()

```
G_CONST_RETURN gchar* gtk_item_factory_path_from_widget
                                                (GtkWidget *widget);
```

Warning

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

If *widget* has been created by an item factory, returns the full path to it. (The full path of a widget is the concatenation of the factory path specified in `gtk_item_factory_new()` with the path specified in the GtkItemFactoryEntry from which the widget was created.)

widget : a widget

Returns : the full path to *widget* if it has been created by an item factory, `NULL` otherwise. This value is owned by GTK+ and must not be modified or freed.

---

gtk_item_factory_get_item ()

```
GtkWidget*  gtk_item_factory_get_item       (GtkItemFactory *ifactory,
                                             const gchar *path);
```

Warning

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

Obtains the menu item which corresponds to *`path`*.

If the widget corresponding to *`path`* is a menu item which opens a submenu, then the item is returned. If you are interested in the submenu, use `gtk_item_factory_get_widget()` instead.

`ifactory` : a GtkItemFactory
`path` : the path to the menu item
Returns : the menu item for the given path, or `NULL` if *`path`* doesn't lead to a menu item

---

gtk_item_factory_get_widget ()

```
GtkWidget*  gtk_item_factory_get_widget     (GtkItemFactory *ifactory,
                                             const gchar *path);
```

Warning

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

Obtains the widget which corresponds to *`path`*.

If the widget corresponding to *`path`* is a menu item which opens a submenu, then the submenu is returned. If you are interested in the menu item, use `gtk_item_factory_get_item()` instead.

`ifactory` : a GtkItemFactory
`path` : the path to the widget
Returns : the widget for the given path, or `NULL` if *`path`* doesn't lead to a widget

---

gtk_item_factory_get_widget_by_action ()

```
GtkWidget*  gtk_item_factory_get_widget_by_action
                                                (GtkItemFactory *ifactory,
                                                 guint action);
```

Warning

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

Obtains the widget which was constructed from the GtkItemFactoryEntry with the given *`action`*.

If there are multiple items with the same action, the result is undefined.

`ifactory` : a GtkItemFactory

`action` : an action as specified in the *`callback_action`* field of GtkItemFactoryEntry

Returns : the widget which corresponds to the given action, or `NULL` if no widget was found

---

gtk_item_factory_get_item_by_action ()

```
GtkWidget*  gtk_item_factory_get_item_by_action
                                                (GtkItemFactory *ifactory,
                                                 guint action);
```

Warning

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

Obtains the menu item which was constructed from the first GtkItemFactoryEntry with the given *`action`*.

`ifactory` : a GtkItemFactory

`action` : an action as specified in the *`callback_action`* field of GtkItemFactoryEntry

Returns : the menu item which corresponds to the given action, or `NULL` if no menu item was found

---

gtk_item_factory_create_item ()

```
void        gtk_item_factory_create_item    (GtkItemFactory *ifactory,
                                             GtkItemFactoryEntry *entry,
                                             gpointer callback_data,
                                             guint callback_type);
```

Warning

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

Creates an item for *`entry`*.

`ifactory` :	a GtkItemFactory
`entry` :	the GtkItemFactoryEntry to create an item for
`callback_data` :	data passed to the callback function of *`entry`*
`callback_type` :	1 if the callback function of *`entry`* is of type GtkItemFactoryCallback1, 2 if it is of type GtkItemFactoryCallback2

---

gtk_item_factory_create_items ()

```
void        gtk_item_factory_create_items   (GtkItemFactory *ifactory,
                                             guint n_entries,
                                             GtkItemFactoryEntry *entries,
                                             gpointer callback_data);
```

Warning

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

Creates the menu items from the *`entries`*.

`ifactory` :	a GtkItemFactory
`n_entries` :	the length of *`entries`*

entries :	an array of GtkItemFactoryEntrys whose *callback* members must by of type GtkItemFactoryCallback1
callback_data :	data passed to the callback functions of all entries

---

gtk_item_factory_create_items_ac ()

```
void        gtk_item_factory_create_items_ac
                                            (GtkItemFactory *ifactory,
                                             guint n_entries,
                                             GtkItemFactoryEntry *entries,
                                             gpointer callback_data,
                                             guint callback_type);
```

Warning

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

Creates the menu items from the *entries*.

ifactory :	a GtkItemFactory
n_entries :	the length of *entries*
entries :	an array of GtkItemFactoryEntrys
callback_data :	data passed to the callback functions of all entries
callback_type :	1 if the callback functions in *entries* are of type GtkItemFactoryCallback1, 2 if they are of type GtkItemFactoryCallback2

---

gtk_item_factory_delete_item ()

```
void        gtk_item_factory_delete_item    (GtkItemFactory *ifactory,
                                             const gchar *path);
```

Warning

gtk_item_factory_delete_item is deprecated and should not be used in newly-written

code.

Deletes the menu item which was created for *path* by the given item factory.

ifactory : a GtkItemFactory
path : a path

---

gtk_item_factory_delete_entry ()

```
void        gtk_item_factory_delete_entry   (GtkItemFactory *ifactory,
                                             GtkItemFactoryEntry *entry);
```

Warning

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

Deletes the menu item which was created from *entry* by the given item factory.

ifactory : a GtkItemFactory
entry : a GtkItemFactoryEntry

---

gtk_item_factory_delete_entries ()

```
void        gtk_item_factory_delete_entries (GtkItemFactory *ifactory,
                                             guint n_entries,
                                             GtkItemFactoryEntry *entries);
```

Warning

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

Deletes the menu items which were created from the *entries* by the given item factory.

ifactory : a GtkItemFactory
n_entries : the length of *entries*
entries : an array of GtkItemFactoryEntrys

---

gtk_item_factory_popup ()

```
void        gtk_item_factory_popup          (GtkItemFactory *ifactory,
                                             guint x,
                                             guint y,
                                             guint mouse_button,
                                             guint32 time_);
```

Warning

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

Pops up the menu constructed from the item factory at (*x*, *y*).

The *mouse_button* parameter should be the mouse button pressed to initiate the menu popup. If the menu popup was initiated by something other than a mouse button press, such as a mouse button release or a keypress, *mouse_button* should be 0.

The *time_* parameter should be the time stamp of the event that initiated the popup. If such an event is not available, use `gtk_get_current_event_time()` instead.

The operation of the *mouse_button* and the *time_* parameter is the same as the *button* and *activation_time* parameters for `gtk_menu_popup()`.

ifactory : a GtkItemFactory of type GTK_TYPE_MENU (see `gtk_item_factory_new()`)
x : the x position
y : the y position
mouse_button : the mouse button which was pressed to initiate the popup
*time_* : the time at which the activation event occurred

---

gtk_item_factory_popup_with_data ()

```
void        gtk_item_factory_popup_with_data
                                            (GtkItemFactory *ifactory,
                                             gpointer popup_data,
                                             GtkDestroyNotify destroy,
                                             guint x,
                                             guint y,
                                             guint mouse_button,
                                             guint32 time_);
```

Warning

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

Pops up the menu constructed from the item factory at (*x*, *y*). Callbacks can access the *popup_data* while the menu is posted via `gtk_item_factory_popup_data()` and `gtk_item_factory_popup_data_from_widget()`.

The *mouse_button* parameter should be the mouse button pressed to initiate the menu popup. If the menu popup was initiated by something other than a mouse button press, such as a mouse button release or a keypress, *mouse_button* should be 0.

The *time_* parameter should be the time stamp of the event that initiated the popup. If such an event is not available, use `gtk_get_current_event_time()` instead.

The operation of the *mouse_button* and the *time_* parameters is the same as the *button* and *activation_time* parameters for `gtk_menu_popup()`.

ifactory :	a GtkItemFactory of type GTK_TYPE_MENU (see `gtk_item_factory_new()`)
popup_data :	data available for callbacks while the menu is posted
destroy :	a GtkDestroyNotify function to be called on *popup_data* when the menu is unposted
x :	the x position
y :	the y position
mouse_button :	the mouse button which was pressed to initiate the popup
*time_* :	the time at which the activation event occurred

---

gtk_item_factory_popup_data ()

```
gpointer    gtk_item_factory_popup_data     (GtkItemFactory *ifactory);
```

Warning

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

Obtains the *`popup_data`* which was passed to `gtk_item_factory_popup_with_data()`. This data is available until the menu is popped down again.

`ifactory` : a GtkItemFactory

Returns : *`popup_data`* associated with *`ifactory`*

---

gtk_item_factory_popup_data_from_widget ()

```
gpointer    gtk_item_factory_popup_data_from_widget
                                            (GtkWidget *widget);
```

Warning

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

Obtains the *`popup_data`* which was passed to `gtk_item_factory_popup_with_data()`. This data is available until the menu is popped down again.

`widget` : a widget

Returns : *`popup_data`* associated with the item factory from which *`widget`* was created, or `NULL` if *`widget`* wasn't created by an item factory

---

gtk_item_factory_from_path ()

```
GtkItemFactory* gtk_item_factory_from_path  (const gchar *path);
```

Warning

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

Finds an item factory which has been constructed using the "<name>" prefix of *path* as the *path* argument for gtk_item_factory_new().

path : a string starting with a factory path of the form "<name>"
Returns : the GtkItemFactory created for the given factory path, or NULL

---

gtk_item_factory_create_menu_entries ()

```
void        gtk_item_factory_create_menu_entries
                                            (guint n_entries,
                                             GtkMenuEntry *entries);
```

Warning

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

Creates the menu items from the *entries*.

n_entries : the length of *entries*
entries : an array of GtkMenuEntrys

---

gtk_item_factories_path_delete ()

```
void        gtk_item_factories_path_delete  (const gchar *ifactory_path,
                                             const gchar *path);
```

Warning

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

Deletes all widgets constructed from the specified path.

ifactory_path : a factory path to prepend to *path*. May be NULL if *path* starts with a factory path

path : a path

---

gtk_item_factory_set_translate_func ()

```
void        gtk_item_factory_set_translate_func
                                            (GtkItemFactory *ifactory,
                                             GtkTranslateFunc func,
                                             gpointer data,
                                             GtkDestroyNotify notify);
```

Warning

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

Sets a function to be used for translating the path elements before they are displayed.

ifactory : a GtkItemFactory

func : the GtkTranslateFunc function to be used to translate path elements

data : data to pass to *func* and *notify*

notify : a GtkDestroyNotify function to be called when *ifactory* is destroyed and when the translation function is changed again

<< GtkCombo **GtkList >>**

GtkList

GtkList — Widget for packing a list of selectable items

Synopsis

```
#include <gtk/gtk.h>

            GtkList;
GtkWidget*  gtk_list_new                    (void);
void        gtk_list_insert_items           (GtkList *list,
                                             GList *items,
                                             gint position);
void        gtk_list_append_items           (GtkList *list,
                                             GList *items);
void        gtk_list_prepend_items          (GtkList *list,
                                             GList *items);
void        gtk_list_remove_items           (GtkList *list,
                                             GList *items);
void        gtk_list_remove_items_no_unref  (GtkList *list,
                                             GList *items);
void        gtk_list_clear_items            (GtkList *list,
                                             gint start,
                                             gint end);
void        gtk_list_select_item            (GtkList *list,
                                             gint item);
void        gtk_list_unselect_item          (GtkList *list,
                                             gint item);
void        gtk_list_select_child           (GtkList *list,
                                             GtkWidget *child);
void        gtk_list_unselect_child         (GtkList *list,
                                             GtkWidget *child);
gint        gtk_list_child_position         (GtkList *list,
                                             GtkWidget *child);
void        gtk_list_set_selection_mode     (GtkList *list,
```

```
                                        GtkSelectionMode mode);
void        gtk_list_extend_selection   (GtkList *list,
                                         GtkScrollType scroll_type,
                                         gfloat position,
                                         gboolean auto_start_selection);
void        gtk_list_start_selection    (GtkList *list);
void        gtk_list_end_selection      (GtkList *list);
void        gtk_list_select_all         (GtkList *list);
void        gtk_list_unselect_all       (GtkList *list);
void        gtk_list_scroll_horizontal  (GtkList *list,
                                         GtkScrollType scroll_type,
                                         gfloat position);
void        gtk_list_scroll_vertical    (GtkList *list,
                                         GtkScrollType scroll_type,
                                         gfloat position);
void        gtk_list_toggle_add_mode    (GtkList *list);
void        gtk_list_toggle_focus_row   (GtkList *list);
void        gtk_list_toggle_row         (GtkList *list,
                                         GtkWidget *item);
void        gtk_list_undo_selection     (GtkList *list);
void        gtk_list_end_drag_selection (GtkList *list);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkWidget
               +----GtkContainer
                     +----GtkList
```

Implemented Interfaces

GtkList implements AtkImplementorIface.

Properties

```
  "selection-mode"       GtkSelectionMode      : Read / Write
```

Signal Prototypes

```
"select-child"
            void        user_function      (GtkList *list,
                                            GtkWidget *widget,
                                            gpointer user_data);
"selection-changed"
            void        user_function      (GtkList *list,
                                            gpointer user_data);
"unselect-child"
            void        user_function      (GtkList *list,
                                            GtkWidget *widget,
                                            gpointer user_data);
```

Description

The GtkList widget is a container whose children are displayed vertically in order, and can be selected. The list has many selection modes, which are programmer selective and depend on how many elements are able to be selected at the same time.

GtkList has been deprecated since GTK+ 2.0 and should not be used in newly written code. Use GtkTreeView instead.

Details

GtkList

```
typedef struct _GtkList GtkList;
```

Warning

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

---

gtk_list_new ()

```
GtkWidget*  gtk_list_new                    (void);
```

Warning

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

Creates a new GtkList.

Returns : the newly-created GtkList

---

gtk_list_insert_items ()

```
void        gtk_list_insert_items           (GtkList *list,
                                             GList *items,
                                             gint position);
```

Warning

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

Inserts *`items`* into the *`list`* at the position *`position`*. The GList items must not be freed after.

`list` : the list widget.
`items` : the items.
`position` : the position to insert *`items`*, starting at 0.

---

gtk_list_append_items ()

```
void        gtk_list_append_items           (GtkList *list,
                                             GList *items);
```

Warning

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

Adds *`items`* to the end of the *`list`*.

`list` : the list widget.
`items` : the items.

---

gtk_list_prepend_items ()

```
void        gtk_list_prepend_items          (GtkList *list,
                                             GList *items);
```

Warning

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

Inserts *`items`* at the beginning of the *`list`*.

`list` : the list widget.
`items` : the items.

---

gtk_list_remove_items ()

```
void        gtk_list_remove_items           (GtkList *list,
                                             GList *items);
```

Warning

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

Removes the *`items`* from the *`list`*.

`list` : the list widget.
`items` : the items to remove.

---

gtk_list_remove_items_no_unref ()

```
void        gtk_list_remove_items_no_unref  (GtkList *list,
                                             GList *items);
```

Warning

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

Removes the *`items`* from the *`list`*, without unreferencing them. It may be useful if you want to move the items from one list to another.

`list` : the list widget.
`items` : the items.

---

gtk_list_clear_items ()

```
void        gtk_list_clear_items            (GtkList *list,
                                             gint start,
                                             gint end);
```

Warning

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

Removes the items between index *`start`* (included) and *`end`* (excluded) from the *`list`*. If *`end`* is negative, or greater than the number of children of *`list`*, it's assumed to be exactly the number of elements. If *`start`* is greater than or equal to *`end`*, nothing is done.

`list` : the list widget.
`start` : the index of the first item to remove.
`end` : the index of the lest item to remove plus one.

---

gtk_list_select_item ()

```
void        gtk_list_select_item            (GtkList *list,
                                             gint item);
```

Warning

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

Selects the child number *`item`* of the *`list`*. Nothing happens if *`item`* is out of bounds. The signal GtkList::select-child will be emitted.

`list` : the list widget.
`item` : the index of the child to select.

---

gtk_list_unselect_item ()

```
void        gtk_list_unselect_item          (GtkList *list,
                                             gint item);
```

Warning

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

Unselects the child number *`item`* of the *`list`*. Nothing happens if *`item`* is out of bounds. The signal GtkList::unselect-child will be emitted.

`list` : the list widget.
`item` : the index of the child to unselect.

---

gtk_list_select_child ()

```
void        gtk_list_select_child           (GtkList *list,
                                             GtkWidget *child);
```

Warning

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

Selects the given *child*. The signal GtkList::select-child will be emitted.

list : the list widget
child : the child to select.

---

gtk_list_unselect_child ()

```
void        gtk_list_unselect_child         (GtkList *list,
                                             GtkWidget *child);
```

Warning

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

Unselects the given *child*. The signal GtkList::unselect-child will be emitted.

list : the list widget.
child : the child to unselect.

---

gtk_list_child_position ()

```
gint        gtk_list_child_position         (GtkList *list,
                                             GtkWidget *child);
```

Warning

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

Searches the children of *list* for the index of *child*.

list : the list widget.
child : the child to look for.
Returns : the index of the child, -1 if not found.

---

gtk_list_set_selection_mode ()

```
void        gtk_list_set_selection_mode     (GtkList *list,
                                             GtkSelectionMode mode);
```

Warning

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

Set the list selection mode. The selection mode can be any value in GtkSelectionMode:

GTK_SELECTION_SINGLE	Zero or one element may be selected.
GTK_SELECTION_BROWSE	Exactly one element is always selected (this can be false after you have changed the selection mode).
GTK_SELECTION_MULTIPLE	Any number of elements may be selected. Clicks toggle the state of an item.
GTK_SELECTION_EXTENDED	Any number of elements may be selected. Click-drag selects a range of elements; the Ctrl key may be used to enlarge the selection, and Shift key to select between the focus and the child pointed to.

`list` : the list widget.
`mode` : the new selection mode.

---

gtk_list_extend_selection ()

```
void        gtk_list_extend_selection       (GtkList *list,
                                             GtkScrollType scroll_type,
                                             gfloat position,
                                             gboolean auto_start_selection);
```

Warning

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

Extends the selection by moving the anchor according to *`scroll_type`*. Only in GTK_SELECTION_EXTENDED.

list :	the list widget.
scroll_type :	the direction and length.
position :	the position if *scroll_type* is GTK_SCROLL_JUMP.
auto_start_selection :	if TRUE, gtk_list_start_selection() is automatically carried out before extending the selection.

---

gtk_list_start_selection ()

```
void        gtk_list_start_selection        (GtkList *list);
```

Warning

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

Starts a selection (or part of selection) at the focused child. Only in GTK_SELECTION_EXTENDED mode.

list : the list widget.

---

gtk_list_end_selection ()

```
void        gtk_list_end_selection          (GtkList *list);
```

Warning

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

Ends the selection. Used with gtk_list_extend_selection() and gtk_list_start_selection(). Only in GTK_SELECTION_EXTENDED mode.

list : the list widget.

---

gtk_list_select_all ()

```
void        gtk_list_select_all             (GtkList *list);
```

Warning

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

Selects all children of *`list`*. A signal will be emitted for each newly selected child.

`list` : the list widget.

---

gtk_list_unselect_all ()

```
void        gtk_list_unselect_all           (GtkList *list);
```

Warning

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

Unselects all children of *`list`*. A signal will be emitted for each newly unselected child.

`list` : the list widget.

---

gtk_list_scroll_horizontal ()

```
void        gtk_list_scroll_horizontal      (GtkList *list,
                                             GtkScrollType scroll_type,
                                             gfloat position);
```

Warning

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

Scrolls *`list`* horizontaly. This supposes that the list is packed into a scrolled window or something similar, and adjustments are well set. Step and page increment are those from the horizontal adjustment of *`list`*. Backward means to the left, and forward to the right. Out of bounds values are truncated. *`scroll_type`* may be any valid GtkScrollType. If *`scroll_type`* is GTK_SCROLL_NONE, nothing is done. If it's GTK_SCROLL_JUMP, the list scrolls to the ratio *`position`*: 0 is full left, 1 is full right.

list : the list widget.
scroll_type : the scrolling type.
position : the position if *scroll_type* is GTK_SCROLL_JUMP

---

gtk_list_scroll_vertical ()

```
void        gtk_list_scroll_vertical        (GtkList *list,
                                             GtkScrollType scroll_type,
                                             gfloat position);
```

Warning

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

Scrolls *list* vertically. This supposes that the list is packed into a scrolled window or something similar, and adjustments are well set. Step and page increment are those from the vertical adjustment of *list*. Backward means up, and forward down. Out of bounds values are truncated. *scroll_type* may be any valid GtkScrollType. If *scroll_type* is GTK_SCROLL_NONE, nothing is done. If it's GTK_SCROLL_JUMP, the list scrolls to the ratio *position*: 0 is top, 1 is bottom.

list : the list widget.
scroll_type : the scrolling type.
position : the position if *scroll_type* is GTK_SCROLL_JUMP

---

gtk_list_toggle_add_mode ()

```
void        gtk_list_toggle_add_mode        (GtkList *list);
```

Warning

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

Toggles between adding to the selection and beginning a new selection. Only in GTK_SELECTION_EXTENDED. Useful with `gtk_list_extend_selection()`.

list : the list widget.

gtk_list_toggle_focus_row ()

```
void        gtk_list_toggle_focus_row       (GtkList *list);
```

Warning

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

Toggles the focus row. If the focus row is selected, it's unselected. If the focus row is unselected, it's selected. If the selection mode of *`list`* is GTK_SELECTION_BROWSE, this has no effect, as the selection is always at the focus row.

`list` : the list widget.

gtk_list_toggle_row ()

```
void        gtk_list_toggle_row             (GtkList *list,
                                             GtkWidget *item);
```

Warning

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

Toggles the child *`item`* of list. If the selection mode of *`list`* is GTK_SELECTION_BROWSE, the item is selected, and the others are unselected.

`list` : the list widget.
`item` : the child to toggle.

gtk_list_undo_selection ()

```
void        gtk_list_undo_selection         (GtkList *list);
```

Warning

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

Restores the selection in the last state, only if selection mode is GTK_SELECTION_EXTENDED. If this function is called twice, the selection is cleared. This function sometimes gives stranges "last states".

`list` : the list widget.

---

gtk_list_end_drag_selection ()

```
void        gtk_list_end_drag_selection     (GtkList *list);
```

Warning

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

Stops the drag selection mode and ungrabs the pointer. This has no effect if a drag selection is not active.

`list` : the list widget.

Properties

The "selection-mode" property

```
  "selection-mode"       GtkSelectionMode      : Read / Write
```

Default value: GTK_SELECTION_NONE

Signals

The "select-child" signal

```
void        user_function                  (GtkList *list,
                                            GtkWidget *widget,
                                            gpointer user_data);
```

The child *widget* has just been selected.

list : the object which received the signal.
widget : the newly selected child.
user_data : user data set when the signal handler was connected.

The "selection-changed" signal

```
void        user_function                   (GtkList *list,
                                             gpointer user_data);
```

The selection of the widget has just changed.

list : the object which received the signal.
user_data : user data set when the signal handler was connected.

The "unselect-child" signal

```
void        user_function                   (GtkList *list,
                                             GtkWidget *widget,
                                             gpointer user_data);
```

The child *widget* has just been unselected.

list : the object which received the signal.
widget : the newly unselected child.
user_data : user data set when the signal handler was connected.

See Also

GtkContainer For functions that apply to every GtkContainer (like GtkList).

GtkListitem Children of a GtkList widget must be of this type.

<< GtkItemFactory **GtkListItem >>**

GtkListItem

GtkListItem — An item in a GtkList

Synopsis

```
#include <gtk/gtk.h>


            GtkListItem;
GtkWidget*  gtk_list_item_new               (void);
GtkWidget*  gtk_list_item_new_with_label    (const gchar *label);
void        gtk_list_item_select            (GtkListItem *list_item);
void        gtk_list_item_deselect          (GtkListItem *list_item);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkWidget
               +----GtkContainer
                     +----GtkBin
                           +----GtkItem
                                 +----GtkListItem
```

Implemented Interfaces

GtkListItem implements AtkImplementorIface.

Signal Prototypes

```
"end-selection"
            void        user_function      (GtkListItem *listitem,
                                            gpointer user_data);
"extend-selection"
            void        user_function      (GtkListItem *listitem,
                                            GtkScrollType scroll_type,
                                            gfloat position,
                                            gboolean auto_start_selection,
                                            gpointer user_data);
"scroll-horizontal"
            void        user_function      (GtkListItem *listitem,
                                            GtkScrollType scroll_type,
                                            gfloat position,
                                            gpointer user_data);
"scroll-vertical"
            void        user_function      (GtkListItem *listitem,
                                            GtkScrollType scroll_type,
                                            gfloat position,
                                            gpointer user_data);
"select-all"
            void        user_function      (GtkListItem *listitem,
                                            gpointer user_data);
"start-selection"
            void        user_function      (GtkListItem *listitem,
                                            gpointer user_data);
"toggle-add-mode"
            void        user_function      (GtkListItem *listitem,
                                            gpointer user_data);
"toggle-focus-row"
            void        user_function      (GtkListItem *listitem,
                                            gpointer user_data);
"undo-selection"
            void        user_function      (GtkListItem *listitem,
                                            gpointer user_data);
"unselect-all"
            void        user_function      (GtkListItem *listitem,
                                            gpointer user_data);
```

Description

The GtkListItem widget is used for each item in a GtkList.

GtkList has has been deprecated since GTK+ 2.0 and should not be used in newly written code. Use GtkTreeView instead.

Details

GtkListItem

```
typedef struct _GtkListItem GtkListItem;
```

Warning

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

The GtkListItem struct contains private data only, and should only be accessed using the functions below.

---

gtk_list_item_new ()

```
GtkWidget*  gtk_list_item_new               (void);
```

Warning

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

Creates a new GtkListitem.

Returns : a new GtkListItem.

---

gtk_list_item_new_with_label ()

```

```

```
GtkWidget*  gtk_list_item_new_with_label    (const gchar *label);
```

Warning

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

Creates a new GtkListItem with a child label containing the given string.

label : the string to use for the child label.

Returns : a new GtkListItem with a child GtkLabel with the text set to *label*.

---

gtk_list_item_select ()

```
void        gtk_list_item_select            (GtkListItem *list_item);
```

Warning

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

Selects the item, by emitting the item's "select" signal. Depending on the selection mode of the list, this may cause other items to be deselected.

list_item : a GtkListItem.

---

gtk_list_item_deselect ()

```
void        gtk_list_item_deselect          (GtkListItem *list_item);
```

Warning

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

Deselects the item, by emitting the item's "deselect" signal.

list_item : a GtkListItem.

Signals

The "end-selection" signal

```
void        user_function                   (GtkListItem *listitem,
                                             gpointer user_data);
```

listitem : the object which received the signal.
user_data : user data set when the signal handler was connected.

The "extend-selection" signal

```
void        user_function                   (GtkListItem *listitem,
                                             GtkScrollType scroll_type,
                                             gfloat position,
                                             gboolean auto_start_selection,
                                             gpointer user_data);
```

listitem : the object which received the signal.
scroll_type :
position :
auto_start_selection :
user_data : user data set when the signal handler was connected.

The "scroll-horizontal" signal

```
void        user_function                   (GtkListItem *listitem,
                                             GtkScrollType scroll_type,
                                             gfloat position,
                                             gpointer user_data);
```

listitem: the object which received the signal.
scroll_type:
position:
user_data: user data set when the signal handler was connected.

The "scroll-vertical" signal

```
void        user_function                   (GtkListItem *listitem,
                                             GtkScrollType scroll_type,
                                             gfloat position,
                                             gpointer user_data);
```

listitem: the object which received the signal.
scroll_type:
position:
user_data: user data set when the signal handler was connected.

The "select-all" signal

```
void        user_function                   (GtkListItem *listitem,
                                             gpointer user_data);
```

listitem: the object which received the signal.
user_data: user data set when the signal handler was connected.

The "start-selection" signal

```
void        user_function                   (GtkListItem *listitem,
                                             gpointer user_data);
```

listitem: the object which received the signal.

user_data : user data set when the signal handler was connected.

The "toggle-add-mode" signal

```
void        user_function                   (GtkListItem *listitem,
                                             gpointer user_data);
```

listitem : the object which received the signal.
user_data : user data set when the signal handler was connected.

The "toggle-focus-row" signal

```
void        user_function                   (GtkListItem *listitem,
                                             gpointer user_data);
```

listitem : the object which received the signal.
user_data : user data set when the signal handler was connected.

The "undo-selection" signal

```
void        user_function                   (GtkListItem *listitem,
                                             gpointer user_data);
```

listitem : the object which received the signal.
user_data : user data set when the signal handler was connected.

The "unselect-all" signal

```
void        user_function                   (GtkListItem *listitem,
                                             gpointer user_data);
```

listitem: the object which received the signal.
user_data: user data set when the signal handler was connected.

See Also

GtkList the parent list widget.

<< GtkList **GtkOldEditable >>**

GtkOldEditable

GtkOldEditable — Base class for text-editing widgets

Synopsis

```
#include <gtk/gtk.h>

            GtkOldEditable;
void        (*GtkTextFunction)              (GtkOldEditable *editable,
                                             guint32 time_);
void        gtk_old_editable_claim_selection
                                            (GtkOldEditable *old_editable,
                                             gboolean claim,
                                             guint32 time_);
void        gtk_old_editable_changed        (GtkOldEditable *old_editable);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkWidget
               +----GtkOldEditable
                     +----GtkText
```

Implemented Interfaces

GtkOldEditable implements AtkImplementorIface and GtkEditable.

Properties

```
  "editable"            gboolean              : Read / Write
  "text-position"       gint                  : Read / Write
```

Signal Prototypes

```
"activate"  void        user_function     (GtkOldEditable *oldeditable,
                                           gpointer user_data);
"copy-clipboard"
            void        user_function     (GtkOldEditable *oldeditable,
                                           gpointer user_data);
"cut-clipboard"
            void        user_function     (GtkOldEditable *oldeditable,
                                           gpointer user_data);
"kill-char" void        user_function     (GtkOldEditable *oldeditable,
                                           gint arg1,
                                           gpointer user_data);
"kill-line" void        user_function     (GtkOldEditable *oldeditable,
                                           gint arg1,
                                           gpointer user_data);
"kill-word" void        user_function     (GtkOldEditable *oldeditable,
                                           gint arg1,
                                           gpointer user_data);
"move-cursor"
            void        user_function     (GtkOldEditable *oldeditable,
                                           gint arg1,
                                           gint arg2,
                                           gpointer user_data);
"move-page" void        user_function     (GtkOldEditable *oldeditable,
                                           gint arg1,
                                           gint arg2,
                                           gpointer user_data);
"move-to-column"
            void        user_function     (GtkOldEditable *oldeditable,
                                           gint arg1,
                                           gpointer user_data);
```

```
"move-to-row"
            void        user_function      (GtkOldEditable *oldeditable,
                                            gint arg1,
                                            gpointer user_data);
"move-word" void        user_function      (GtkOldEditable *oldeditable,
                                            gint arg1,
                                            gpointer user_data);
"paste-clipboard"
            void        user_function      (GtkOldEditable *oldeditable,
                                            gpointer user_data);
"set-editable"
            void        user_function      (GtkOldEditable *oldeditable,
                                            gboolean arg1,
                                            gpointer user_data);
```

Description

GtkOldEditable has been deprecated since GTK+ 2.0 and should not be used in newly written code. Use the GtkEditable interface instead.

Details

GtkOldEditable

```
typedef struct {
  guint      current_pos;

  guint      selection_start_pos;
  guint      selection_end_pos;
  guint      has_selection : 1;
} GtkOldEditable;
```

Warning

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

GtkTextFunction ()

```
void        (*GtkTextFunction)              (GtkOldEditable *editable,
                                             guint32 time_);
```

Warning

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

editable :
*time_* :

---

gtk_old_editable_claim_selection ()

```
void        gtk_old_editable_claim_selection
                                            (GtkOldEditable *old_editable,
                                             gboolean claim,
                                             guint32 time_);
```

Warning

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

Claims or gives up ownership of the selection.

old_editable : a GtkOldEditable
claim : if TRUE, claim ownership of the selection, if FALSE, give up ownership
*time_* : timestamp for this operation

---

gtk_old_editable_changed ()

```
void        gtk_old_editable_changed        (GtkOldEditable *old_editable);
```

Warning

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

Emits the ::changed signal on *`old_editable`*.

`old_editable` : a GtkOldEditable

Properties

The "editable" property

```
  "editable"            gboolean              : Read / Write
```

Default value: FALSE

The "text-position" property

```
  "text-position"       gint                  : Read / Write
```

Allowed values: >= -2147483647

Default value: 0

Signals

The "activate" signal

```
void        user_function                     (GtkOldEditable *oldeditable,
                                               gpointer user_data);
```

`oldeditable` : the object which received the signal.
`user_data` : user data set when the signal handler was connected.

The "copy-clipboard" signal

```
void        user_function                   (GtkOldEditable *oldeditable,
                                             gpointer user_data);
```

oldeditable : the object which received the signal.
user_data : user data set when the signal handler was connected.

The "cut-clipboard" signal

```
void        user_function                   (GtkOldEditable *oldeditable,
                                             gpointer user_data);
```

oldeditable : the object which received the signal.
user_data : user data set when the signal handler was connected.

The "kill-char" signal

```
void        user_function                   (GtkOldEditable *oldeditable,
                                             gint arg1,
                                             gpointer user_data);
```

oldeditable : the object which received the signal.
arg1 :
user_data : user data set when the signal handler was connected.

The "kill-line" signal

```
void        user_function                   (GtkOldEditable *oldeditable,
                                             gint arg1,
                                             gpointer user_data);
```

oldeditable : the object which received the signal.
arg1 :
user_data : user data set when the signal handler was connected.

The "kill-word" signal

```
void        user_function                   (GtkOldEditable *oldeditable,
                                             gint arg1,
                                             gpointer user_data);
```

oldeditable : the object which received the signal.
arg1 :
user_data : user data set when the signal handler was connected.

The "move-cursor" signal

```
void        user_function                   (GtkOldEditable *oldeditable,
                                             gint arg1,
                                             gint arg2,
                                             gpointer user_data);
```

oldeditable : the object which received the signal.
arg1 :
arg2 :
user_data : user data set when the signal handler was connected.

The "move-page" signal

```
void        user_function                   (GtkOldEditable *oldeditable,
                                             gint arg1,
                                             gint arg2,
                                             gpointer user_data);
```

oldeditable : the object which received the signal.
arg1 :
arg2 :
user_data : user data set when the signal handler was connected.

The "move-to-column" signal

```
void        user_function                   (GtkOldEditable *oldeditable,
                                             gint arg1,
                                             gpointer user_data);
```

oldeditable : the object which received the signal.
arg1 :
user_data : user data set when the signal handler was connected.

The "move-to-row" signal

```
void        user_function                   (GtkOldEditable *oldeditable,
                                             gint arg1,
                                             gpointer user_data);
```

oldeditable : the object which received the signal.
arg1 :
user_data : user data set when the signal handler was connected.

The "move-word" signal

```
void        user_function                   (GtkOldEditable *oldeditable,
                                             gint arg1,
                                             gpointer user_data);
```

oldeditable : the object which received the signal.

arg1 :
user_data : user data set when the signal handler was connected.

The "paste-clipboard" signal

```
void        user_function                   (GtkOldEditable *oldeditable,
                                             gpointer user_data);
```

oldeditable : the object which received the signal.
user_data : user data set when the signal handler was connected.

The "set-editable" signal

```
void        user_function                   (GtkOldEditable *oldeditable,
                                             gboolean arg1,
                                             gpointer user_data);
```

oldeditable : the object which received the signal.
arg1 :
user_data : user data set when the signal handler was connected.

<< GtkListItem **GtkOptionMenu >>**

GtkOptionMenu

GtkOptionMenu — A widget used to choose from a list of valid choices

Synopsis

```
#include <gtk/gtk.h>


            GtkOptionMenu;
GtkWidget*  gtk_option_menu_new             (void);
GtkWidget*  gtk_option_menu_get_menu        (GtkOptionMenu *option_menu);
void        gtk_option_menu_set_menu        (GtkOptionMenu *option_menu,
                                             GtkWidget *menu);
void        gtk_option_menu_remove_menu     (GtkOptionMenu *option_menu);
void        gtk_option_menu_set_history     (GtkOptionMenu *option_menu,
                                             guint index_);
gint        gtk_option_menu_get_history     (GtkOptionMenu *option_menu);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkWidget
               +----GtkContainer
                     +----GtkBin
                           +----GtkButton
                                 +----GtkOptionMenu
```

Implemented Interfaces

GtkOptionMenu implements AtkImplementorIface.

Properties

```
  "menu"                GtkMenu               : Read / Write
```

Style Properties

```
  "indicator-size"      GtkRequisition        : Read
  "indicator-spacing"   GtkBorder             : Read
```

Signal Prototypes

```
"changed"   void        user_function      (GtkOptionMenu *optionmenu,
                                            gpointer user_data);
```

Description

A GtkOptionMenu is a widget that allows the user to choose from a list of valid choices. The GtkOptionMenu displays the selected choice. When activated the GtkOptionMenu displays a popup GtkMenu which allows the user to make a new choice.

Using a GtkOptionMenu is simple; build a GtkMenu, by calling `gtk_menu_new()`, then appending menu items to it with `gtk_menu_shell_append()`. Set that menu on the option menu with `gtk_option_menu_set_menu()`. Set the selected menu item with `gtk_option_menu_set_history()`; connect to the "changed" signal on the option menu; in the "changed" signal, check the new selected menu item with `gtk_option_menu_get_history()`.

As of GTK+ 2.4, GtkOptionMenu has been deprecated in favor of GtkComboBox.

Details

GtkOptionMenu

```
typedef struct _GtkOptionMenu GtkOptionMenu;
```

Warning

`GtkOptionMenu` is deprecated and should not be used in newly-written code. Use GtkComboBox instead.

The GtkOptionMenu-struct struct contains private data only, and should be accessed using the functions below.

---

gtk_option_menu_new ()

```
GtkWidget*  gtk_option_menu_new             (void);
```

Warning

`gtk_option_menu_new` is deprecated and should not be used in newly-written code. Use GtkComboBox instead.

Creates a new GtkOptionMenu.

Returns : a new GtkOptionMenu.

---

gtk_option_menu_get_menu ()

```
GtkWidget*  gtk_option_menu_get_menu        (GtkOptionMenu *option_menu);
```

Warning

`gtk_option_menu_get_menu` is deprecated and should not be used in newly-written code. Use GtkComboBox instead.

Returns the GtkMenu associated with the GtkOptionMenu.

option_menu : a GtkOptionMenu.

Returns : the GtkMenu associated with the GtkOptionMenu.

---

gtk_option_menu_set_menu ()

```
void        gtk_option_menu_set_menu        (GtkOptionMenu *option_menu,
                                             GtkWidget *menu);
```

Warning

`gtk_option_menu_set_menu` is deprecated and should not be used in newly-written code. Use GtkComboBox instead.

Provides the GtkMenu that is popped up to allow the user to choose a new value. You should provide a simple menu avoiding the use of tearoff menu items, submenus, and accelerators.

option_menu : a GtkOptionMenu.

menu : the GtkMenu to associate with the GtkOptionMenu.

---

gtk_option_menu_remove_menu ()

```
void        gtk_option_menu_remove_menu     (GtkOptionMenu *option_menu);
```

Warning

`gtk_option_menu_remove_menu` is deprecated and should not be used in newly-written code. Use GtkComboBox instead.

Removes the menu from the option menu.

option_menu : a GtkOptionMenu.

---

gtk_option_menu_set_history ()

```
void        gtk_option_menu_set_history     (GtkOptionMenu *option_menu,
                                             guint index_);
```

Warning

`gtk_option_menu_set_history` is deprecated and should not be used in newly-written code. Use GtkComboBox instead.

Selects the menu item specified by *index_* making it the newly selected value for the option menu.

option_menu : a GtkOptionMenu.

*index_* : the index of the menu item to select. Index values are from 0 to n-1.

---

gtk_option_menu_get_history ()

```
gint        gtk_option_menu_get_history     (GtkOptionMenu *option_menu);
```

Warning

`gtk_option_menu_get_history` is deprecated and should not be used in newly-written code. Use GtkComboBox instead.

Retrieves the index of the currently selected menu item. The menu items are numbered from top to bottom, starting with 0.

option_menu : a GtkOptionMenu

Returns : index of the selected menu item, or -1 if there are no menu items

Properties

The "menu" property

```
  "menu"                GtkMenu              : Read / Write
```

The menu of options.

Style Properties

The "indicator-size" style property

```
  "indicator-size"      GtkRequisition       : Read
```

Size of dropdown indicator.

The "indicator-spacing" style property

```
  "indicator-spacing"   GtkBorder            : Read
```

Spacing around indicator.

Signals

The "changed" signal

```
void        user_function                  (GtkOptionMenu *optionmenu,
                                            gpointer user_data);
```

optionmenu : the object which received the signal.
user_data : user data set when the signal handler was connected.

<< GtkOldEditable **GtkPixmap >>**

GtkPixmap

GtkPixmap — A widget displaying a graphical image or icon

Synopsis

```
#include <gtk/gtk.h>


            GtkPixmap;
GtkWidget*  gtk_pixmap_new                  (GdkPixmap *pixmap,
                                             GdkBitmap *mask);
void        gtk_pixmap_set                  (GtkPixmap *pixmap,
                                             GdkPixmap *val,
                                             GdkBitmap *mask);
void        gtk_pixmap_get                  (GtkPixmap *pixmap,
                                             GdkPixmap **val,
                                             GdkBitmap **mask);
void        gtk_pixmap_set_build_insensitive
                                            (GtkPixmap *pixmap,
                                             gboolean build);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkWidget
               +----GtkMisc
```

```
                    +----GtkPixmap
```

Implemented Interfaces

GtkPixmap implements AtkImplementorIface.

Description

The GtkPixmap widget displays a graphical image or icon. The icon is typically created using `gdk_pixmap_colormap_create_from_xpm()` or `gdk_pixmap_colormap_create_from_xpm_d()`.

The pixels in a GtkPixmap cannot be manipulated by the application after creation, since under the X Window system the pixel data is stored on the X server and so is not available to the client application. If you want to create graphical images which can be manipulated by the application, look at GtkImage and GdkRGB.

GtkPixmap has been deprecated since GTK+ 2.0 and should not be used in newly written code. Use GtkImage instead.

Details

GtkPixmap

```
typedef struct _GtkPixmap GtkPixmap;
```

Warning

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

The GtkPixmap-struct struct contains private data only, and should be accessed using the functions below.

---

gtk_pixmap_new ()

```
GtkWidget*  gtk_pixmap_new                  (GdkPixmap *pixmap,
                                             GdkBitmap *mask);
```

Warning

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

Creates a new GtkPixmap, using the given GDK pixmap and mask.

`pixmap` : a GDKPixmap.

`mask` : a GDKBitmap which indicates which parts of the *`pixmap`* should be transparent.

Returns : a new GtkPixmap.

---

gtk_pixmap_set ()

```
void        gtk_pixmap_set                  (GtkPixmap *pixmap,
                                             GdkPixmap *val,
                                             GdkBitmap *mask);
```

Warning

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

Sets the GdkPixmap and GdkBitmap mask.

`pixmap` : a GtkPixmap.

`val` : a GdkPixmap.

`mask` : a GdkBitmap, which indicates which parts of the *`pixmap`* should be transparent. This can be NULL, in which case none of the *`pixmap`* is transparent.

gtk_pixmap_get ()

```
void        gtk_pixmap_get                      (GtkPixmap *pixmap,
                                                 GdkPixmap **val,
                                                 GdkBitmap **mask);
```

Warning

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

Gets the current GdkPixmap and GdkBitmap mask.

`pixmap` : a GtkPixmap.

`val` : returns the current GdkPixmap.

`mask` : returns the current GdkBitmap mask.

---

gtk_pixmap_set_build_insensitive ()

```
void        gtk_pixmap_set_build_insensitive
                                            (GtkPixmap *pixmap,
                                             gboolean build);
```

Warning

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

Sets wether an extra pixmap should be automatically created and used when the pixmap is insensitive. The default value is TRUE.

`pixmap` : a GtkPixmap.

build : set to TRUE if an extra pixmap should be automatically created to use when the pixmap is insensitive.

<< GtkOptionMenu **GtkPreview >>**

GtkPreview

GtkPreview — A widget to display RGB or grayscale data

Synopsis

```
#include <gtk/gtk.h>

            GtkPreview;
            GtkPreviewInfo;
union       GtkDitherInfo;
void        gtk_preview_uninit              (void);
GtkWidget*  gtk_preview_new                 (GtkPreviewType type);
void        gtk_preview_size                (GtkPreview *preview,
                                             gint width,
                                             gint height);
void        gtk_preview_put                 (GtkPreview *preview,
                                             GdkWindow *window,
                                             GdkGC *gc,
                                             gint srcx,
                                             gint srcy,
                                             gint destx,
                                             gint desty,
                                             gint width,
                                             gint height);
void        gtk_preview_draw_row            (GtkPreview *preview,
                                             guchar *data,
                                             gint x,
                                             gint y,
                                             gint w);
void        gtk_preview_set_expand          (GtkPreview *preview,
```

```
                                                gboolean expand);
void        gtk_preview_set_gamma               (double gamma_);
void        gtk_preview_set_color_cube          (guint nred_shades,
                                                 guint ngreen_shades,
                                                 guint nblue_shades,
                                                 guint ngray_shades);
void        gtk_preview_set_install_cmap        (gint install_cmap);
void        gtk_preview_set_reserved            (gint nreserved);
void        gtk_preview_set_dither              (GtkPreview *preview,
                                                 GdkRgbDither dither);
GdkVisual*  gtk_preview_get_visual              (void);
GdkColormap* gtk_preview_get_cmap               (void);
GtkPreviewInfo* gtk_preview_get_info            (void);
void        gtk_preview_reset                   (void);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkWidget
               +----GtkPreview
```

Implemented Interfaces

GtkPreview implements AtkImplementorIface.

Properties

```
  "expand"                gboolean              : Read / Write
```

Description

The GtkPreview widget provides a simple interface used to display images as RGB or grayscale data. It's deprecated; just use a GdkPixbuf displayed by a GtkImage, or perhaps a GtkDrawingArea. GtkPreview has no advantage over those approaches.

Details

GtkPreview

```
typedef struct _GtkPreview GtkPreview;
```

Warning

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

The GtkPreview-struct struct contains private data only, and should be accessed using the functions below.

GtkPreviewInfo

```
typedef struct {
  guchar *lookup;

  gdouble gamma;
} GtkPreviewInfo;
```

Warning

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

Contains information about global properties of preview widgets. The GtkPreviewInfo struct contains the following fields. (These fields should be considered read-only. They should never be set by an application.)

GdkVisual *visual;	the visual used by all previews.
GdkColormap *cmap;	the colormap used by all previews.
gdouble gamma;	the gamma correction value used by all previews (See `gtk_preview_set_gamma()`).

---

union GtkDitherInfo

```
union GtkDitherInfo
{
  gushort s[2];
  guchar c[4];
};
```

Warning

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

This union not used in GTK+.

---

gtk_preview_uninit ()

```
void        gtk_preview_uninit              (void);
```

Warning

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

This function is deprecated and does nothing.

---

gtk_preview_new ()

```
GtkWidget*  gtk_preview_new                 (GtkPreviewType type);
```

Warning

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

Create a new preview widget.

`type` : the type data contained by the widget. (Grayscale or RGB)

Returns :

---

gtk_preview_size ()

```
void        gtk_preview_size                (GtkPreview *preview,
                                             gint width,
                                             gint height);
```

Warning

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

Set the size that the preview widget will request in response to a "size_request" signal. The drawing area may actually be allocated a size larger than this depending on how it is packed within the enclosing containers. The effect of this is determined by whether the preview is set to expand or not (see `gtk_preview_expand()`)

`preview` : a GtkPreview.

`width` : the new width.

`height` : the new height.

---

gtk_preview_put ()

```
void        gtk_preview_put                 (GtkPreview *preview,
                                             GdkWindow *window,
                                             GdkGC *gc,
                                             gint srcx,
                                             gint srcy,
                                             gint destx,
                                             gint desty,
                                             gint width,
                                             gint height);
```

Warning

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

Takes a portion of the contents of a preview widget and draws it onto the given drawable, *`window`*.

`preview` :	a GtkPreview.
`window` :	a window or pixmap.
`gc` :	The graphics context for the operation. Only the clip mask for this GC matters.
`srcx` :	the x coordinate of the upper left corner in the source image.
`srcy` :	the y coordinate of the upper left corner in the source image.
`destx` :	the x coordinate of the upper left corner in the destination image.
`desty` :	the y coordinate of the upper left corner in the destination image.
`width` :	the width of the rectangular portion to draw.
`height` :	the height of the rectangular portion to draw.

---

gtk_preview_draw_row ()

```
void        gtk_preview_draw_row            (GtkPreview *preview,
                                             guchar *data,
                                             gint x,
                                             gint y,
                                             gint w);
```

Warning

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

Sets the data for a portion of a row.

preview : a GtkPreview.

data : the new data for the portion. It should contain *w* bytes of data if the preview is of type GTK_TYPE_GRAYSCALE, and 3**w* bytes of data if the preview is of type GTK_TYPE_COLOR.

x : the starting value on the row to set.

y : the row to change.

w : the number of pixels in the row to change.

---

gtk_preview_set_expand ()

```
void        gtk_preview_set_expand          (GtkPreview *preview,
                                             gboolean expand);
```

Warning

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

Determines the way that the the preview widget behaves when the size it is allocated is larger than the requested size. If *expand* is FALSE, then the preview's window and buffer will be no larger than the size set with `gtk_preview_size()`, and the data set will be centered in the allocation if it is larger. If *expand* is TRUE then the window and buffer will expand with the allocation; the application is responsible for catching the "size_allocate" signal and providing the data appropriate for this size.

preview : a GtkPreview.

expand : whether the preview's window should expand or not.

---

gtk_preview_set_gamma ()

```
void        gtk_preview_set_gamma           (double gamma_);
```

Warning

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

Set the gamma-correction value for all preview widgets. (This function will eventually be replaced with a function that sets a per-preview-widget gamma value). The resulting intensity is given by: `destination_value * pow (source_value/255, 1/gamma)`. The gamma value is applied when the data is set with `gtk_preview_draw_row()` so changing this value will not affect existing data in preview widgets.

*gamma_* : the new gamma value.

---

gtk_preview_set_color_cube ()

```
void        gtk_preview_set_color_cube      (guint nred_shades,
                                             guint ngreen_shades,
                                             guint nblue_shades,
                                             guint ngray_shades);
```

Warning

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

This function is deprecated and does nothing. GdkRGB automatically picks an optimium color cube for the display.

nred_shades : ignored
ngreen_shades : ignored
nblue_shades : ignored
ngray_shades : ignored

---

gtk_preview_set_install_cmap ()

```
void        gtk_preview_set_install_cmap    (gint install_cmap);
```

Warning

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

This function is deprecated and does nothing. GdkRGB will automatically pick a private colormap if it cannot allocate sufficient colors.

`install_cmap` : ignored.

---

gtk_preview_set_reserved ()

```
void        gtk_preview_set_reserved        (gint nreserved);
```

Warning

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

This function is deprecated and does nothing.

`nreserved` : ignored.

---

gtk_preview_set_dither ()

```
void        gtk_preview_set_dither          (GtkPreview *preview,
                                             GdkRgbDither dither);
```

Warning

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

Set the dithering mode for the display.

`preview` : a GtkPreview.

`dither` : the dithering mode.

---

gtk_preview_get_visual ()

```
GdkVisual*  gtk_preview_get_visual          (void);
```

Warning

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

Returns the visual used by preview widgets. This function is deprecated, and you should use `gdk_rgb_get_visual()` instead.

Returns : the visual for previews.

---

gtk_preview_get_cmap ()

```
GdkColormap* gtk_preview_get_cmap           (void);
```

Warning

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

Returns the colormap used by preview widgets. This function is deprecated, and you should use `gdk_rgb_get_cmap()` instead.

Returns : the colormap for previews.

---

gtk_preview_get_info ()

```
GtkPreviewInfo* gtk_preview_get_info        (void);
```

Warning

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

Return a GtkPreviewInfo structure containing global information about preview widgets.

Returns : a GtkPreviewInfo structure. The return value belongs to GTK+ and must not be modified or freed.

---

gtk_preview_reset ()

```
void        gtk_preview_reset               (void);
```

Warning

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

This function is deprecated and does nothing. It was once used for changing the colormap and visual on the fly.

Properties

The "expand" property

"expand"	gboolean	: Read / Write

Whether the preview widget should take up the entire space it is allocated.

Default value: FALSE

See Also

GdkRGB the backend used by GtkPreview.

<< GtkPixmap **GtkProgress >>**

GtkProgress

GtkProgress — Base class for GtkProgressBar

Synopsis

```
#include <gtk/gtk.h>

            GtkProgress;
void        gtk_progress_set_show_text      (GtkProgress *progress,
                                             gboolean show_text);
void        gtk_progress_set_text_alignment (GtkProgress *progress,
                                             gfloat x_align,
                                             gfloat y_align);
void        gtk_progress_set_format_string  (GtkProgress *progress,
                                             const gchar *format);
void        gtk_progress_set_adjustment     (GtkProgress *progress,
                                             GtkAdjustment *adjustment);
void        gtk_progress_set_percentage     (GtkProgress *progress,
                                             gdouble percentage);
void        gtk_progress_set_value          (GtkProgress *progress,
                                             gdouble value);
gdouble     gtk_progress_get_value          (GtkProgress *progress);
void        gtk_progress_set_activity_mode  (GtkProgress *progress,
                                             gboolean activity_mode);
gchar*      gtk_progress_get_current_text   (GtkProgress *progress);
gchar*      gtk_progress_get_text_from_value
                                            (GtkProgress *progress,
                                             gdouble value);
gdouble     gtk_progress_get_current_percentage
                                            (GtkProgress *progress);
gdouble     gtk_progress_get_percentage_from_value
                                            (GtkProgress *progress,
```

```
                                                gdouble value);
void        gtk_progress_configure             (GtkProgress *progress,
                                                gdouble value,
                                                gdouble min,
                                                gdouble max);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkWidget
               +----GtkProgress
                     +----GtkProgressBar
```

Implemented Interfaces

GtkProgress implements AtkImplementorIface.

Properties

```
  "activity-mode"         gboolean              : Read / Write
  "show-text"             gboolean              : Read / Write
  "text-xalign"           gfloat                : Read / Write
  "text-yalign"           gfloat                : Read / Write
```

Description

A GtkProgress is the abstract base class used to derive a GtkProgressBar which provides a visual representation of the progress of a long running operation.

Details

GtkProgress

```
typedef struct _GtkProgress GtkProgress;
```

The GtkProgress-struct struct contains private data only. and should be accessed using the functions below.

---

gtk_progress_set_show_text ()

```
void        gtk_progress_set_show_text      (GtkProgress *progress,
                                             gboolean show_text);
```

Warning

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

Controls whether progress text is shown.

progress : a GtkProgress.

show_text : a boolean indicating whether the progress text is shown.

---

gtk_progress_set_text_alignment ()

```
void        gtk_progress_set_text_alignment (GtkProgress *progress,
                                             gfloat x_align,
                                             gfloat y_align);
```

Warning

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

Controls the alignment of the text within the progress bar area.

progress : a GtkProgress.

x_align : a number between 0.0 and 1.0 indicating the horizontal alignment of the progress text within the GtkProgress.

y_align : a number between 0.0 and 1.0 indicating the vertical alignment of the progress text within the GtkProgress.

---

gtk_progress_set_format_string ()

```
void        gtk_progress_set_format_string  (GtkProgress *progress,
                                             const gchar *format);
```

Warning

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

Sets a format string used to display text indicating the current progress. The string can contain the following substitution characters:

- %v - the current progress value.
- %l - the lower bound for the progress value.
- %u - the upper bound for the progress value.
- %p - the current progress percentage.

progress : a GtkProgress.

format : a string used to display progress text, or `NULL` to restore to the default format.

---

gtk_progress_set_adjustment ()

```
void        gtk_progress_set_adjustment     (GtkProgress *progress,
                                             GtkAdjustment *adjustment);
```

Warning

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

Associates a GtkAdjustment with the GtkProgress. A GtkAdjustment is used to represent the upper and lower bounds and the step interval of the underlying value for which progress is shown.

`progress` : a GtkProgress.
`adjustment` : the GtkAdjustment to be associated with the GtkProgress.

---

gtk_progress_set_percentage ()

```
void        gtk_progress_set_percentage     (GtkProgress *progress,
                                             gdouble percentage);
```

Warning

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

Sets the current percentage completion for the GtkProgress.

`progress` : a GtkProgress.
`percentage` : the percentage complete which must be between 0.0 and 1.0.

---

gtk_progress_set_value ()

```
void        gtk_progress_set_value          (GtkProgress *progress,
                                             gdouble value);
```

Warning

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

Sets the value within the GtkProgress to an absolute value. The value must be within the valid range of values

for the underlying GtkAdjustment.

progress : a GtkProgress.
value : the value indicating the current completed amount.

---

gtk_progress_get_value ()

```
gdouble     gtk_progress_get_value          (GtkProgress *progress);
```

Warning

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

Returns the current progress complete value.

progress : a GtkProgress.
Returns : the current progress complete value.

---

gtk_progress_set_activity_mode ()

```
void        gtk_progress_set_activity_mode  (GtkProgress *progress,
                                             gboolean activity_mode);
```

Warning

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

A GtkProgress can be in one of two different modes: percentage mode (the default) and activity mode. In activity mode, the progress is simply indicated as activity rather than as a percentage complete.

progress : a GtkProgress.
activity_mode : a boolean, TRUE for activity mode.

---

gtk_progress_get_current_text ()

```
gchar*      gtk_progress_get_current_text   (GtkProgress *progress);
```

Warning

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

Returns the current text associated with the GtkProgress. This text is the based on the underlying format string after any substitutions are made.

progress : a GtkProgress.

Returns : the text indicating the current progress.

---

gtk_progress_get_text_from_value ()

```
gchar*      gtk_progress_get_text_from_value
                                        (GtkProgress *progress,
                                         gdouble value);
```

Warning

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

Returns the text indicating the progress based on the supplied value. The current value for the GtkProgress remains unchanged.

progress : a GtkProgress.

value : an absolute progress value to use when formatting the progress text.

Returns : a string indicating the progress.

---

gtk_progress_get_current_percentage ()

```
gdouble     gtk_progress_get_current_percentage
                                            (GtkProgress *progress);
```

Warning

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

Returns the current progress as a percentage.

progress : a GtkProgress.

Returns : a number between 0.0 and 1.0 indicating the percentage complete.

---

gtk_progress_get_percentage_from_value ()

```
gdouble     gtk_progress_get_percentage_from_value
                                            (GtkProgress *progress,
                                             gdouble value);
```

Warning

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

Returns the progress as a percentage calculated from the supplied absolute progress value.

progress : a GtkProgress.

value : an absolute progress value.

Returns : a number between 0.0 and 1.0 indicating the percentage complete represented by *value*.

---

gtk_progress_configure ()

```
void        gtk_progress_configure          (GtkProgress *progress,
                                             gdouble value,
                                             gdouble min,
                                             gdouble max);
```

Warning

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

Allows the configuration of the minimum, maximum, and current values for the GtkProgress.

progress : a GtkProgress.
value : the current progress value.
min : the minimum progress value.
max : the maximum progress value.

Properties

The "activity-mode" property

```
  "activity-mode"        gboolean              : Read / Write
```

If TRUE the GtkProgress is in activity mode, meaning that it signals something is happening, but not how much of the activity is finished. This is used when you're doing something that you don't know how long it will take.

Default value: FALSE

The "show-text" property

```
  "show-text"            gboolean              : Read / Write
```

Whether the progress is shown as text.

Default value: FALSE

The "text-xalign" property

```
"text-xalign"          gfloat                : Read / Write
```

A number between 0.0 and 1.0 specifying the horizontal alignment of the text in the progress widget.

Allowed values: [0,1]

Default value: 0.5

The "text-yalign" property

```
"text-yalign"          gfloat                : Read / Write
```

A number between 0.0 and 1.0 specifying the vertical alignment of the text in the progress widget.

Allowed values: [0,1]

Default value: 0.5

<< GtkPreview **GtkText >>**

GtkText

GtkText — A text widget

Synopsis

```
#include <gtk/gtk.h>

            GtkText;
            GtkTextFont;
            GtkPropertyMark;
GtkWidget*  gtk_text_new                    (GtkAdjustment *hadj,
                                             GtkAdjustment *vadj);
void        gtk_text_set_editable           (GtkText *text,
                                             gboolean editable);
void        gtk_text_set_word_wrap          (GtkText *text,
                                             gboolean word_wrap);
void        gtk_text_set_line_wrap          (GtkText *text,
                                             gboolean line_wrap);
void        gtk_text_set_adjustments        (GtkText *text,
                                             GtkAdjustment *hadj,
                                             GtkAdjustment *vadj);
void        gtk_text_set_point              (GtkText *text,
                                             guint index);
guint       gtk_text_get_point              (GtkText *text);
guint       gtk_text_get_length             (GtkText *text);
void        gtk_text_freeze                 (GtkText *text);
void        gtk_text_thaw                   (GtkText *text);
void        gtk_text_insert                 (GtkText *text,
                                             GdkFont *font,
                                             const GdkColor *fore,
```

```
                                                const GdkColor *back,
                                                const char *chars,
                                                gint length);
gboolean    gtk_text_backward_delete        (GtkText *text,
                                                guint nchars);
gboolean    gtk_text_forward_delete         (GtkText *text,
                                                guint nchars);
#define     GTK_TEXT_INDEX                  (t, index)
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkWidget
               +----GtkOldEditable
                     +----GtkText
```

Implemented Interfaces

GtkText implements AtkImplementorIface and GtkEditable.

Properties

```
  "hadjustment"          GtkAdjustment         : Read / Write
  "line-wrap"            gboolean              : Read / Write
  "vadjustment"          GtkAdjustment         : Read / Write
  "word-wrap"            gboolean              : Read / Write
```

Signal Prototypes

```
"set-scroll-adjustments"
            void        user_function      (GtkText *text,
                                            GtkAdjustment *arg1,
                                            GtkAdjustment *arg2,
                                            gpointer user_data);
```

Description

Warning

GtkText is deprecated and unsupported. It is known to be buggy. To use it, you must define the symbol GTK_ENABLE_BROKEN prior to including the GTK+ header files. Use GtkTextView instead.

A GtkText widget allows one to display any given text and manipulate it by deleting from one point to another, selecting a region, and various other functions as outlined below. It is inherited from GtkEditable.

Details

GtkText

```
typedef struct _GtkText GtkText;
```

Warning

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

Most of the GtkText-struct struct members should not be accessed directly. Listed below are a few exceptions and how to use them.

GtkTextFont

```
typedef struct _GtkTextFont GtkTextFont;
```

Warning

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

Internal GtkText data type.

GtkPropertyMark

```
typedef struct {
  /* Position in list. */
  GList* property;

  /* Offset into that property. */
  guint offset;

  /* Current index. */
  guint index;
} GtkPropertyMark;
```

Warning

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

Internal GtkText data type. Should not be accessed directly.

---

gtk_text_new ()

```
GtkWidget*  gtk_text_new                            (GtkAdjustment *hadj,
                                                     GtkAdjustment *vadj);
```

Warning

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

Creates a new GtkText widget, initialized with the given pointers to GtkAdjustments. These pointers can be used to track the viewing position of the GtkText widget. Passing `NULL` to either or both of them will make the GtkText create its own. You can set these later with the function `gtk_text_set_adjustment()`.

hadj : horizontal adjustment.
vadj : vertical adjustment.
Returns : the new GtkText widget.

---

gtk_text_set_editable ()

```
void        gtk_text_set_editable           (GtkText *text,
                                             gboolean editable);
```

Warning

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

Sets whether the GtkText widget can be edited by the user or not. This still allows you the programmer to make changes with the various GtkText functions.

text : the GtkText widget
editable : TRUE makes it editable, FALSE makes it immutable by the user

---

gtk_text_set_word_wrap ()

```
void        gtk_text_set_word_wrap          (GtkText *text,
                                             gboolean word_wrap);
```

Warning

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

Sets whether the GtkText widget wraps words down to the next line if it can't be completed on the current line.

`text` : the GtkText widget

`word_wrap` : TRUE makes it word wrap, FALSE disables word wrapping

---

gtk_text_set_line_wrap ()

```
void        gtk_text_set_line_wrap          (GtkText *text,
                                             gboolean line_wrap);
```

Warning

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

Controls how GtkText handles long lines of continuous text. If line wrap is on, the line is broken when it reaches the extent of the GtkText widget viewing area and the rest is displayed on the next line. If it is not set, the line continues regardless size of current viewing area. Similar to word wrap but it disregards word boundaries.

`text` : the GtkText widget

`line_wrap` : TRUE turns line wrap on, FALSE turns it off

---

gtk_text_set_adjustments ()

```
void        gtk_text_set_adjustments        (GtkText *text,
                                             GtkAdjustment *hadj,
```

```
                                                GtkAdjustment *vadj);
```

Warning

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

Allows you to set GtkAdjustment pointers which in turn allows you to keep track of the viewing position of the GtkText widget.

`text` : the GtkText widget

`hadj` : the horizontal adjustment

`vadj` : the vertical adjustment

---

gtk_text_set_point ()

```
void        gtk_text_set_point              (GtkText *text,
                                             guint index);
```

Warning

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

Sets the cursor at the given point. In this case a point constitutes the number of characters from the extreme upper left corner of the GtkText widget.

`text` : the GtkText widget

`index` : the number of characters from the upper left corner

---

gtk_text_get_point ()

```
guint       gtk_text_get_point              (GtkText *text);
```

Warning

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

Gets the current position of the cursor as the number of characters from the upper left corner of the GtkText widget.

`text` : the GtkText widget

Returns : the number of characters from the upper left corner

---

gtk_text_get_length ()

```
guint       gtk_text_get_length             (GtkText *text);
```

Warning

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

Returns the length of the all the text contained within the GtkText widget; disregards current point position.

`text` : the GtkText widget

Returns : the length of the text

---

gtk_text_freeze ()

```
void        gtk_text_freeze                 (GtkText *text);
```

Warning

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

Freezes the GtkText widget which disallows redrawing of the widget until it is thawed. This is useful if a large number of changes are going to made to the text within the widget, reducing the amount of flicker seen by the user.

text : the GtkText widget

---

gtk_text_thaw ()

```
void        gtk_text_thaw                       (GtkText *text);
```

Warning

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

Allows the GtkText widget to be redrawn again by GTK.

text : the GtkText widget

---

gtk_text_insert ()

```
void        gtk_text_insert                     (GtkText *text,
                                                 GdkFont *font,
                                                 const GdkColor *fore,
                                                 const GdkColor *back,
                                                 const char *chars,
                                                 gint length);
```

Warning

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

Inserts given text into the GtkText widget with the given properties as outlined below.

text : the GtkText widget
font : the GdkFont to use
fore : the foreground color to insert with
back : the background color to insert with
chars : the actual text to be inserted
length : the length of the text to be inserted, passing -1 makes it insert all the text.

---

gtk_text_backward_delete ()

```
gboolean    gtk_text_backward_delete        (GtkText *text,
                                             guint nchars);
```

Warning

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

Deletes from the current point position backward the given number of characters.

text : the GtkText widget
nchars : the number of characters to delete
Returns : TRUE if the operation was successful, otherwise returns FALSE

---

gtk_text_forward_delete ()

```
gboolean    gtk_text_forward_delete         (GtkText *text,
                                             guint nchars);
```

Warning

`gtk_text_forward_delete` is deprecated and should not be used in newly-written

code.

Deletes from the current point position forward the given number of characters.

text : the GtkText widget
nchars : the number of characters to delete
Returns : TRUE if the operation was successful, otherwise returns FALSE

GTK_TEXT_INDEX()

```
#define     GTK_TEXT_INDEX(t, index)
```

Warning

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

Returns the character at the given index within the GtkText widget.

t : the GtkText widget
index : the number of characters from the upper left corner

Properties

The "hadjustment" property

```
  "hadjustment"          GtkAdjustment         : Read / Write
```

Used by the GtkText widget to keep track of the size of its horizontal text.

The "line-wrap" property

```
  "line-wrap"           gboolean              : Read / Write
```

Boolean value indicating whether line wrap is enabled or not.

Default value: TRUE

The "vadjustment" property

```
  "vadjustment"         GtkAdjustment         : Read / Write
```

Used by the GtkText widget to keep track of the size of its vertical text.

The "word-wrap" property

```
  "word-wrap"           gboolean              : Read / Write
```

Boolean value indicated whether word wrap is enabled or not.

Default value: FALSE

Signals

The "set-scroll-adjustments" signal

```
void        user_function                     (GtkText *text,
                                               GtkAdjustment *arg1,
                                               GtkAdjustment *arg2,
                                               gpointer user_data);
```

text : the object which received the signal.

arg1 :

arg2 :

user_data : user data set when the signal handler was connected.

<< GtkProgress **GtkTipsQuery >>**

GtkTipsQuery

GtkTipsQuery — Displays help about widgets in the user interface

Synopsis

```
#include <gtk/gtk.h>


            GtkTipsQuery;
GtkWidget*  gtk_tips_query_new              (void);
void        gtk_tips_query_start_query      (GtkTipsQuery *tips_query);
void        gtk_tips_query_stop_query       (GtkTipsQuery *tips_query);
void        gtk_tips_query_set_caller       (GtkTipsQuery *tips_query,
                                             GtkWidget *caller);
void        gtk_tips_query_set_labels       (GtkTipsQuery *tips_query,
                                             const gchar *label_inactive,
                                             const gchar *label_no_tip);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkWidget
               +----GtkMisc
                     +----GtkLabel
                           +----GtkTipsQuery
```

Implemented Interfaces

GtkTipsQuery implements AtkImplementorIface.

Properties

```
  "caller"              GtkWidget             : Read / Write
  "emit-always"         gboolean              : Read / Write
  "label-inactive"      gchararray            : Read / Write
  "label-no-tip"        gchararray            : Read / Write
```

Signal Prototypes

```
"start-query"
            void        user_function      (GtkTipsQuery *tipsquery,
                                            gpointer user_data);
"stop-query"
            void        user_function      (GtkTipsQuery *tipsquery,
                                            gpointer user_data);
"widget-entered"
            void        user_function      (GtkTipsQuery *tipsquery,
                                            GtkWidget *widget,
                                            gchar *tip_text,
                                            gchar *tip_private,
                                            gpointer user_data);
"widget-selected"
            gboolean    user_function      (GtkTipsQuery *tipsquery,
                                            GtkWidget *widget,
                                            gchar *tip_text,
                                            gchar *tip_private,
                                            GdkEventButton *event,
                                            gpointer user_data);
```

Description

The GtkTipsQuery widget is a subclass of GtkLabel which is used to display help about widgets in a user interface.

A query is started with a call to `gtk_tips_query_start_query()`, usually when some kind of 'Help' button is pressed. The GtkTipsQuery then grabs all events, stopping the user interface from functioning normally. Then as the user moves the mouse over the widgets, the GtkTipsQuery displays each widget's tooltip text.

By connecting to the "widget-entered" or "widget-selected" signals, it is possible to customize the GtkTipsQuery to perform other actions when widgets are entered or selected. For example, a help browser could be opened with documentation on the widget selected.

At some point a call to `gtk_tips_query_stop_query()` must be made in order to stop the query and return the interface to its normal state. The `gtk_tips_query_set_caller()` function can be used to specify a widget which the user can select to stop the query (often the same button used to start the query).

Details

GtkTipsQuery

```
typedef struct _GtkTipsQuery GtkTipsQuery;
```

Warning

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

The GtkTipsQuery-struct struct contains private data only, and should be accessed using the functions below.

---

gtk_tips_query_new ()

```
GtkWidget*  gtk_tips_query_new              (void);
```

Warning

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

Creates a new GtkTipsQuery.

Returns : a new GtkTipsQuery.

---

gtk_tips_query_start_query ()

```
void        gtk_tips_query_start_query      (GtkTipsQuery *tips_query);
```

Warning

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

Starts a query. The GtkTipsQuery widget will take control of the mouse and as the mouse moves it will display the tooltip of the widget beneath the mouse.

`tips_query` : a GtkTipsQuery.

---

gtk_tips_query_stop_query ()

```
void        gtk_tips_query_stop_query       (GtkTipsQuery *tips_query);
```

Warning

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

Stops a query.

`tips_query` : a GtkTipsQuery.

---

gtk_tips_query_set_caller ()

```
void        gtk_tips_query_set_caller       (GtkTipsQuery *tips_query,
                                             GtkWidget *caller);
```

Warning

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

Sets the widget which initiates the query, usually a button. If the *`caller`* is selected while the query is running, the query is automatically stopped.

`tips_query` : a GtkTipsQuery.
`caller` : the widget which initiates the query.

---

gtk_tips_query_set_labels ()

```
void        gtk_tips_query_set_labels       (GtkTipsQuery *tips_query,
                                             const gchar *label_inactive,
                                             const gchar *label_no_tip);
```

Warning

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

Sets the text to display when the query is not in effect, and the text to display when the query is in effect but the widget beneath the pointer has no tooltip.

`tips_query` : a GtkTipsQuery.
`label_inactive` : the text to display when the query is not running.
`label_no_tip` : the text to display when the query is running but the widget beneath the pointer has no tooltip.

Properties

The "caller" property

```
  "caller"                GtkWidget             : Read / Write
```

The widget that starts the tips query, usually a button. If it is selected while the query is in effect the query is automatically stopped.

The "emit-always" property

```
"emit-always"          gboolean              : Read / Write
```

TRUE if the widget-entered and widget-selected signals are emitted even when the widget has no tooltip set.

Default value: FALSE

The "label-inactive" property

```
"label-inactive"       gchararray            : Read / Write
```

The text to display in the GtkTipsQuery widget when the query is not in effect.

Default value: NULL

The "label-no-tip" property

```
"label-no-tip"         gchararray            : Read / Write
```

The text to display in the GtkTipsQuery widget when the query is running and the widget that the pointer is over has no tooltip.

Default value: NULL

Signals

The "start-query" signal

```
void        user_function                   (GtkTipsQuery *tipsquery,
                                             gpointer user_data);
```

Emitted when the query is started.

tipsquery : the object which received the signal.
user_data : user data set when the signal handler was connected.

The "stop-query" signal

```
void        user_function                   (GtkTipsQuery *tipsquery,
                                             gpointer user_data);
```

Emitted when the query is stopped.

tipsquery : the object which received the signal.
user_data : user data set when the signal handler was connected.

The "widget-entered" signal

```
void        user_function                   (GtkTipsQuery *tipsquery,
                                             GtkWidget *widget,
                                             gchar *tip_text,
                                             gchar *tip_private,
                                             gpointer user_data);
```

Emitted when a widget is entered by the pointer while the query is in effect.

tipsquery : the object which received the signal.
widget : the widget that was entered by the pointer.
tip_text : the widget's tooltip.
tip_private : the widget's private tooltip (see gtk_tooltips_set_tip()).

user_data : user data set when the signal handler was connected.

The "widget-selected" signal

```
gboolean    user_function                   (GtkTipsQuery *tipsquery,
                                             GtkWidget *widget,
                                             gchar *tip_text,
                                             gchar *tip_private,
                                             GdkEventButton *event,
                                             gpointer user_data);
```

Emitted when a widget is selected during a query.

tipsquery : the object which received the signal.
widget : the widget that was selected.
tip_text : the widget's tooltip.
tip_private : the widget's private tooltip (see gtk_tooltips_set_tip()).
event : the button press or button release event.
user_data : user data set when the signal handler was connected.
Returns : TRUE if the query should be stopped.

See Also

GtkTooltips the object which handles tooltips.

<< GtkText **GtkTree >>**

GtkTree

GtkTree — A tree widget

Synopsis

```
#include <gtk/gtk.h>


            GtkTree;
#define     GTK_IS_ROOT_TREE                (obj)
#define     GTK_TREE_ROOT_TREE              (obj)
#define     GTK_TREE_SELECTION_OLD          (obj)
enum        GtkTreeViewMode;
GtkWidget*  gtk_tree_new                    (void);
void        gtk_tree_append                 (GtkTree *tree,
                                             GtkWidget *tree_item);
void        gtk_tree_prepend                (GtkTree *tree,
                                             GtkWidget *tree_item);
void        gtk_tree_insert                 (GtkTree *tree,
                                             GtkWidget *tree_item,
                                             gint position);
void        gtk_tree_remove_items           (GtkTree *tree,
                                             GList *items);
void        gtk_tree_clear_items            (GtkTree *tree,
                                             gint start,
                                             gint end);
void        gtk_tree_select_item            (GtkTree *tree,
                                             gint item);
void        gtk_tree_unselect_item          (GtkTree *tree,
                                             gint item);
void        gtk_tree_select_child           (GtkTree *tree,
                                             GtkWidget *tree_item);
void        gtk_tree_unselect_child         (GtkTree *tree,
                                             GtkWidget *tree_item);
gint        gtk_tree_child_position         (GtkTree *tree,
                                             GtkWidget *child);
void        gtk_tree_set_selection_mode     (GtkTree *tree,
                                             GtkSelectionMode mode);
void        gtk_tree_set_view_mode          (GtkTree *tree,
                                             GtkTreeViewMode mode);
```

```
void        gtk_tree_set_view_lines         (GtkTree *tree,
                                             gboolean flag);
void        gtk_tree_remove_item            (GtkTree *tree,
                                             GtkWidget *child);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkWidget
               +----GtkContainer
                     +----GtkTree
```

Implemented Interfaces

GtkTree implements AtkImplementorIface.

Signal Prototypes

```
"select-child"
            void        user_function      (GtkTree *tree,
                                            GtkWidget *widget,
                                            gpointer user_data);
"selection-changed"
            void        user_function      (GtkTree *tree,
                                            gpointer user_data);
"unselect-child"
            void        user_function      (GtkTree *tree,
                                            GtkWidget *widget,
                                            gpointer user_data);
```

Description

Warning

GtkTree is deprecated and unsupported. It is known to be buggy. To use it, you must define the symbol `GTK_ENABLE_BROKEN` prior to including the GTK+ header files. Use GtkTreeView instead.

The GtkTree widget is a container that shows users a list of items, in a tree format complete with branches and leafnodes. Branches can be expanded to show their child items, or collapsed to hide them.

Details

GtkTree

```
typedef struct _GtkTree GtkTree;
```

Warning

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

```
struct _GtkTree
{
  GtkContainer container;
  GList *children;
  GtkTree* root_tree; /* owner of selection list */
  GtkWidget* tree_owner;
  GList *selection;
  guint level;
  guint indent_value;
  guint current_indent;
  guint selection_mode : 2;
  guint view_mode : 1;
  guint view_line : 1;
};
```

GTK_IS_ROOT_TREE()

```
#define GTK_IS_ROOT_TREE(obj)   ((GtkObject*) GTK_TREE(obj)->root_tree ==
(GtkObject*)obj)
```

Warning

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

A macro that returns a boolean value which indicates if *`obj`* is a root tree or not.

`obj` : A pointer to the GtkTree. *`obj`* will accept any pointer, but if the pointer does not point to a GtkTree, the results are undefined.

GTK_TREE_ROOT_TREE()

```
#define GTK_TREE_ROOT_TREE(obj) (GTK_TREE(obj)->root_tree ? GTK_TREE(obj)->root_tree
: GTK_TREE(obj))
```

Warning

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

A macro that returns the root tree of *`obj`*.

If *`obj`* is already a root tree, *`obj`* is cast to GtkTree and returned.

`obj` : A pointer to the GtkTree. *`obj`* will accept any pointer, but if the pointer does not point to a GtkTree, the results are undefined.

GTK_TREE_SELECTION_OLD()

```
#define GTK_TREE_SELECTION_OLD(obj) (GTK_TREE_ROOT_TREE(obj)->selection)
```

Warning

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

`obj` :

---

enum GtkTreeViewMode

```
typedef enum
{
  GTK_TREE_VIEW_LINE,  /* default view mode */
  GTK_TREE_VIEW_ITEM
} GtkTreeViewMode;
```

Warning

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

---

gtk_tree_new ()

```
GtkWidget*  gtk_tree_new                    (void);
```

Warning

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

Creates a new GtkTree.

Returns : A pointer to the newly allocated widget.

---

gtk_tree_append ()

```
void        gtk_tree_append                 (GtkTree *tree,
                                             GtkWidget *tree_item);
```

Warning

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

Adds the GtkTreeItem in *`tree_item`* to the end of the items in *`tree`*.

`tree` : A pointer to a GtkTree.

`tree_item` : A pointer to the GtkWidget that is to be appended to the tree.

---

gtk_tree_prepend ()

```
void        gtk_tree_prepend                (GtkTree *tree,
                                             GtkWidget *tree_item);
```

Warning

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

Adds the GtkTreeItem in *`tree_item`* to the start of the items in *`tree`*.

`tree` : A pointer to a GtkTree.

tree_item : A pointer to the GtkWidget that is to be prepended to the tree.

---

gtk_tree_insert ()

```
void        gtk_tree_insert                 (GtkTree *tree,
                                             GtkWidget *tree_item,
                                             gint position);
```

Warning

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

Adds the GtkTreeItem in *tree_item* to the list of items in *tree* at the position indicated by *position*.

tree : A pointer to a GtkTree.
tree_item : A pointer to the GtkWidget that is to be added to the tree.
position : A gint that indicates the position in the tree, that the *tree_item* is to be added at.

---

gtk_tree_remove_items ()

```
void        gtk_tree_remove_items           (GtkTree *tree,
                                             GList *items);
```

Warning

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

Removes a list of items from the GtkTree in *tree*.

If only one item is to be removed from the GtkTree, `gtk_container_remove()` can be used instead.

Removing an item from a GtkTree dereferences the item, and thus usually destroys the item and any subtrees it may contain. If the item is not to be destroyed, use `g_object_ref()` before removing it.

tree : A pointer to a GtkTree.
items : A pointer to a GList that contains the items to be removed.

---

gtk_tree_clear_items ()

```
void        gtk_tree_clear_items            (GtkTree *tree,
                                             gint start,
                                             gint end);
```

Warning

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

Removes the items at positions between *`start`* and *`end`* from the GtkTree *`tree`*.

Removing an item from a GtkTree dereferences the item, and thus usually destroys the item and any subtrees it may contain. If the item is not to be destroyed, use `g_object_ref()` before removing it.

`tree` : A pointer to a GtkTree.
`start` : A gint.
`end` : A gint.

---

gtk_tree_select_item ()

```
void        gtk_tree_select_item            (GtkTree *tree,
                                             gint item);
```

Warning

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

Emits the select_item signal for the child at position *`item`*, and thus selects it (unless it is unselected in a signal handler).

`tree` : A pointer to a GtkTree.
`item` : A gint.

---

gtk_tree_unselect_item ()

```
void        gtk_tree_unselect_item          (GtkTree *tree,
                                             gint item);
```

Warning

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

Emits the unselect_item for the child at position *`item`*, and thus unselects it.

`tree` : A pointer to a GtkTree.
`item` : A gint.

---

gtk_tree_select_child ()

```
void        gtk_tree_select_child           (GtkTree *tree,
                                             GtkWidget *tree_item);
```

Warning

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

Emits the select_item signal for the child *`tree_item`*, and thus selects it (unless it is unselected in a signal handler).

`tree` : A pointer to a GtkTree.
`tree_item` : A pointer to the GtkWidget that is to be selected.

---

gtk_tree_unselect_child ()

```
void        gtk_tree_unselect_child         (GtkTree *tree,
                                             GtkWidget *tree_item);
```

Warning

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

Emits the unselect_item signal for the child *`tree_item`*, and thus unselects it.

`tree` : A pointer to a GtkTree.
`tree_item` : A pointer to the GtkWidget that is to be selected.

---

gtk_tree_child_position ()

```
gint        gtk_tree_child_position         (GtkTree *tree,
```

```
                                                GtkWidget *child);
```

Warning

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

Returns the position of *`child`* in the GtkTree *`tree`*.

If *`child`* is not a child of *`tree`*, then -1 is returned.

`tree` : A pointer to a GtkTree.
`child` : A pointer to a GtkWidget.
Returns : A gint.

---

gtk_tree_set_selection_mode ()

```
void        gtk_tree_set_selection_mode     (GtkTree *tree,
                                             GtkSelectionMode mode);
```

Warning

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

Sets the selection mode for the GtkTree *`tree`*.

`mode` can be one of

- `GTK_SELECTION_SINGLE` for when only one item can be selected at a time.
- `GTK_SELECTION_BROWSE` for when one item must be selected.
- `GTK_SELECTION_MULTIPLE` for when many items can be selected at once.
- `GTK_SELECTION_EXTENDED` Reserved for later use.

The selection mode is only defined for a root tree, as the root tree "owns" the selection.

The default mode is `GTK_SELECTION_SINGLE`.

`tree` : A pointer to a GtkTree.
`mode` : A GtkSelectionMode.

---

gtk_tree_set_view_mode ()

```
void        gtk_tree_set_view_mode          (GtkTree *tree,
                                             GtkTreeViewMode mode);
```

Warning

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

Sets the 'viewmode' for the GtkTree in *`tree`*. The 'viewmode' defines how the tree looks when an item is selected.

`mode` can be one of:

- `GTK_TREE_VIEW_LINE` : When an item is selected the entire GtkTreeItem is highlighted.
- `GTK_TREE_VIEW_ITEM` : When an item is selected only the selected item's child widget is highlighted.

The default mode is `GTK_TREE_VIEW_LINE`.

`tree` : A pointer to a GtkTree.
`mode` : A GtkTreeViewMode.

---

gtk_tree_set_view_lines ()

```
void        gtk_tree_set_view_lines         (GtkTree *tree,
                                             gboolean flag);
```

Warning

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

Sets whether or not the connecting lines between branches and children are drawn.

`tree` : A pointer to a GtkTree.
`flag` : A guint, indicating `TRUE`, or `FALSE`.

---

gtk_tree_remove_item ()

```
void        gtk_tree_remove_item            (GtkTree *tree,
                                             GtkWidget *child);
```

Warning

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

Removes the item *`child`* from the GtkTree *`tree`*.

`tree` : A pointer to a GtkTree.
`child` : A pointer to the GtkWidget that is to be removed from the tree.

Signals

The "select-child" signal

```
void        user_function                  (GtkTree *tree,
                                            GtkWidget *widget,
                                            gpointer user_data);
```

This signal is emitted by *`tree`* whenever *`widget`* is about to be selected.

`tree` : the object which received the signal.
`widget` : The child that is about to be selected.
`user_data` : user data set when the signal handler was connected.

The "selection-changed" signal

```
void        user_function                  (GtkTree *tree,
                                            gpointer user_data);
```

This signal is emitted by the root tree whenever the selection changes.

`tree` : the object which received the signal.
`user_data` : user data set when the signal handler was connected.

The "unselect-child" signal

```
void        user_function                  (GtkTree *tree,
                                            GtkWidget *widget,
                                            gpointer user_data);
```

This signal is emitted by *`tree`* whenever *`widget`* is about to be unselected.

tree : the object which received the signal.
widget : The child that is about to be unselected.
user_data : user data set when the signal handler was connected.

See Also

GtkTreeList for the items to put into a GtkTree.

GtkScrolledWindow for details on how to scroll around a GtkTree.

<< GtkTipsQuery **GtkTreeItem >>**

GtkTreeItem

GtkTreeItem — The widget used for items in a GtkTree

Synopsis

```
#include <gtk/gtk.h>


            GtkTreeItem;
#define     GTK_TREE_ITEM_SUBTREE           (obj)
GtkWidget*  gtk_tree_item_new               (void);
GtkWidget*  gtk_tree_item_new_with_label    (const gchar *label);
void        gtk_tree_item_set_subtree       (GtkTreeItem *tree_item,
                                             GtkWidget *subtree);
void        gtk_tree_item_remove_subtree    (GtkTreeItem *tree_item);
void        gtk_tree_item_select            (GtkTreeItem *tree_item);
void        gtk_tree_item_deselect          (GtkTreeItem *tree_item);
void        gtk_tree_item_expand            (GtkTreeItem *tree_item);
void        gtk_tree_item_collapse          (GtkTreeItem *tree_item);
```

Object Hierarchy

```
  GObject
   +----GtkObject
         +----GtkWidget
               +----GtkContainer
                     +----GtkBin
```

```
                              +----GtkItem
                                    +----GtkTreeItem
```

Implemented Interfaces

GtkTreeItem implements AtkImplementorIface.

Signal Prototypes

```
"collapse"  void        user_function      (GtkTreeItem *treeitem,
                                            gpointer user_data);
"expand"    void        user_function      (GtkTreeItem *treeitem,
                                            gpointer user_data);
```

Description

Warning

GtkTree is deprecated and unsupported. It is known to be buggy. To use it, you must define the symbol `GTK_ENABLE_BROKEN` prior to including the GTK+ header files. Use GtkTreeView instead.

Details

GtkTreeItem

```
typedef struct _GtkTreeItem GtkTreeItem;
```

Warning

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

GTK_TREE_ITEM_SUBTREE()

```
#define GTK_TREE_ITEM_SUBTREE(obj)      (GTK_TREE_ITEM(obj)->subtree)
```

Warning

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

obj :

---

gtk_tree_item_new ()

```
GtkWidget*  gtk_tree_item_new                (void);
```

Warning

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

Returns :

---

gtk_tree_item_new_with_label ()

```
GtkWidget*  gtk_tree_item_new_with_label    (const gchar *label);
```

Warning

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

label :

Returns :

---

gtk_tree_item_set_subtree ()

```
void        gtk_tree_item_set_subtree       (GtkTreeItem *tree_item,
                                             GtkWidget *subtree);
```

Warning

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

`tree_item` :

`subtree` :

---

gtk_tree_item_remove_subtree ()

```
void        gtk_tree_item_remove_subtree    (GtkTreeItem *tree_item);
```

Warning

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

`tree_item` :

---

gtk_tree_item_select ()

```
void        gtk_tree_item_select            (GtkTreeItem *tree_item);
```

Warning

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

tree_item:

---

gtk_tree_item_deselect ()

```
void        gtk_tree_item_deselect          (GtkTreeItem *tree_item);
```

Warning

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

tree_item:

---

gtk_tree_item_expand ()

```
void        gtk_tree_item_expand            (GtkTreeItem *tree_item);
```

Warning

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

tree_item:

---

gtk_tree_item_collapse ()

```
void        gtk_tree_item_collapse          (GtkTreeItem *tree_item);
```

Warning

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

`tree_item`:

Signals

The "collapse" signal

```
void        user_function                   (GtkTreeItem *treeitem,
                                             gpointer user_data);
```

`treeitem`: the object which received the signal.
`user_data`: user data set when the signal handler was connected.

The "expand" signal

```
void        user_function                   (GtkTreeItem *treeitem,
                                             gpointer user_data);
```

`treeitem`: the object which received the signal.
`user_data`: user data set when the signal handler was connected.

<< GtkTree **Part IV. Migrating from Previous Versions of GTK+ >>**

Migrating from Previous Versions of GTK+

This part describes what you need to change in programs use older versions of GTK+ so that they can use the new features.

<< GtkTreeItem **Migration Checklist >>**

Migration Checklist

This chapter includes a checklist of things you need to do to ensure that your programs are good citizens in the GTK+ world. By paying attention to the points in the checklist, you ensure that many automatic features of GTK+ will work correctly in your program.

Implement GtkWidget::popup_menu

Why. By handling this signal, you let widgets have context-sensitive menus that can be invoked with the standard key bindings.

The GtkWidget::popup_menu signal instructs the widget for which it is emitted to create a context-sensitive popup menu. By default, the key binding mechanism is set to emit this signal when the **Shift-F10** or **Menu** keys are pressed while a widget has the focus. If a widget in your application shows a popup menu when you press a mouse button, you can make it work as well through the normal key binding mechanism in the following fahion:

1. Write a function to create and show a popup menu. This function needs to know the button number and the event's time to pass them to gtk_menu_popup(). You can implement such a function like this:

```
static void
do_popup_menu (GtkWidget *my_widget, GdkEventButton *event)
{
  GtkWidget *menu;
  int button, event_time;

  menu = gtk_menu_new ();
  g_signal_connect (menu, "deactivate",
                    G_CALLBACK (gtk_widget_destroy), NULL);

  /* ... add menu items ... */

  if (event)
    {
      button = event->button;
      event_time = event->time;
    }
  else
    {
      button = 0;
      event_time = gtk_get_current_event_time ();
    }
```

```
  gtk_menu_popup (GTK_MENU (popup), NULL, NULL, NULL, NULL,
                  button, event_time);
}
```

2. In your button_press handler, call this function when you need to pop up a menu:

```
static gboolean
my_widget_button_press_event_handler (GtkWidget *widget, GdkEventButton *event)
{
  /* Ignore double-clicks and triple-clicks */
  if (event->button == 3 && event->type == GDK_BUTTON_PRESS)
    {
      do_popup_menu (widget, event);
      return TRUE;
    }

  return FALSE;
}
```

3. Implement a handler for the popup_menu signal:

```
static gboolean
my_widget_popup_menu_handler (GtkWidget *widget)
{
  do_popup_menu (widget, NULL);
  return TRUE;
}
```

Note

If you do not pass a positioning function to gtk_menu_popup(), it will show the menu at the mouse position by default. This is what you usually want when the menu is shown as a result of pressing a mouse button. However, if you press the **Shift-F10** or **Menu** keys while the widget is focused, the mouse cursor may not be near the widget at all. In the example above, you may want to provide your own menu-positioning function in the case where the *event* is NULL. This function should compute the desired position for a menu when it is invoked through the keyboard. For example, GtkEntry(3) aligns the top edge of its popup menu with the bottom edge of the entry.

Note

For the standard key bindings to work, your widget must be able to take the keyboard focus. In general, widgets should be fully usable through the keyboard and not just the mouse. The very first step of this is to ensure that your widget turns on the GTK_CAN_FOCUS FLAG.

<< Part IV. Migrating from Previous Versions of GTK+ **Use GdkEventExpose.region >>**

Use GdkEventExpose.region

Why. The *region* field of GdkEventExpose allows you to redraw less than the traditional *GdkEventRegion.area*.

In GTK+ 1.x, the GdkEventExpose structure only had an *area* field to let you determine the region that you needed to redraw. In GTK+ 2.x, this field exists for compatibility and as a simple interface. However, there is also a *region* field which contains a fine-grained region. The *area* field is simply the bounding rectangle of the *region*.

Widgets that are very expensive to re-render, such as an image editor, may prefer to use the *GdkEventExpose.region* field to paint as little as possible. Widgets that just use a few drawing primitives, such as labels and buttons, may prefer to use the traditional *GdkEventExpose.area* field for simplicity.

Regions have an internal representation that is accessible as a list of rectangles. To turn the *GdkEventExpose.region* field into such a list, use gdk_region_get_rectangles():

```
static gboolean
my_widget_expose_event_handler (GtkWidget *widget, GdkEventExpose *event)
{
  GdkRectangle *rects;
  int n_rects;
  int i;

  gdk_region_get_rectangles (event->region, &rects, &n_rects);

  for (i = 0; i < n_rects; i++)
    {
      /* Repaint rectangle: (rects[i].x, rects[i].y),
       *                    (rects[i].width, rects[i].height)
       */
    }

  g_free (rects);

  return FALSE;
}
```

<< Migration Checklist **Test for modifier keys correctly >>**

Test for modifier keys correctly

Why. With `gtk_accelerator_get_default_mod_mask()` you can test for modifier keys reliably; this way your key event handlers will work correctly even if **NumLock** or **CapsLock** are activated.

In a GdkEventKey, the *`state`* field is a bit mask which indicates the modifier state at the time the key was pressed. Modifiers are keys like **Control** and **NumLock**. When implementing a GtkWidget::key_press_event handler, you should use `gtk_accelerator_get_default_mod_mask()` to test against modifier keys. This function returns a bit mask which encompasses all the modifiers which the user may be actively pressing, such as **Control**, **Shift**, and **Alt**, but ignores "inocuous" modifiers such as **NumLock** and **CapsLock**.

Say you want to see if **Control-F10** was pressed. Doing a simple test like `event->keysym == GDK_F10 && event->state == GDK_CONTROL_MASK` is not enough. If **CapsLock** is pressed, then *`event->state`* will be equal to `GDK_CONTROL_MASK | GDK_LOCK_MASK`, and the simple test will fail. By taking the logical-and of *`event->state`* and `gtk_accelerator_get_default_mod_mask()`, you can ignore the modifiers which are not actively pressed by the user at the same time as the base key.

The following example correctly tests for **Control-F10** being pressed.

```
static gboolean
my_widget_key_press_event_handler (GtkWidget *widget, GdkEventKey *event)
{
  guint modifiers;

  modifiers = gtk_accelerator_get_default_mod_mask ();

  if (event->keysym == GDK_F10
      && (event->state & modifiers) == GDK_CONTROL_MASK)
    {
      g_print ("Control-F10 was pressed\n");
      return TRUE;
    }

  return FALSE;
}
```

Changes from 1.0 to 1.2

Changes from 1.0 to 1.2 — Incompatible changes made between version 1.0 and version 1.2

Incompatible changes from 1.0 to 1.2

- GtkAcceleratorTable has been replaced with GtkAccelGroup.
- GtkMenuFactory has been replaced with GtkItemFactory, although a version of GtkMenuFactory is currently still provided to ease the migration phase.
- The GtkTypeInfo structures used in the `gtk_*_type_init()` functions have changed a bit, the old format:

```
GtkTypeInfo bin_info =
{
  "GtkBin",
  sizeof (GtkBin),
  sizeof (GtkBinClass),
  (GtkClassInitFunc) gtk_bin_class_init,
  (GtkObjectInitFunc) gtk_bin_init,
  (GtkArgSetFunc) NULL,
  (GtkArgGetFunc) NULL,
};
```

  needs to be converted to:

```
static const GtkTypeInfo bin_info =
{
  "GtkBin",
  sizeof (GtkBin),
  sizeof (GtkBinClass),
  (GtkClassInitFunc) gtk_bin_class_init,
  (GtkObjectInitFunc) gtk_bin_init,
  /* reserved_1 */ NULL,
  /* reserved_2 */ NULL,
  (GtkClassInitFunc) NULL,
};
```

  the `GtkArgSetFunc` and `GtkArgGetFunc` functions are not supported from the type system anymore, and you should make sure that your code only fills in these fields with `NULL` and doesn't use the deprecated function typedefs `(GtkArgSetFunc)` and `(GtkArgGetFunc)` anymore.
- A number of GTK+ functions were renamed. For compatibility, `gtkcompat.h` #define's the old 1.0.x function names in terms of the new names. To assure your GTK+ program doesn't rely on outdated function variants, compile

your program with `-DGTK_DISABLE_COMPAT_H` to disable the compatibility aliases. Here is the list of the old names and replacements:

Old	Replacement
gtk_accel_label_accelerator_width	gtk_accel_label_get_accel_width
gtk_check_menu_item_set_state	gtk_check_menu_item_set_active
gtk_container_border_width	gtk_container_set_border_width
gtk_label_set	gtk_label_set_text
gtk_notebook_current_page	gtk_notebook_get_current_page
gtk_packer_configure	gtk_packer_set_child_packing
gtk_paned_gutter_size	gtk_paned_set_gutter_size
gtk_paned_handle_size	gtk_paned_set_handle_size
gtk_scale_value_width	gtk_scale_get_value_width
gtk_style_apply_default_pixmap	gtk_style_apply_default_background
gtk_toggle_button_set_state	gtk_toggle_button_set_active
gtk_window_position	gtk_window_set_position

Note that `gtk_style_apply_default_background()` has an additional argument, `set_bg`. This parameter should be `FALSE` if the background is being set for a `NO_WINDOW` widget, otherwise `TRUE`.

- During the development phase of the 1.1.x line of GTK+ certain functions were deprecated and later removed. Functions affected are:

Removed	Replacement
gtk_clist_set_border	gtk_clist_set_shadow_type
gtk_container_block_resize	gtk_container_set_resize_mode
gtk_container_unblock_resize	gtk_container_set_resize_mode
gtk_container_need_resize	gtk_container_check_resize
gtk_ctree_show_stub	gtk_ctree_set_show_stub
gtk_ctree_set_reorderable	gtk_clist_set_reorderable
gtk_ctree_set_use_drag_icons	gtk_clist_set_use_drag_icons
gtk_entry_adjust_scroll	-
gtk_object_class_add_user_signal	gtk_object_class_user_signal_new
gtk_preview_put_row	gtk_preview_put
gtk_progress_bar_construct	gtk_progress_set_adjustment
gtk_scrolled_window_construct	gtk_scrolled_window_set_{h|v}adjustment
gtk_spin_button_construct	gtk_spin_button_configure
gtk_widget_thaw_accelerators	gtk_widget_unlock_accelerators
gtk_widget_freeze_accelerators	gtk_widget_lock_accelerators

Note that `gtk_entry_adjust_scroll()` is no longer needed as GtkEntry should automatically keep the scroll adjusted properly.

- Additionally, all `gtk_*_interp()` functions were removed. `gtk_*_full()` versions were provided as of GTK+ 1.0 and should be used instead.
- GtkButton has been changed to derive from GtkBin. To access a button's child, use `GTK_BIN (button)->child`, instead of the old `GTK_BUTTON (button)->child`.
- The selection API has been slightly modified: `gtk_selection_add_handler()` and `gtk_selection_add_handler_full()` have been removed. To supply the selection, one now registers the targets one is interested in with:

```
void gtk_selection_add_target (GtkWidget          *widget,
                               GdkAtom             selection,
                               GdkAtom             target,
                               guint               info);
```

or:

```
void gtk_selection_add_targets (GtkWidget          *widget,
                                GdkAtom             selection,
                                GtkTargetEntry     *targets,
                                guint               ntargets);
```

When a request for a selection is received, the new "selection_get" signal will be called:

```
 void  "selection_get"          (GtkWidget          *widget,
                                 GtkSelectionData   *selection_data,
                                 guint               info,
                                 guint               time);
```

A "time" parameter has also been added to the "selection_received" signal.

```
void  "selection_received"     (GtkWidget          *widget,
                                GtkSelectionData   *selection_data,
                                guint               time);
```

- The old drag and drop API has been completely removed and replaced. See the reference documentation for details on the new API.
- Support for Themes has been added. In general, this does not affect application code, however, a few new rules should be observed:
  - To set a shape for a window, you must use `gtk_widget_shape_combine_mask()` instead of `gdk_window_shape_combine_mask()`, or the shape will be reset when switching themes.
  - It is no longer permissable to draw directly on an arbitrary widget, or to set an arbitrary widget's background pixmap. If you need to do that, use a GtkDrawingArea or (for a toplevel) a GtkWindow where `gtk_widget_set_app_paintable()` has been called.

- The GtkScrolledWindow widget no longer creates a GtkViewport automatically. Instead, it has been generalized to accept any "self-scrolling" widget.

  The self-scrolling widgets in the GTK+ core are GtkViewport, GtkCList, GtkCTree, GtkText, and GtkLayout. All of these widgets can be added to a scrolled window as normal children with `gtk_container_add()` and scrollbars will be set up automatically.

  To add scrollbars to a non self-scrolling widget, (such as a GtkList), first add it to a viewport, then add the viewport to a scrolled window. The scrolled window code provides a convenience function to do this:

  ```
  void gtk_scrolled_window_add_with_viewport (GtkScrolledWindow *scrollwin,
                                              GtkWidget         *child);
  ```

  This does exactly what it says - it creates a viewport, adds the child widget to it, then adds the viewport to the scrolled window.

  The scrollbars have been removed from the GtkCList and GtkCTree, because they are now scrolled by simply adding them to a scrolled window. The scrollbar policy is set on the scrolled window with `gtk_scrolled_window_set_policy()` and not on the child widgets (e.g. GtkCList's `gtk_clist_set_policy()` was removed).
- The "main loop" of GTK+ has been moved to GLib. This should not affect existing programs, since compatibility functions have been provided. However, you may want to consider migrating your code to use the GLib main loop directly.
- the `GTK_BASIC` flag was removed, and with it the corresponding macro and function `GTK_WIDGET_BASIC()` and `gtk_widget_basic()`.
- All freeze/thaw methods are now recursive - that is, if you freeze a widget n times, you must also thaw it n times. Therefore, if you have code like:

  ```
  gboolean frozen;
  frozen = GTK_CLIST_FROZEN (clist);
  gtk_clist_freeze (clist);
  [...]
  if (!frozen)
    gtk_clist_thaw (clist);
  ```

  it will not work anymore. It must be, simply:

  ```
  gtk_clist_freeze (clist);
  [...]
  gtk_clist_thaw (clist);
  ```

- The thread safety in GTK+ 1.2 is slightly different than that which appeared in early versions in the 1.1 development track. The main difference is that it relies on the thread primitives in GLib, and on the thread-safe GLib main loop.

  This means:

- You must call `g_thread_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.
- There is no longer a special `--with-threads` **configure** option for GTK+. To use threads in a GTK+ program, you must:
  1. If you want to use the native thread implementation, make sure GLib found this in configuration, otherwise, call you must provide a thread implementation to `g_thread_init()`.
  2. Link with the libraries returned by **gtk-config --libs gthread** and use the cflags from **gtk-config --cflags gthread**. You can get these `CFLAGS` and `LIBS` by passing `gthread` as the fourth parameter to the `AM_PATH_GTK` automake macro.

- Prior to GTK+ 1.2, there were two conflicting interpretations of `widget->requisition`. It was either taken to be the size that the widget requested, or that size modified by calls to `gtk_widget_set_usize()`. In GTK+ 1.2, it is always interpreted the first way.

  Container widgets are affected in two ways by this:
  1. Container widgets should not pass `widget->requisition` as the second parameter to `gtk_widget_size_request()`. Instead they should call it like:

     ```
     GtkRequisition child_requisition;
     gtk_widget_size_request (widget, &child_requisition);
     ```

  2. Container widgets should not access `child->requisition` directly. Either they should use the values returned by `gtk_widget_size_request()`, or they should call the new function:

     ```
     void gtk_widget_get_child_requisition (GtkWidget      *widget,
                                            GtkRequisition *requisition);
     ```

     which returns the requisition of the given widget, modified by calls to `gtk_widget_set_usize()`.

<< Test for modifier keys correctly **Changes from 1.2 to 2.0 >>**

Changes from 1.2 to 2.0

Changes from 1.2 to 2.0 — Incompatible changes made between version 1.2 and version 2.0

Incompatible changes from 1.2 to 2.0

The GNOME 2.0 porting guide on http://developer.gnome.org has some more detailed discussion of porting from 1.2 to 2.0. See the sections on GLib and GTK+.

GTK+ changed fairly substantially from version 1.2 to 2.0, much more so than from 1.0 to 1.2. Subsequent updates (possibilities are 2.0 to 2.2, 2.2 to 2.4, then to 3.0) will almost certainly be much, much smaller. Nonetheless, most programs written for 1.2 compile against 2.0 with few changes. The bulk of changes listed below are to obscure features or very specialized features, and compatibility interfaces exist whenever possible.

- `gtk_container_get_toplevels()` was removed and replaced with `gtk_window_list_toplevels()`, which has different memory management on the return value (`gtk_window_list_toplevels()` copies the GList and also references each widget in the list, so you have to `g_list_free()` the list after first unref'ing each list member).
- The `gdk_time*` functions have been removed. This functionality has been unused since the main loop was moved into GLib prior to 1.2.
- The signature for `GtkPrintFunc` (used for `gtk_item_factory_dump_items()`) has been changed to take a const gchar * instead of gchar *, to match what we do for GLib, and other similar cases.
- The detail arguments in the GtkStyleClass structure are now const gchar *.
- `gtk_paned_set_gutter_size()` has been removed, since the small handle tab has been changed to include the entire area previously occupied by the gutter.
- `gtk_paned_set_handle_size()` has been removed, in favor of a style property, since this is an option that only makes sense for themes to adjust.
- GDK no longer selects OwnerGrabButtonMask for button presses. This means that the automatic grab that occurs when the user presses a button will have `owner_events = FALSE`, so all events are redirected to the grab window, even events that would normally go to other windows of the window's owner.
- GtkColorSelectionDialog has now been moved into it's own set of files, `gtkcolorseldialog.c` and `gtkcolorseldialog.h`.
- `gtk_widget_shape_combine_mask()` now keeps a reference count on the mask pixmap that is passed in.
- The GtkPatternSpec has been moved to GLib as GPatternSpec, the pattern arguments to `gtk_item_factory_dump_items()` and `gtk_item_factory_dump_rc()` have thusly been changed to take a GPatternSpec instead of a GtkPatternSpec.
- Type system changes:
  - `GTK_TYPE_OBJECT` is not a fundamental type anymore. Type checks of the style `(GTK_FUNDAMENTAL_TYPE (some_type) == GTK_TYPE_OBJECT)` will not work anymore. As a replacement, `(GTK_TYPE_IS_OBJECT (some_type))` can be used now.
  - The following types vanished: `GTK_TYPE_ARGS`, `GTK_TYPE_CALLBACK`, `GTK_TYPE_C_CALLBACK`, `GTK_TYPE_FOREIGN`. The corresponding GtkArg fields and field access macros are also gone.
  - The following type aliases vanished: `GTK_TYPE_FLAT_FIRST`, `GTK_TYPE_FLAT_LAST`, `GTK_TYPE_STRUCTURED_FIRST`, `GTK_TYPE_STRUCTURED_LAST`.
  - The type macros `GTK_TYPE_MAKE()` and `GTK_TYPE_SEQNO()` vanished, use of `GTK_FUNDAMENTAL_TYPE()` is discouraged. Instead, the corresponding GType API should be used: `G_TYPE_FUNDAMENTAL()`, `G_TYPE_DERIVE_ID()`, `G_TYPE_BRANCH_SEQNO()`. Note that the GLib type system doesn't build new type ids based on a global incremental sequential number anymore, but numbers new type ids sequentially per fundamental type branch.
  - The following type functions vanished/were replaced:

Old Function	Replacement
`gtk_type_query()`	being investigated
`gtk_type_set_varargs_type()`	-
`gtk_type_get_varargs_type()`	-
`gtk_type_check_object_cast()`	`g_type_check_instance_cast()`
`gtk_type_check_class_cast()`	`g_type_check_class_cast()`
`gtk_type_describe_tree()`	-
`gtk_type_describe_heritage()`	-
`gtk_type_free()`	-
`gtk_type_children_types()`	`g_type_children()`
`gtk_type_set_chunk_alloc()`	*`GTypeInfo.n_preallocs`*
`gtk_type_register_enum()`	`g_enum_register_static()`
`gtk_type_register_flags()`	`g_flags_register_static()`
`gtk_type_parent_class()`	`g_type_parent()`/`g_type_class_peek_parent()`

  Use of `g_type_class_ref()`/`g_type_class_unref()` and `g_type_class_peek()` is recommended over usage of `gtk_type_class()`. Use of `g_type_register_static()`/`g_type_register_dynamic()` is recommended over usage of `gtk_type_unique()`.

Changes from 1.2 to 2.0

- Object system changes: GtkObject derives from GObject, so is not the basic object type anymore. This imposes the following source incompatible changes:
  - GtkObject has no *klass* field anymore, an object's class can be retrieved with the object's coresponding GTK_<OBJECT>_GET_CLASS (object) macro.
  - GtkObjectClass has no *type* field anymore, a class's type can be retrived with the GTK_CLASS_TYPE (class) macro.
  - GtkObjectClass does not introduce the finalize() and shutdown() methods anymore. While shutdown() is intended for GTK+ internal use only, finalize() is required by a variety of object implementations. GObjectClass.finalize should be overriden here, e.g.:

```
static void gtk_label_finalize (GObject *gobject)
{
  GtkLabel *label = GTK_LABEL (gobject);

  G_OBJECT_CLASS (parent_class)->finalize (object);
}
static void gtk_label_class_init (GtkLabelClass *class)
{
  GObjectClass *gobject_class = G_OBJECT_CLASS (class);

  gobject_class->finalize = gtk_label_finalize;
}
```

- The GtkObject::destroy signal can now be emitted multiple times on an object. ::destroy implementations should check that make sure that they take this into account, by checking to make sure that resources are there before freeing them. For example:

```
if (object->foo_data)
  {
    g_free (object->foo_data);
    object->foo_data = NULL;
  }
```

  Also, ::destroy implementations have to release object references that the object holds. Code in finalize implementations such as:

```
if (object->adjustment)
  {
    gtk_object_unref (object->adjustment);
    object->adjustment = NULL;
  }
```

  have to be moved into the ::destroy implementations. The reason for doing this is that all object reference cycles should be broken at destruction time. Because the ::destroy signal can be emitted multiple times, it no longer makes sense to check if a widget has been destroyed using the GTK_OBJECT_DESTROYED() macro, and this macro has been removed. If catching destruction is still needed, it can be done with a signal connection to ::destroy.

- Signal system changes: The GTK+ 2.0 signal system merely proxies the GSignal system now. For future usage, direct use of the GSignal API is recommended, this avoids significant performance hits where GtkArg structures have to be converted into GValues. For language bindings, GSignal+GClosure provide a much more flexible and convenient mechanism to hook into signal emissions or install class default handlers, so the old GtkSignal API for language bindings is not supported anymore.

  Functions that got removed in the GTK+ signal API: gtk_signal_n_emissions(), gtk_signal_n_emissions_by_name(), gtk_signal_set_funcs(), gtk_signal_handler_pending_by_id(), gtk_signal_add_emission_hook(), gtk_signal_add_emission_hook_full(), gtk_signal_remove_emission_hook(), gtk_signal_query(). Also, the GtkCallbackMarshal argument to gtk_signal_connect_full() is not supported anymore. For many of the removed functions, similar variants are available in the g_signal_* namespace. The GSignal system performs emissions in a slightly different manner than the old GtkSignal code. Signal handlers that are connected to signal "foo" on object "bar" while "foo" is being emitted, will not be called anymore during the emission they were connected within.
- Inserting and deleting text in GtkEntry though functions such as gtk_entry_insert_text() now leave the cursor at its original position in the text instead of moving it to the location of the insertion/deletion.
- The *label* field of GtkFrame widgets has been removed (as part of a change to allow arbitrary widgets in the title position). The text can now be retrieved with the new function gtk_frame_get_text().
- The 'font' and 'font_set' declarations in RC files are now ignored. There is a new 'font_name' field that holds the string form of a Pango font.
- A number of types in GDK have become subclasses of GObject. For the most part, this should not break anyone's code. However, it's now possible/encouraged to use g_object_ref()/g_object_unref() and other GObject features with these GDK types. The converted types are: GdkWindow, GdkDrawable, GdkPixmap, GdkImage, GdkGC, GdkDragContext, GdkColormap.
- All drawables including pixmaps used to have a type tag, the GdkWindowType enumeration, which included GDK_WINDOW_PIXMAP. GdkWindowType is now a property of GdkWindow *only*, and there is no GDK_WINDOW_PIXMAP. You can use the GDK_IS_PIXMAP() macro to see if you have a pixmap, if you need to know that.
- GtkStyle and GtkRcStyle are now subclasses of GObject as well. This requires fairly extensive changes to theme engines, but shouldn't affect most other code.
- *xthickness* and *ythickness* have moved from GtkStyleClass to GtkStyle (from class to instance). This gives themes a bit more flexibility and is generally more of the Right Thing. You can trivially fix your code with s/style->klass->xthickness/style->xthickness/g and same for ythickness.
- Some GtkStyle draw_* methods have been removed (cross, oval, ramp) and others have been added (expander, layout). This will require changes to theme engines.
- If you were using private GDK types, they have been rearranged significantly. You shouldn't use private types. ;-)

Changes from 1.2 to 2.0

- The visual for a widget, and also the default visual is now derived from the colormap for the widget and the default colormap. `gtk_widget_set_visual()`, `gtk_widget_set_default_visual()`, `gtk_widget_push_visual()` and `gtk_widget_pop_visual()` now do nothing. Since the visual always had to match that of the colormap, it is safe to simply delete all references to these functions.
- A number of functions in GDK have been renamed for consistency and clarity. #defines to provide backwards compatibility have been included, but can be disabled by defining `GDK_DISABLE_DEPRECATED`.

Old function	**Defined As**
`gdk_draw_pixmap`	`gdk_draw_drawable`
`gdk_draw_bitmap`	`gdk_draw_drawable`
`gdk_window_get_size`	`gdk_drawable_get_size`
`gdk_window_get_type`	`gdk_window_get_window_type`
`gdk_window_get_colormap`	`gdk_drawable_get_colormap`
`gdk_window_set_colormap`	`gdk_drawable_set_colormap`
`gdk_window_get_visual`	`gdk_drawable_get_visual`
`gdk_window_ref`	`gdk_drawable_ref`
`gdk_window_unref`	`gdk_drawable_unref`
`gdk_bitmap_ref`	`gdk_drawable_ref`
`gdk_bitmap_unref`	`gdk_drawable_unref`
`gdk_pixmap_ref`	`gdk_drawable_ref`
`gdk_pixmap_unref`	`gdk_drawable_unref`
`gdk_gc_destroy`	`gdk_gc_unref`
`gdk_image_destroy`	`gdk_image_unref`
`gdk_cursor_destroy`	`gdk_cursor_unref`
`gdk_window_copy_area(drawable,gc,x,y,source_drawable,source_x,source_y,width,height)`	`gdk_draw_pixmap(drawable,gc,source_drawable,source_x,source_y,x,y,width,height)`
`gdk_rgb_get_cmap`	`gdk_rgb_get_colormap`

(Note that `g_object_ref()` and `g_object_unref()` may be used for all of the above ref and unref functions.) `gtk_widget_popup()` was removed, it was only usable for GtkWindows, and there the same effect can be achieved by `gtk_window_move()` and `gtk_widget_show()`.

- `gdk_pixmap_foreign_new()` no longer calls `XFreePixmap()` on the pixmap when the GdkPixmap is finalized. This change corresponds to the behavior of `gdk_window_foreign_new()`, and fixes a lot of problems with code where the pixmap wasn't supposed to be freed. If `XFreePixmap()` is needed, it can be done using the destroy-notification facilities of `g_object_set_data()`.
- GtkProgress/GtkProgressBar had serious problems in GTK+ 1.2.
  - Only 3 or 4 functions are really needed for 95% of progress interfaces; GtkProgress/GtkProgressBar had about 25 functions, and didn't even include these 3 or 4.
  - In activity mode, the API involves setting the adjustment to any random value, just to have the side effect of calling the progress bar update function - the adjustment is totally ignored in activity mode.
  - You set the activity step as a pixel value, which means to set the activity step you basically need to connect to size_allocate.
  - There are `ctree_set_expander_style()`-functions, to randomly change look-and-feel for no good reason.
  - The split between GtkProgress and GtkProgressBar makes no sense to me whatsoever.

This was a big wart on GTK+ and made people waste lots of time, both learning and using the interface. So, we have added what we feel is the correct API, and marked all the rest deprecated. However, the changes are 100% backward-compatible and should break no existing code. The following 5 functions are the new programming interface and you should consider changing your code to use them:

```
  void       gtk_progress_bar_pulse               (GtkProgressBar *pbar);
  void       gtk_progress_bar_set_text            (GtkProgressBar *pbar,
                                                   const gchar    *text);
  void       gtk_progress_bar_set_fraction        (GtkProgressBar *pbar,
                                                   gfloat          fraction);

  void       gtk_progress_bar_set_pulse_step      (GtkProgressBar *pbar,
                                                   gfloat          fraction);
  void       gtk_progress_bar_set_orientation     (GtkProgressBar *pbar,
                                                   GtkProgressBarOrientation
orientation);
```

- The GtkNotebookPage structure has been removed from the public header files; this was never meant to be a public structure, and all functionality that could be done by accessing the struct fields of this structure should be accessible otherwise.
- Negative values of the *`position`* parameter to `gtk_notebook_reorder_child()` now cause the page to be appended, not inserted at the beginning. (This gives consistency with `gtk_box_reorder_child()`, `gtk_menu_reorder_child()`.)
- `GtkMenuPositionFunc` has a new parameter `push_in` which controls how menus placed outside the screen is handled. If this is set to `TRUE` and part of the menu is outside the screen then GTK+ pushes it into the visible area. Otherwise the menu is cut of at the end of the visible screen area.

Regardless of what happens to the size of the menu, the result is always that the items are placed in the same place as if the menu was placed outside the screen, using menu scrolling if necessary.

- The "draw" signal and virtual method on GtkWidget has been removed. All drawing should now occur by invalidating a region of the widget (call `gdk_window_invalidate_rect()` or `gtk_widget_queue_draw()` for example to invalidate a region). GTK+ merges all invalid regions, and sends expose events to the widget in an idle handler for the invalid regions. `gtk_widget_draw()` is deprecated but still works; it adds the passed-in area to the invalid region and immediately sends expose events for the current invalid region. Most widgets will work fine if you just delete their "draw" implementation, since they will already have working expose_event implementations. The draw method was rarely called in practice anyway.
- The GdkExposeEvent has a new *`region`* field. This can be used instead of the *`area`* field if you want a more exact representation of the area to update.
- Sending synthetic exposes using `gtk_widget_event()` is no longer allowed. If you just need an expose call you should use `gdk_window_invalidate_rect()` or `gdk_window_invalidate_region()` instead. For the case of container widgets that need to propagate expose events to `NO_WINDOW` children you can either use `gtk_container_propagate_expose()`, or chain to the default container expose handler.
- The draw_default and draw_focus methods/signals on GtkWidget are gone; simply draw things in your expose handler. `gtk_widget_draw_focus()` and `gtk_widget_draw_default()` wrapper functions are also gone; just queue a draw on the widget, or the part affected by the focus/default anyway. Also, GtkWidget now has default implementations for focus_in_event and focus_out_event. These set/unset `GTK_HAS_FOCUS`, and queue a draw. So if your focus in/out handler just does that, you can delete it.
- GtkText and GtkTree are buggy and broken. We don't recommend using them, and changing old code to avoid them is a good idea. The recommended alternatives are GtkTextView and GtkTreeView. The broken widgets are not declared in the headers by default; to use them, define the symbol `GTK_ENABLE_BROKEN` during compilation. In some future release, these widgets will be removed from GTK+.
- GdkColorContext is gone; you probably weren't using it anyway. Use GdkColormap and the `gdk_rgb_*` functions instead.
- GtkMenuBar now draws the *`GtkContainer::border_width`* space outside the frame, not inside the frame.
- In GTK+ 1.2, if an event handler returned `TRUE` it prevented propagation of that event to parent widgets. That is, the event signal would not be emitted on parent widgets. In GTK+ 2.0, if an event handler returns `TRUE`, the current signal emission on the current widget is immediately stopped. That is, other callbacks connected to the signal will not be invoked.
- `gtk_toolbar_new()` no longer has arguments. This function was broken because the default `GtkToolbarStyle` (icons, text, both) is now a user preference, which is overridden when you call `gtk_toolbar_set_style()`. The constructor forced everyone to override the preference, which was undesirable. So to port your app, decide if you want to force the toolbar style or conform to the user's global defaults; if you want to force it, call `gtk_toolbar_set_style()`.

  The orientation arg was removed from `gtk_toolbar_new()` as well, just because it wasn't very useful and we were breaking the function anyway so had an opportunity to lose it. Call `gtk_toolbar_set_orientation()` to set toolbar orientation.
- GtkRange/GtkScrollbar/GtkScale were rewritten; this means that most theme engines won't draw them properly, and any custom subclasses of these widgets will need a rewrite (though if you could figure out how to subclass the old version of GtkRange, you have our respect). Also, `GtkTroughType` is gone.

  Here are some notable changes:
  - `stepper_size` style property is the height for vertical ranges, width for horizontal; the other dimension matches the trough size.
  - Added the ability to do NeXT-style steppers (and several other styles that don't make any sense).
  - Added `min_slider_length`, `fixed_slider_length` properties to GtkScrollbar.
  - Cleaned some private (or at least useless) functions out of `gtkscale.h`, e.g. `gtk_scale_value_width`.
  - Moved bindings from subclasses to GtkScale, even arrow keys, since blind users don't know scale orientation.
  - Changed `move_slider` action signal to use new GtkScrollType, remove GtkTroughType argument.
  - Digits rounds the values a range will input to the given number of decimals, but will not try to force adjustment values set by other controllers. That is, we no longer modify `adjustment->value` inside a `value_changed` handler.
  - Added getters for GtkScale setters.
  - Middle-click begins a slider drag.

- The GtkContainer::focus signal/virtual function and `gtk_container_focus()` call were replaced by GtkWidget::focus and `gtk_widget_child_focus()`. The semantics are the same, so you should be able to just replace `container_class->focus = mywidget_focus` with `widget_class->focus = mywidget_focus` and replace `gtk_container_focus()` calls with `gtk_widget_child_focus()` calls.

  The purpose of this change was to allow non-containers to have focusable elements.
- `gtk_rc_set_image_loader()` and `gtk_rc_load_image()` have been removed, now that GTK+ includes decent image loading capabilities itself.
- An extra GtkSettings argument has been added to `gtk_rc_find_pixmap_in_path()`. This function is only actually useful from a theme engine during parsing, at which point the GtkSettings is provided.
- The child argument facility in `gtkcontainer.c` has been converted to a child property facility using GParamSpec and other facilities for GObject.
- The `set_child_arg()` and `get_child_arg()` virtual methods have been replaced with `set_child_property()`/`get_child_property()`, which work similar to GObject->set_property/get_property.
- Other removed GtkContainer functions with the replacements:

Old function	Replacement
`gtk_container_add_child_arg_type`	`gtk_container_class_install_child_property`
`gtk_container_query_child_args`	`gtk_container_class_list_child_properties`
`gtk_container_child_getv`	`gtk_container_child_set_property`
`gtk_container_child_setv`	`gtk_container_child_get_property`
`gtk_container_add_with_args`	`gtk_container_add_with_properties`
`gtk_container_addv`	`gtk_container_add`/`gtk_container_child_set_property`

- `gdk_image_get()` (or rather its replacement, `gdk_drawable_get_image()`) now handles errors properly by returning `NULL`, previously it would crash. Also, a window being offscreen is no longer considered an error; instead, the area contains undefined contents for the offscreen areas. In most cases, code using `gdk_image_get()` should really be ported to `gdk_pixbuf_get_from_drawable()`.
- `gtk_widget_set_usize()` has been renamed to `gtk_widget_set_size_request()`, however the old name still exists unless you define `GTK_DISABLE_DEPRECATED`.
- `gtk_widget_set_uposition()` is deprecated; use `gtk_window_move()`, `gtk_fixed_put()`, or `gtk_layout_put()` instead.
- `gtk_window_set_policy()` is deprecated. To get the effect of "allow_shrink", call `gtk_widget_set_size_request (window, 0, 0)`. To get the effect of "allow_grow", call `gtk_window_set_resizable (window, TRUE)`. You didn't want the effect of "auto_shrink", it made no sense. But maybe if you were using it you want to use `gtk_window_resize (window, 1, 1)` to snap a window back to its minimum size (the 1, 1 will be rounded up to the minimum window size).

- The core GTK+ now takes care of handling mapping, unmapping and realizing the child widgets of containers in `gtk_widget_set_parent()`. In most cases, this allows container implementations to be simplified by removing the code in `add()` methods to map and realize children. However, there are a couple of things to watch out for here:
- If the parent is realized before the `add()` happens, `gtk_widget_set_parent_window()` must be called before `gtk_widget_set_parent()`, since `gtk_widget_set_parent()` will realize the child.
- If a container depended on its children not being mapped unless it did so itself (for example, GtkNotebook only mapped the current page), then the new function `gtk_widget_set_child_visible()` must be called to keep widgets that should not be mapped not mapped.

  As part of this change, most containers also will no longer need custom implementations of the `map()` and `unmap()` virtual functions. The only cases where this is necessary are:
  - For `!NO_WINDOW` widgets, if you create children of `widget->window` and don't map them in `realize()` then you must map them in `map()`. [In almost all cases, you can simply map the windows in `realize()`.]
  - For `NO_WINDOW` widgets, if you create windows in your `realize()` method, you must map then in `map()` and unmap them in `unmap()`.

- `gtk_widget_set_default_style()`, `gtk_widget_push_style()`, and `gtk_widget_pop_style()` have been removed, since they did not work properly with themes and there were better alternatives for modifying the appearance of widgets. You should generally use `gtk_widget_modify_*()` instead.
- `gtk_image_new()` now takes no arguments and creates an empty GtkImage widget. To create a GtkImage widget from a GdkImage (the least common usage of GdkImage), use `gtk_image_new_from_image()`.
- `GTK_SELECTION_EXTENDED` is now deprecated, and neither the GtkList/GtkTree nor the GtkCList/GtkCTree support `GTK_SELECTION_EXTENDED` anymore. However, the old extended behavior replaces `MULTIPLE` behavior.
- The following variables are no longer exported from GDK. (Other variables are also no longer exported; the following are the ones found used externally in a large sample of GTK+ code.)

Variable	Replacement
`gdk_null_window_warnings`	None - did nothing in GTK+ 1.2
`gdk_leader_window`	None - private variable
`gdk_screen`	`gdk_x11_get_default_screen ()`
`gdk_root_window`	`gdk_x11_get_default_root_xwindow ()`
`gdk_root_parent`	`gdk_get_default_root_window ()`
`gdk_error_code`	`gdk_error_trap_push ()/pop ()`
`gdk_error_warnings`	`gdk_error_trap_push ()/pop ()`
`gdk_display_name`	`gdk_get_display ()`
`gdk_wm_delete_window`	`gdk_atom_intern ("WM_DELETE_WINDOW", FALSE)`
`gdk_wm_take_focus`	`gdk_atom_intern ("WM_TAKE_FOCUS", FALSE)`
`gdk_wm_protocols`	`gdk_atom_intern ("WM_PROTOCOLS", FALSE)`

- The handling of colormaps and widgets has been changed:
  - The default colormap for widgets is now the GdkRGB colormap, not the system default colormap. If you try to use resources created for a widget (e.g., `widget->style`) with a window using the system colormap, errors will result on some machines.
  - `gtk_widget_push()`/`gtk_widget_pop_colormap()` only cause the colormap to be explicitly set on toplevel widgets, not on all widgets. The colormap for other widgets (when not set using `gtk_widget_set_colormap()`), is determined by finding the nearest ancestor with a colormap set on it explicitly, or if that fails, the default colormap.

- The default selected day for GtkCalendar is now the current day in the month, not the first day in the month. The current month and year were already used.
- GDK is no longer put into threaded mode automatically when `g_thread_init()` has been called. In order to use the global GDK thread mutex with `gdk_threads_enter()` and `gdk_threads_leave()`, you must call `gdk_threads_init()` explicitly. If you aren't using GDK and GTK+ functions from multiple threads, there is no reason to call `gdk_threads_init()`.
- The GtkPreviewInfo struct has had its visual and colormap fields removed. Also, `gtk_preview_get_cmap()` and `gtk_preview_get_visual()` are deprecated, as GdkRGB works on any colormap and visual. You no longer need to `gtk_widget_push_cmap (gtk_preview_get_cmap ())` in your code.
- The GtkBox, GtkTable, and GtkAlignment widgets now call `gtk_widget_set_redraw_on_allocate (widget, FALSE);` on themselves. If you want to actually draw contents in a widget derived from one of these widgets, you'll probably want to change this in your `init()` function.
- A number of widgets are now `NO_WINDOW` widgets (most importantly GtkButton, but also GtkRange and GtkNotebook) This has a couple of effects:
  - If you are deriving from one of these widgets, you need to adapt your code appropriately -- for instance, drawing coordinates start from `widget->allocation.x, widget->allocation.y`.
  - If you are embedding one of these widgets in a custom widget, you must make sure you call `gtk_container_propagate_expose()` correctly, as you must for any `NO_WINDOW` widgets.

  GtkFixed is a little special; it is now created by default as a `NO_WINDOW` widget, but if you do

```
gtk_fixed_set_has_window (fixed, TRUE);
```

  after creating a fixed widget, it will create a window and handle it properly.

- GtkLayout no longer has the *`xoffset`*, *`yoffset`* fields, which used to store the difference between world and window coordinates for `layout->bin_window`. These coordinate systems are now always the same.
- `gtk_paint_focus()`, `gtk_draw_focus()` and `GtkStyle::draw_focus()` have been changed a bit:
  - A `GtkStateType` argument has been added to `gtk_paint_focus()`.
  - The default implementation of the `GtkStyle::draw_focus()` virtual function now draws a focus rectangle whose width is determined by the GtkWidget::focus-width style property.
  - The rectangle passed in is the bounding box, instead of the rectangle used in the `gdk_draw_rectangle()` call, so it is no longer necessary to subtract 1 from the width and height.

<< Changes from 1.0 to 1.2 **Migrating from GtkFileSelection to GtkFileChooser >>**

Migrating from GtkFileSelection to GtkFileChooser

Federico Mena-Quintero

<federico@ximian.com>

GtkFileChooser, starting with GTK+ 2.4, is the new set of APIs for file selection widgets and dialogs. Previous versions of GTK+ used GtkFileSelection, which has numerous problems.

GtkFileChooser is an abstract interface that can be implemented by widgets that perform file selection tasks. Two widgets in GTK+ implement this interface: GtkFileChooserDialog and GtkFileChooserWidget. Most applications simply need to use GtkFileChooserDialog, which is a dialog box that allows the user to select existing files for opening them, or to pick new filenames for saving documents. GtkFileChooserWidget is for special applications that need to embed a file selection widget inside a larger window. In the context of GTK+, GtkFileChooserDialog is simply a GtkDialog box with a GtkFileChooserWidget inside.

Creating a GtkFileChooserDialog

To create a GtkFileChooserDialog, you simply call gtk_file_chooser_dialog_new(). This function is similar to gtk_dialog_new() in that it takes parameters for the title of the dialog box and its transient parent, as well as its buttons. In addition, it takes in an argument that determines whether the file chooser dialog will be used for opening existing files or for saving to a possibly new file.

Please see Example 7, “Typical usage” for how to create a simple file chooser dialog and extract the selected filename from it.

Selection Modes

GtkFileChooser can be used in two modes, to select a single file at a time or to select a set of more than one file. To set this, use gtk_file_chooser_set_select_multiple(). In single-selection mode, you can use gtk_file_chooser_get_filename() to get a file name from the local file system or gtk_file_chooser_get_uri() to get a full-formed URI. In multiple-selection mode, you can use gtk_file_chooser_get_filenames() to get a GSList of filename strings, or gtk_file_chooser_get_uris() to get a list of URI strings.

Also, you can configure GtkFileChooser to select files or folders. Consider a backup program that needs to let the user select a folder that will be backed up along with its subfolders. To configure whether GtkFileChooser is used to select files or folders, use gtk_file_chooser_set_action(). In addition, this lets you configure whether the file chooser will be used to select existing files or folders (e.g. for "File/Open"), or to type in new filenames (for "File/Save As...").

<< Migrating from GtkFileSelection to GtkFileChooser **Installing a Preview widget >>**

Installing a Preview widget

Many applications need to have a preview facility within their file chooser dialogs. Previous to GTK+ 2.4, one needed to access the GtkFileSelection widget hierarchy directly to hook in a preview widget. With GtkFileChooser, there is a dedicated API to do this.

Please see the section on creating preview widgets for more information.

<< Selection Modes **Installing Extra Widgets >>**

Installing Extra Widgets

Some applications need to install extra widgets in a file chooser. For example, an application may want to provide a toggle button to give the user the option of opening a file read-only.

Please see the section on creating extra widgets for more information.

New features

New features in GtkFileChooser include the following:

- Ability to select URIs rather than just local files. You must use a GtkFileSystem implementation that supports this, for example the gnome-vfs backend.
- Present a list of application-specific shortcut folders. For example, a paint program may want to add a shortcut for its `/usr/share/paint_program/Clipart` folder.
- Define custom filters so that not all the files in a folder are listed. For example, you could filter out backup files, or show only image files.

To see how to use these features, please consult the GtkFileChooser reference documentation.

<< Installing Extra Widgets **Migrating from old menu and toolbar systems to GtkAction >>**

Migrating from old menu and toolbar systems to GtkAction

Federico Mena-Quintero

<federico@ximian.com>

Prior to GTK+ 2.4, there were several APIs in use to create menus and toolbars. GTK+ itself included GtkItemFactory, which was historically used in the GIMP; libgnomeui provided the gnome-ui set of macros; libbonoboui provided a complex mechanism to do menu merging across embedded components. GTK+ 2.4 includes a system for creating menus and toolbars, with merging of items, based around the GtkAction mechanism.

Actions and Action Groups

A GtkAction represents an operation that the user can perform from the menus and toolbars of an application. It is similar to "verbs" in other menu systems. A GtkAction has a name, which is its identifier, and it can have several widgets that represent it in the user interface. For example, an action for EditCopy can have a menu item as well as a toolbar button associated to it. If there is nothing selected in the document, the application can simply de-sensitize the EditCopy action; this will cause both the menu item and the toolbar button to be de-sensitized automatically. Similarly, whenever the user selects the menu item or the toolbar button associated to the EditCopy action, the corresponding GtkAction object will emit an "activate" signal.

GtkActionGroup is simply a group of GtkAction objects. An application may want to have several groups: one for global actions such as "new document", "about", and "exit"; then one group for each open document with actions specific to the document, such as "cut", "copy", "paste", and "print".

Normal actions are simply commands, such as FileSave or EditCopy. Toggle actions can be active or inactive, such as FormatBold or ViewShowRulers. Radio actions define a set of items for which one and

only one can be active at a time, for example, { ViewHighQuality, ViewNormalQuality, ViewLowQuality }.

<< New features **User Interface Manager Object >>**

User Interface Manager Object

GtkUIManager is an object that can construct menu and toolbar widgets from an XML description. These widgets are in turn associated to corresponding actions and action groups.

GtkUIManager supports merging of menus and toolbars for applications that have multiple components, each with separate sets of commands. For example, a word processor that can embed images may want to have toolbar buttons for Bold and Italic when the cursor is on a text block, but Crop and Brightness/Contrast buttons when the cursor is on an image. These actions, which change depending on the state of the application, can be merged and de-merged from a GtkUIManager as appropriate.

<< Migrating from old menu and toolbar systems to GtkAction Migrating from GnomeUIInfo >>

Migrating from GnomeUIInfo

Prior to GTK+ 2.4, some applications used the GnomeUIInfo mechanism from `<libgnomeui/gnome-app-helper.h>` to define their menus and toolbars. With it, a program decleres an array of GnomeUIInfo structures, which contain information for menu or toolbar items such as their label, icon, and accelerator key. Then, one calls `gnome_app_fill_menu()` or `gnome_app_fill_toolbar()`, or one of the related functions, to create the appropriate widgets based on these structures.

A downside of this API is that the same structures are used to pass back pointers to the widgets that got created. This means that the structures cannot simply be kept around if the program requires multiple instances of the user interface (e.g. several windows); each new invocation of `gnome_app_fill_menu()` would overwrite the widget fields of the structures.

Another disadvantage is that there is no automatic way to synchronize the state of related controls. If there are toolbar toogle buttons for "Bold", "Italic", "Underline", and also corresponding menu items under "Format/Bold", etc., one has to synchronize their toggled states by hand whenever the user selects any one of them.

Finally, there is no way to do menu and toolbar merging for applications that require embedded components.

To convert an application that uses GnomeUIInfo into the new GtkAction mechanism, you need to do several things:

1. Separate your existing GnomeUIInfo entries into normal actions, toggle actions, and radio actions, and then create a separate array of GtkActionEntry structures for each group. This will allow you to create the necessary GtkActionGroup objects. Note that this does not describe the actual "shape" that your menus and toolbars will have; it simply defines the set of commands that will appear in them.
2. Create an XML description of your menus and toolbars for use with GtkUIManager. This defines the actual shape of the menus and toolbars.
3. Port the code that uses gnome-app and gnome-app-helper to GtkAction and GtkUIManager.

Example 1. GnomeUIInfo Example

The following code shows a declaration of a simple menu bar to be used with `gnome_app_fill_menu()` or similar. The menu hierarchy looks like this:

- File

  Open
  —
  Exit
- View

  Zoom In
  Zoom Out
  —
  [ ] Full Screen
  —
  ( ) High Quality

( ) Normal Quality
( ) Low Quality

```
static GnomeUIInfo file_menu_items[] = {
  { GNOME_APP_UI_ITEM, "_Open", "Open a file",
    open_callback, NULL, NULL, GNOME_APP_PIXMAP_STOCK, GTK_STOCK_OPEN,
    'o', GDK_CONTROL_MASK, NULL },
  { GNOME_APP_UI_SEPARATOR },
  { GNOME_APP_UI_ITEM, "E_xit", "Exit the program",
    exit_callback, NULL, NULL, GNOME_APP_PIXMAP_STOCK, GTK_STOCK_QUIT,
    'q', GDK_CONTROL_MASK, NULL},
  { GNOME_APP_UI_ENDOFINFO }
};

static GnomeUIInfo view_radio_items[] = {
  { GNOME_APP_UI_ITEM, "_High Quality", "Display images in high quality, slow mode",
    high_quality_callback, NULL, NULL, GNOME_APP_PIXMAP_NONE, NULL,
    0, 0, NULL },
  { GNOME_APP_UI_ITEM, "_Normal Quality", "Display images in normal quality",
    normal_quality_callback, NULL, NULL, GNOME_APP_PIXMAP_NONE, NULL,
    0, 0, NULL },
  { GNOME_APP_UI_ITEM, "_Low Quality", "Display images in low quality, fast mode",
    low_quality_callback, NULL, NULL, GNOME_APP_PIXMAP_NONE, NULL,
    0, 0, NULL },
  { GNOME_APP_UI_ENDOFINFO }
};

static GnomeUIInfo view_menu_items[] = {
  { GNOME_APP_UI_ITEM, "Zoom _In", "Zoom into the image",
    zoom_in_callback, NULL, NULL, GNOME_APP_PIXMAP_STOCK, GTK_STOCK_ZOOM_IN,
    GDK_PLUS, 0, NULL },
  { GNOME_APP_UI_ITEM, "Zoom _Out", "Zoom away from the image",
    zoom_out_callback, NULL, NULL, GNOME_APP_PIXMAP_STOCK, GTK_STOCK_ZOOM_OUT,
    GDK_MINUS, 0, NULL },
  { GNOME_APP_UI_SEPARATOR },
  { GNOME_APP_UI_TOGGLEITEM, "_Full Screen", "Switch between full screen and windowed
mode",
    full_screen_callback, NULL, NULL, GNOME_APP_PIXMAP_NONE, NULL,
    GDK_F11, 0, NULL },
  { GNOME_APP_UI_SEPARATOR },
  { GNOME_APP_UI_RADIOITEMS, NULL, NULL, view_radio_items },
  { GNOME_APP_UI_ENDOFINFO }
};

static GnomeUIInfo menubar[] = {
  { GNOME_APP_UI_SUBTREE, "_File", NULL, file_menu_items },
  { GNOME_APP_UI_SUBTREE, "_View", NULL, view_menu_items },
  { GNOME_APP_UI_ENDOFINFO }
}
```

Example 2. GtkActionEntry Structures

The following code is the set of actions that are present in the previous example. Note that the toggle and radio entries are separate from normal actions. Also, note that GtkActionEntry structures take key names in the format of gdk_accelerator_parse() rather than key values plus modifiers; you will have to convert these values by hand. For example, `GDK_F11` with no modifiers is equivalent to a key name of `"F11"`. Likewise, `"o"` with `GDK_CONTROL_MASK` is equivalent to `"<ontrol>O"`.

```
/* Normal items */
static GtkActionEntry entries[] = {
  { "FileMenu", NULL, "_File" },
  { "ViewMenu", NULL, "_View" },
  { "Open", GTK_STOCK_OPEN, "_Open", "<control>O", "Open a file",
open_action_callback },
  { "Exit", GTK_STOCK_OPEN, "E_xit", "<control>Q", "Exit the program",
exit_action_callback },
  { "ZoomIn", GTK_STOCK_ZOOM_IN, "Zoom _In", "plus", "Zoom into the image",
zoom_in_action_callback },
  { "ZoomOut", GTK_STOCK_ZOOM_OUT, "Zoom _Out", "minus", "Zoom away from the image",
zoom_out_action_callback },
};

/* Toggle items */
static GtkToggleActionEntry toggle_entries[] = {
  { "FullScreen", NULL, "_Full Screen", "F11", "Switch between full screen and
windowed mode", full_screen_action_callback, FALSE }
};

/* Radio items */
static GtkRadioActionEntry radio_entries[] = {
  { "HighQuality", NULL, "_High Quality", NULL, "Display images in high quality, slow
mode", 0 },
  { "NormalQuality", NULL, "_Normal Quality", NULL, "Display images in normal
quality", 1 },
  { "LowQuality", NULL, "_Low Quality", NULL, "Display images in low quality, fast
mode", 2 }
};
```

Example 3. XML Description

After extracting the actions, you will need to create an XML description of the actual layout of your menus and toolbars for use with GtkUIManager. The following code shows a simple menu bar that corresponds to the previous example. Note that the File and View menus have their names specified in the action entries, not in the XML itself. This is because the XML description only contains *identifiers* for the items in the GUI, rather than human-readable names.

```
static const char *ui_description =
"<ui>"
"  <menubar name='MainMenu'>"
"    <menu action='FileMenu'>"
"      <menuitem action='Open'/>"
```

```
"      <menuitem action='Exit'/>"
"    </menu>"
"    <menu action='ViewMenu'>"
"      <menuitem action='ZoomIn'/>"
"      <menuitem action='ZoomOut'/>"
"      <separator/>"
"      <menuitem action='FullScreen'/>"
"      <separator/>"
"      <menuitem action='HighQuality'/>"
"      <menuitem action='NormalQuality'/>"
"      <menuitem action='LowQuality'/>"
"    </menu>"
"  </menubar>"
"</ui>";
```

Example 4. Creating the Menu Bar

In this last example, we will create a GtkActionGroup based on the action entries we created above. We will then create a GtkUIManager with the XML description of the menu layout. We will also extract the accelerator group and the widgets from the GtkUIManager put them into a window.

```
GtkWidget *window;
GtkWidget *vbox;
GtkWidget *menubar;
GtkActionGroup *action_group;
GtkUIManager *ui_manager;
GtkAccelGroup *accel_group;
GError *error;

window = gtk_window_new (GTK_WINDOW_TOPLEVEL);

vbox = gtk_vbox_new (FALSE, 0);
gtk_container_add (GTK_CONTAINER (window), vbox);

action_group = gtk_action_group_new ("MenuActions");
gtk_action_group_add_actions (action_group, entries, G_N_ELEMENTS (entries), window);
gtk_action_group_add_toggle_actions (action_group, toggle_entries, G_N_ELEMENTS
(toggle_entries), window);
gtk_action_group_add_radio_actions (action_group, radio_entries, G_N_ELEMENTS
(radio_entries), 0, radio_action_callback, window);

ui_manager = gtk_ui_manager_new ();
gtk_ui_manager_insert_action_group (ui_manager, action_group, 0);

accel_group = gtk_ui_manager_get_accel_group (ui_manager);
gtk_window_add_accel_group (GTK_WINDOW (window), accel_group);

error = NULL;
if (!gtk_ui_manager_add_ui_from_string (ui_manager, ui_description, -1, &error))
  {
    g_message ("building menus failed: %s", error->message);
```

```
    g_error_free (error);
    exit (EXIT_FAILURE);
  }

menubar = gtk_ui_manager_get_widget (ui_manager, "/MainMenu");
gtk_box_pack_start (GTK_BOX (vbox), menubar, FALSE, FALSE, 0);

gtk_widget_show_all (window);
```

<< User Interface Manager Object

Migrating from GtkOptionMenu and GtkCombo to GtkComboBox and GtkComboBoxEntry >>

Migrating from GtkOptionMenu and GtkCombo to GtkComboBox and GtkComboBoxEntry

Prior to 2.4, GTK+ offered two widgets for the task of selecting one item from a list of options. GtkOptionMenu presents the list of options as a menu while GtkCombo presents them in a Windows-style list popup. The only difference between the two is that a GtkCombo allows to manually edit the selected value, while the GtkOptionMenu does not.

In GTK+ 2.4, a unified API for list selection was introduced, with GtkComboBox for the non-editable case and GtkComboBoxEntry for the editable case. The selection of the display style — menu or list — is no longer done at the API level, but has been made themeable via the style property GtkComboBox::appearance.

Migrating from GtkOptionMenu to GtkComboBox

Here is an example of a simple, but typical use of GtkOptionMenu:

```
GtkWidget *option_menu, *menu, *menu_item;

option_menu = gtk_option_menu_new ();
menu = gtk_menu_new ();

menu_item = gtk_menu_item_new_with_label ("First Item");
gtk_menu_shell_append (GTK_MENU_SHELL (menu), menu_item);
gtk_widget_show (menu_item);
menu_item = gtk_menu_item_new_with_label ("Second Item");
gtk_menu_shell_append (GTK_MENU_SHELL (menu), menu_item);
gtk_widget_show (menu_item);
menu_item = gtk_menu_item_new_with_label ("Third Item");
gtk_menu_shell_append (GTK_MENU_SHELL (menu), menu_item);
gtk_widget_show (menu_item);

gtk_option_menu_set_menu (GTK_OPTION_MENU (option_menu), menu);
```

In order to react to the user's selection, connect to the "changed" signal on the option menu and use `gtk_option_menu_get_history()` to retrieve the index of the selected item.

And here is how it would be done with a GtkComboBox:

```
GtkWidget *combo_box;

combo_box = gtk_combo_box_new_text ();

gtk_combo_box_append_text (GTK_COMBO_BOX (combo_box), "First Item");
gtk_combo_box_append_text (GTK_COMBO_BOX (combo_box), "Second Item");
gtk_combo_box_append_text (GTK_COMBO_BOX (combo_box), "Third Item");
```

In order to react to the user's selection, connect to the "changed" signal on the combo box and use `gtk_combo_box_get_active()` to retrieve the index of the selected item.

A slightly more complex example involving images:

```
GtkWidget *option_menu, *menu, *menu_item;

option_menu = gtk_option_menu_new ();
menu = gtk_menu_new ();

menu_item = gtk_image_menu_item_new_with_label ("First Item");
gtk_image_menu_item_set_image (gtk_image_new_from_pixbuf (pixbuf1));
gtk_menu_shell_append (GTK_MENU_SHELL (menu), menu_item);
gtk_widget_show (menu_item);
menu_item = gtk_image_menu_item_new_with_label ("Second Item");
gtk_image_menu_item_set_image (gtk_image_new_from_pixbuf (pixbuf2));
gtk_menu_shell_append (GTK_MENU_SHELL (menu), menu_item);
gtk_widget_show (menu_item);
menu_item = gtk_image_menu_item_new_with_label ("Third Item");
gtk_image_menu_item_set_image (gtk_image_new_from_pixbuf (pixbuf3));
gtk_menu_shell_append (GTK_MENU_SHELL (menu), menu_item);
gtk_widget_show (menu_item);

gtk_option_menu_set_menu (GTK_OPTION_MENU (option_menu), menu);
```

can be done using a GtkComboBox as follows:

```
GtkListStore *store;
GtkTreeIter iter;
GtkCellRenderer *renderer;
GtkWidget *combo_box;

store = gtk_list_store_new (2, GDK_TYPE_PIXBUF, G_TYPE_STRING);

gtk_list_store_append (store, &iter);
gtk_list_store_set (store, &iter, 0, pixbuf1, 1, "First Item", -1);
gtk_list_store_append (store, &iter);
gtk_list_store_set (store, &iter, 0, pixbuf2, 1, "Second Item", -1);
gtk_list_store_append (store, &iter);
gtk_list_store_set (store, &iter, 0, pixbuf3, 1, "Third Item", -1);

combo_box = gtk_combo_box_new_with_model (GTK_TREE_MODEL (store));

renderer = gtk_cell_renderer_pixbuf_new ();
gtk_cell_layout_pack_start (GTK_CELL_LAYOUT (combo_box), renderer, FALSE);
gtk_cell_layout_set_attributes (GTK_CELL_LAYOUT (combo_box), renderer,
                                "pixbuf", 0,
                                NULL);

renderer = gtk_cell_renderer_text_new ();
gtk_cell_layout_pack_start (GTK_CELL_LAYOUT (combo_box), renderer, TRUE);
gtk_cell_layout_set_attributes (GTK_CELL_LAYOUT (combo_box), renderer,
                                "text", 1,
                                NULL);
```

<< Migrating from GnomeUIInfo **Migrating from GtkCombo to GtkComboBoxEntry >>**

Migrating from GtkCombo to GtkComboBoxEntry

Here is an example of a simple, but typical use of a GtkCombo:

```
GtkWidget *combo;
GList *items = NULL;

items = g_list_append (items, "First Item");
items = g_list_append (items, "Second Item");
items = g_list_append (items, "Third Item");

combo = gtk_combo_new ();
gtk_combo_set_popdown_strings (GTK_COMBO (combo), items);
```

In order to react to the user's selection, connect to the "changed" signal on the combo and use `gtk_entry_get_text (GTK_ENTRY (combo->entry))` to retrieve the selected text.

And here is how it would be done using GtkComboBoxEntry:

```
combo_box = gtk_combo_box_entry_new_text ();

gtk_combo_box_append_text (GTK_COMBO_BOX (combo_box), "First Item");
gtk_combo_box_append_text (GTK_COMBO_BOX (combo_box), "Second Item");
gtk_combo_box_append_text (GTK_COMBO_BOX (combo_box), "Third Item");
```

In order to react to the user's selection, connect to the "changed" signal on the combo and use `gtk_entry_get_text (GTK_ENTRY (GTK_BIN (combo_box)->child))` to retrieve the selected text.

New features

The new widgets have more to offer than a mere combination of the features of GtkOptionMenu and GtkCombo. Notable new features include:

Grid mode	Sometimes it is preferable to display the available options not in a linear list, but in a grid. A typical example would be a "color combo" where the individual items are small square color swatches. The new widgets support gridded display with the functions `gtk_combo_box_set_wrap_width()`, `gtk_combo_box_set_row_span_column()` and `gtk_combo_box_set_column_span_column()`.
Display of icons	An often-heard complaint about GtkOptionMenu is that the icons which appear in the image menu items in its menu are not displayed in the button showing the selected item. This limitation has been removed in GtkComboBox; the selected item appears in the same way as the options in the popup.
Full tree model power	Since the new widgets are built around the same models that are used for GtkTreeView, all of the powerful machinery of tree models and cell renderers can be used.

GTK+ Tools

<< New features gtk-query-immodules-2.0 >>

gtk-query-immodules-2.0

gtk-query-immodules-2.0 — Input method module registration utility

Synopsis

`gtk-query-immodules-2.0` [module...]

Description

gtk-query-immodules-2.0 collects information about loadable input method modules for GTK+ and writes it to `stdout`.

If called without arguments, it looks for modules in the GTK+ input method module path.

If called with arguments, it looks for the specified modules. The arguments may be absolute or relative paths.

Environment

The environment variable `GTK_PATH` can be used to prepend directories to the input method module path.

Bugs

None known yet.

gtk-update-icon-cache

gtk-update-icon-cache — Icon theme caching utility

Synopsis

`gtk-update-icon-cache` [--force] {iconpath}

Description

gtk-update-icon-cache creates mmap()able cache files for icon themes.

If expects to be given the path to a icon theme directory, e.g. `/usr/share/icons/hicolor`, and writes a `icon-theme.cache` containing cached information about the icons in the directory tree below the given directory.

GTK+ can use the cache files created by **gtk-update-icon-cache** to avoid a lot of system call and disk seek overhead when the application starts. Since the format of the cache files allows them to be mmap()ed shared between multiple applications, the overall memory consumption is reduced as well.

If called with the --force argument, **gtk-update-icon-cache** will overwrite an existing cache file even if it appears to be uptodate.

Bugs

None known yet.

Glossary

allocation

The final size of a *widget* within its *parent*. For example, a widget may request a minimum size of 20×20 pixels, but its parent may decide to allocate 50×20 pixels for it instead.

See Also *requisition* .

bin

A *container* that can hold at most one child widget. The base class for bins is GtkBin.

See Also *container* .

child

A *container's* child is a *widget* contained inside it.

column

GTK+ contains several widgets which display data in columns, e.g. the GtkTreeView. These *view columns* in the tree view are represented by GtkTreeViewColumn objects inside GTK+. They should not be confused with *model columns* which are used to organize the data in tree models.

See Also model-view widget.

container

A *widget* that contains other widgets; in that case, the container is the *parent* of the *child* widgets. Some containers don't draw anything on their own, but rather just organize their children's *geometry*; for example, GtkVBox lays out its children vertically without painting anything on its own. Other containers include decorative elements; for example, GtkFrame contains the frame's child and a label in addition to the shaded frame it draws. The base class for containers is GtkContainer.

See Also *widget geometry* .

display

GDK inherited the concept of display from the X window system, which considers a display to be

the combination of a keyboard, a pointing device and one or more *screens*. Applications open a display to show *windows* and interact with the user. In GDK, a display is represented by a GdkDisplay.

event

Events are the way in which GDK informs GTK+ about external events like pointer motion, button clicks, key presses, etc.

geometry

A *widget's* position and size. Within its parent, this is called the widget's *allocation*.

mapping

This is the step in a *widget's* life cycle where it actually shows the GdkWindows it created when it was *realized*. When a widget is mapped, it must turn on its `GTK_MAPPED` flag.

Note that due to the asynchronous nature of the X window system, a widget's window may not appear on the screen immediatly after one calls `gdk_window_show()`: you must wait for the corresponding map *event* to be received. You can do this with the `GtkWidget::map-event` signal.

model column

A column in a tree model, holding data of a certain type. The types which can be stored in the columns of a model have to be specified when the model is constructed, see e.g. gtk_list_store_new().

See Also *view column* .

model-view widget

These widgets follow the well-known model-view pattern, which separates the data (the model) to be displayed from the component which does the actual visualization (the view). Examples of this pattern in GTK+ are the GtkTreeView/GtkTreeModel and GtkTextView/GtkTextBuffer

One important advantage of this pattern is that it is possible to display the same model in multiple views; another one that the separation of the model allows a great deal of flexibility, as demonstrated by e.g. GtkTreeModelSort or GtkTreeModelFilter.

no-window widget

A widget that does not have a GdkWindow of its own on which to draw its contents, but rather shares its *parent's*. Such a widget has the `GTK_NO_WINDOW` flag set, and can be tested with the `GTK_WIDGET_NO_WINDOW()` macro.

parent

A *widget's* parent is the *container* inside which it resides.

realization

This is the step in a *widget's* life cycle where it creates its own GdkWindow, or otherwise associates itself with its *parent's* GdkWindow. If the widget has its own window, then it must also attach a *style* to it. A widget becomes unrealized by destroying its associated GdkWindow. When a widget is realized, it must turn on its `GTK_REALIZED` flag.

Widgets that don't own the GdkWindow on which they draw are called *no-window widgets*. This can be tested with the `GTK_WIDGET_NO_WINDOW()` macro. Normally, these widgets draw on their parent's GdkWindow.

Note that when a widget creates a window in its `::realize()` handler, it does not actually show the window. That is, the window's structure is just created in memory. The widget actually shows the window when it gets *mapped*.

requisition

The size requisition of a *widget* is the minimum amount of space it requests from its *parent*. Once the parent computes the widget's final size, it gives it its *size allocation*.

See Also *allocation* .

screen

GDK inherited the concept of screen from the X window system, which considers a screen to be a rectangular area, on which applications may place their windows. Screens under X may have quite dissimilar *visuals*. Each screen can stretch across multiple physical monitors.

In GDK, screens are represented by GdkScreen objects.

style

A style encapsulates what GTK+ needs to know in order to draw a widget. Styles can be modified with resource files.

toplevel

A *widget* that does not require a *parent* container. The only toplevel widgets in GTK+ are GtkWindow and widgets derived from it.

See Also *container* .

unmap

See *mapping*.

unrealize

See *realization*.

view column

A displayed column in a tree view, represented by a GtkTreeViewColumn object.

See Also *model column* .

visual

A visual describes how color information is stored in pixels. A *screen* may support multiple visuals. On modern hardware, the most common visuals are truecolor visuals, which store a fixed number of bits (typically 8) for the red, green and blue components of a color.

On ancient hardware, one may still meet indexed visuals, which store color information as an index into a color map, or even monochrome visuals.

widget

A control in a graphical user interface. Widgets can draw themselves and process events from the mouse and keyboard. Widget types include buttons, menus, text entry lines, and lists. Widgets can be arranged into *containers*, and these take care of assigning the *geometry* of the widgets: every widget thus has a parent except those widgets which are *toplevels*. The base class for widgets is GtkWidget.

See Also *container* .

<< gtk-update-icon-cache **Index >>**

Index

G

<< Glossary **Index of deprecated symbols >>**

Index of deprecated symbols

G

<< Index **Index of new symbols in 2.2 >>**

Index of new symbols in 2.2

G

gtk_tree_store_iter_is_valid, gtk_tree_store_iter_is_valid ()
gtk_tree_store_move_after, gtk_tree_store_move_after ()
gtk_tree_store_move_before, gtk_tree_store_move_before ()
gtk_tree_store_reorder, gtk_tree_store_reorder ()
gtk_tree_store_swap, gtk_tree_store_swap ()
gtk_tree_view_column_focus_cell, gtk_tree_view_column_focus_cell ()
gtk_tree_view_expand_to_path, gtk_tree_view_expand_to_path ()
gtk_tree_view_set_cursor_on_cell, gtk_tree_view_set_cursor_on_cell ()
gtk_widget_class_find_style_property, gtk_widget_class_find_style_property ()
gtk_widget_class_list_style_properties, gtk_widget_class_list_style_properties ()
gtk_widget_get_clipboard, gtk_widget_get_clipboard ()
gtk_widget_get_display, gtk_widget_get_display ()
gtk_widget_get_root_window, gtk_widget_get_root_window ()
gtk_widget_get_screen, gtk_widget_get_screen ()
gtk_widget_has_screen, gtk_widget_has_screen ()
gtk_window_fullscreen, gtk_window_fullscreen ()
gtk_window_get_screen, gtk_window_get_screen ()
gtk_window_get_skip_pager_hint, gtk_window_get_skip_pager_hint ()
gtk_window_get_skip_taskbar_hint, gtk_window_get_skip_taskbar_hint ()
gtk_window_set_auto_startup_notification, gtk_window_set_auto_startup_notification ()
gtk_window_set_default_icon_from_file, gtk_window_set_default_icon_from_file ()
gtk_window_set_icon_from_file, gtk_window_set_icon_from_file ()
gtk_window_set_screen, gtk_window_set_screen ()
gtk_window_set_skip_pager_hint, gtk_window_set_skip_pager_hint ()
gtk_window_set_skip_taskbar_hint, gtk_window_set_skip_taskbar_hint ()
gtk_window_unfullscreen, gtk_window_unfullscreen ()

<< Index of deprecated symbols **Index of new symbols in 2.4 >>**

Index of new symbols in 2.4

G

gtk_action_group_remove_action, gtk_action_group_remove_action ()
gtk_action_group_set_sensitive, gtk_action_group_set_sensitive ()
gtk_action_group_set_translate_func, gtk_action_group_set_translate_func ()
gtk_action_group_set_translation_domain, gtk_action_group_set_translation_domain ()
gtk_action_group_set_visible, gtk_action_group_set_visible ()
gtk_action_is_sensitive, gtk_action_is_sensitive ()
gtk_action_is_visible, gtk_action_is_visible ()
gtk_action_new, gtk_action_new ()
gtk_action_set_accel_group, gtk_action_set_accel_group ()
gtk_action_set_accel_path, gtk_action_set_accel_path ()
gtk_action_unblock_activate_from, gtk_action_unblock_activate_from ()
gtk_alignment_get_padding, gtk_alignment_get_padding ()
gtk_alignment_set_padding, gtk_alignment_set_padding ()
gtk_button_box_get_child_secondary, gtk_button_box_get_child_secondary ()
gtk_button_get_alignment, gtk_button_get_alignment ()
gtk_button_get_focus_on_click, gtk_button_get_focus_on_click ()
gtk_button_set_alignment, gtk_button_set_alignment ()
gtk_button_set_focus_on_click, gtk_button_set_focus_on_click ()
gtk_calendar_get_display_options, gtk_calendar_get_display_options ()
gtk_calendar_set_display_options, gtk_calendar_set_display_options ()
gtk_cell_layout_add_attribute, gtk_cell_layout_add_attribute ()
gtk_cell_layout_clear, gtk_cell_layout_clear ()
gtk_cell_layout_clear_attributes, gtk_cell_layout_clear_attributes ()
gtk_cell_layout_pack_end, gtk_cell_layout_pack_end ()
gtk_cell_layout_pack_start, gtk_cell_layout_pack_start ()
gtk_cell_layout_reorder, gtk_cell_layout_reorder ()
gtk_cell_layout_set_attributes, gtk_cell_layout_set_attributes ()
gtk_cell_layout_set_cell_data_func, gtk_cell_layout_set_cell_data_func ()
gtk_cell_renderer_editing_canceled, gtk_cell_renderer_editing_canceled ()
gtk_check_menu_item_get_draw_as_radio, gtk_check_menu_item_get_draw_as_radio ()
gtk_check_menu_item_set_draw_as_radio, gtk_check_menu_item_set_draw_as_radio ()
gtk_clipboard_request_targets, gtk_clipboard_request_targets ()
gtk_clipboard_wait_for_targets, gtk_clipboard_wait_for_targets ()
gtk_color_button_get_alpha, gtk_color_button_get_alpha ()
gtk_color_button_get_color, gtk_color_button_get_color ()
gtk_color_button_get_title, gtk_color_button_get_title ()
gtk_color_button_get_use_alpha, gtk_color_button_get_use_alpha ()
gtk_color_button_new, gtk_color_button_new ()

gtk_entry_completion_set_minimum_key_length, gtk_entry_completion_set_minimum_key_length ()
gtk_entry_completion_set_model, gtk_entry_completion_set_model ()
gtk_entry_completion_set_text_column, gtk_entry_completion_set_text_column ()
gtk_entry_get_alignment, gtk_entry_get_alignment ()
gtk_entry_get_completion, gtk_entry_get_completion ()
gtk_entry_set_alignment, gtk_entry_set_alignment ()
gtk_entry_set_completion, gtk_entry_set_completion ()
gtk_event_box_get_above_child, gtk_event_box_get_above_child ()
gtk_event_box_get_visible_window, gtk_event_box_get_visible_window ()
gtk_event_box_set_above_child, gtk_event_box_set_above_child ()
gtk_event_box_set_visible_window, gtk_event_box_set_visible_window ()
gtk_expander_get_expanded, gtk_expander_get_expanded ()
gtk_expander_get_label, gtk_expander_get_label ()
gtk_expander_get_label_widget, gtk_expander_get_label_widget ()
gtk_expander_get_spacing, gtk_expander_get_spacing ()
gtk_expander_get_use_markup, gtk_expander_get_use_markup ()
gtk_expander_get_use_underline, gtk_expander_get_use_underline ()
gtk_expander_new, gtk_expander_new ()
gtk_expander_new_with_mnemonic, gtk_expander_new_with_mnemonic ()
gtk_expander_set_expanded, gtk_expander_set_expanded ()
gtk_expander_set_label, gtk_expander_set_label ()
gtk_expander_set_label_widget, gtk_expander_set_label_widget ()
gtk_expander_set_spacing, gtk_expander_set_spacing ()
gtk_expander_set_use_markup, gtk_expander_set_use_markup ()
gtk_expander_set_use_underline, gtk_expander_set_use_underline ()
gtk_file_chooser_add_filter, gtk_file_chooser_add_filter ()
gtk_file_chooser_add_shortcut_folder, gtk_file_chooser_add_shortcut_folder ()
gtk_file_chooser_add_shortcut_folder_uri, gtk_file_chooser_add_shortcut_folder_uri ()
gtk_file_chooser_dialog_new, gtk_file_chooser_dialog_new ()
gtk_file_chooser_dialog_new_with_backend, gtk_file_chooser_dialog_new_with_backend ()
gtk_file_chooser_error_quark, gtk_file_chooser_error_quark ()
gtk_file_chooser_get_action, gtk_file_chooser_get_action ()
gtk_file_chooser_get_current_folder, gtk_file_chooser_get_current_folder ()
gtk_file_chooser_get_current_folder_uri, gtk_file_chooser_get_current_folder_uri ()
gtk_file_chooser_get_extra_widget, gtk_file_chooser_get_extra_widget ()
gtk_file_chooser_get_filename, gtk_file_chooser_get_filename ()
gtk_file_chooser_get_filenames, gtk_file_chooser_get_filenames ()
gtk_file_chooser_get_filter, gtk_file_chooser_get_filter ()

gtk_file_chooser_get_local_only, gtk_file_chooser_get_local_only ()
gtk_file_chooser_get_preview_filename, gtk_file_chooser_get_preview_filename ()
gtk_file_chooser_get_preview_uri, gtk_file_chooser_get_preview_uri ()
gtk_file_chooser_get_preview_widget, gtk_file_chooser_get_preview_widget ()
gtk_file_chooser_get_preview_widget_active, gtk_file_chooser_get_preview_widget_active ()
gtk_file_chooser_get_select_multiple, gtk_file_chooser_get_select_multiple ()
gtk_file_chooser_get_uri, gtk_file_chooser_get_uri ()
gtk_file_chooser_get_uris, gtk_file_chooser_get_uris ()
gtk_file_chooser_list_filters, gtk_file_chooser_list_filters ()
gtk_file_chooser_list_shortcut_folders, gtk_file_chooser_list_shortcut_folders ()
gtk_file_chooser_list_shortcut_folder_uris, gtk_file_chooser_list_shortcut_folder_uris ()
gtk_file_chooser_remove_filter, gtk_file_chooser_remove_filter ()
gtk_file_chooser_remove_shortcut_folder, gtk_file_chooser_remove_shortcut_folder ()
gtk_file_chooser_remove_shortcut_folder_uri, gtk_file_chooser_remove_shortcut_folder_uri ()
gtk_file_chooser_select_all, gtk_file_chooser_select_all ()
gtk_file_chooser_select_filename, gtk_file_chooser_select_filename ()
gtk_file_chooser_select_uri, gtk_file_chooser_select_uri ()
gtk_file_chooser_set_action, gtk_file_chooser_set_action ()
gtk_file_chooser_set_current_folder, gtk_file_chooser_set_current_folder ()
gtk_file_chooser_set_current_folder_uri, gtk_file_chooser_set_current_folder_uri ()
gtk_file_chooser_set_current_name, gtk_file_chooser_set_current_name ()
gtk_file_chooser_set_extra_widget, gtk_file_chooser_set_extra_widget ()
gtk_file_chooser_set_filename, gtk_file_chooser_set_filename ()
gtk_file_chooser_set_filter, gtk_file_chooser_set_filter ()
gtk_file_chooser_set_local_only, gtk_file_chooser_set_local_only ()
gtk_file_chooser_set_preview_widget, gtk_file_chooser_set_preview_widget ()
gtk_file_chooser_set_preview_widget_active, gtk_file_chooser_set_preview_widget_active ()
gtk_file_chooser_set_select_multiple, gtk_file_chooser_set_select_multiple ()
gtk_file_chooser_set_uri, gtk_file_chooser_set_uri ()
gtk_file_chooser_set_use_preview_label, gtk_file_chooser_set_use_preview_label ()
gtk_file_chooser_unselect_all, gtk_file_chooser_unselect_all ()
gtk_file_chooser_unselect_filename, gtk_file_chooser_unselect_filename ()
gtk_file_chooser_unselect_uri, gtk_file_chooser_unselect_uri ()
gtk_file_chooser_widget_new, gtk_file_chooser_widget_new ()
gtk_file_chooser_widget_new_with_backend, gtk_file_chooser_widget_new_with_backend ()
gtk_file_filter_add_custom, gtk_file_filter_add_custom ()
gtk_file_filter_add_mime_type, gtk_file_filter_add_mime_type ()
gtk_file_filter_add_pattern, gtk_file_filter_add_pattern ()

<< Index of new symbols in 2.2 **Index of new symbols in 2.6 >>**

Index of new symbols in 2.6

G

gtk_alternative_dialog_button_order, gtk_alternative_dialog_button_order ()
gtk_cell_renderer_combo_new, gtk_cell_renderer_combo_new ()
gtk_cell_renderer_progress_new, gtk_cell_renderer_progress_new ()
gtk_cell_view_get_cell_renderers, gtk_cell_view_get_cell_renderers ()
gtk_cell_view_get_size_of_row, gtk_cell_view_get_size_of_row ()
gtk_cell_view_new, gtk_cell_view_new ()
gtk_cell_view_new_with_markup, gtk_cell_view_new_with_markup ()
gtk_cell_view_new_with_pixbuf, gtk_cell_view_new_with_pixbuf ()
gtk_cell_view_new_with_text, gtk_cell_view_new_with_text ()
gtk_cell_view_set_background_color, gtk_cell_view_set_background_color ()
gtk_cell_view_set_displayed_row, gtk_cell_view_set_displayed_row ()
gtk_cell_view_set_model, gtk_cell_view_set_model ()
gtk_cell_view_set_value, gtk_cell_view_set_value ()
gtk_cell_view_set_values, gtk_cell_view_set_values ()
gtk_clipboard_set_can_store, gtk_clipboard_set_can_store ()
gtk_clipboard_store, gtk_clipboard_store ()
gtk_clipboard_wait_is_target_available, gtk_clipboard_wait_is_target_available ()
gtk_combo_box_get_active_text, gtk_combo_box_get_active_text ()
gtk_combo_box_get_column_span_column, gtk_combo_box_get_column_span_column ()
gtk_combo_box_get_focus_on_click, gtk_combo_box_get_focus_on_click ()
gtk_combo_box_get_popup_accessible, gtk_combo_box_get_popup_accessible ()
gtk_combo_box_get_row_separator_func, gtk_combo_box_get_row_separator_func ()
gtk_combo_box_get_row_span_column, gtk_combo_box_get_row_span_column ()
gtk_combo_box_get_wrap_width, gtk_combo_box_get_wrap_width ()
gtk_combo_box_set_add_tearoffs, gtk_combo_box_set_add_tearoffs ()
gtk_combo_box_set_focus_on_click, gtk_combo_box_set_focus_on_click ()
gtk_combo_box_set_row_separator_func, gtk_combo_box_set_row_separator_func ()
gtk_dialog_set_alternative_button_order, gtk_dialog_set_alternative_button_order ()
gtk_drag_dest_add_image_targets, gtk_drag_dest_add_image_targets ()
gtk_drag_dest_add_text_targets, gtk_drag_dest_add_text_targets ()
gtk_drag_dest_add_uri_targets, gtk_drag_dest_add_uri_targets ()
gtk_drag_source_add_text_targets, gtk_drag_source_add_text_targets ()
gtk_entry_completion_get_inline_completion, gtk_entry_completion_get_inline_completion ()
gtk_entry_completion_get_popup_completion, gtk_entry_completion_get_popup_completion ()
gtk_entry_completion_get_text_column, gtk_entry_completion_get_text_column ()
gtk_entry_completion_insert_prefix, gtk_entry_completion_insert_prefix ()
gtk_entry_completion_set_inline_completion, gtk_entry_completion_set_inline_completion ()
gtk_entry_completion_set_popup_completion, gtk_entry_completion_set_popup_completion ()

gtk_tree_view_get_hover_expand, gtk_tree_view_get_hover_expand ()
gtk_tree_view_get_hover_selection, gtk_tree_view_get_hover_selection ()
gtk_tree_view_get_row_separator_func, gtk_tree_view_get_row_separator_func ()
gtk_tree_view_set_fixed_height_mode, gtk_tree_view_set_fixed_height_mode ()
gtk_tree_view_set_hover_expand, gtk_tree_view_set_hover_expand ()
gtk_tree_view_set_hover_selection, gtk_tree_view_set_hover_selection ()
gtk_tree_view_set_row_separator_func, gtk_tree_view_set_row_separator_func ()
gtk_window_get_focus_on_map, gtk_window_get_focus_on_map ()
gtk_window_get_icon_name, gtk_window_get_icon_name ()
gtk_window_set_default_icon_name, gtk_window_set_default_icon_name ()
gtk_window_set_focus_on_map, gtk_window_set_focus_on_map ()
gtk_window_set_icon_name, gtk_window_set_icon_name ()

<< Index of new symbols in 2.4