Part VII. GTK+ Platform Support: GTK+ 3 Reference Manual

Table of Contents

broadwayd

broadwayd — Broadway display server

Synopsis

`broadwayd` [--port `PORT`] [--address `ADDRESS`] [--unixsocket `ADDRESS`] [`:DISPLAY`]

Description

broadwayd is a display server for the Broadway GDK backend. It allows multiple GTK+ applications to display their windows in the same web browser, by connecting to broadwayd.

When using broadwayd, specify the display number to use, prefixed with a colon, similar to X. The default display number is 0.

```
broadwayd :5
```

Then point your web browser at `http://127.0.0.1:8085`. Start your applications like this:

```
GDK_BACKEND=broadway BROADWAY_DISPLAY=:5 gtk3-demo
```

You can add password protection for your session by creating a file in `$XDG_CONFIG_HOME/broadway.passwd` or `$HOME/.config/broadway.passwd` with a crypt(3) style password hash. A simple way to generate it is with openssl:

```
openssl passwd -1  > ~/.config/broadway.passwd
```

Options

--port	Use `PORT` as the HTTP port, instead of the default 8080 + (`DISPLAY` - 1).
--address	Use `ADDRESS` as the HTTP address, instead of the default `http://127.0.0.1:PORT`.
--unixsocket	Use `ADDRESS` as the unix domain socket address. This option overrides `--address` and `--port`. It is available only on Unix-like systems.

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 with 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 you 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.bz2` or `tar.xz` file which you unpack into a directory full of the source files as follows:

```
tar xvfj gtk+-3.2.0.tar.bz2
tar xvfJ gtk+-3.2.0.tar.xz
```

In the toplevel 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 files 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 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**.)

Some of the libraries that GTK+ depends on are maintained by by the GTK+ team: GLib, GdkPixbuf, Pango, ATK and GObject Introspection. 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 or here.
- The GdkPixbuf library provides facilities for loading images in a variety of file formats. It is available here.
- Pango is a library for internationalized text handling. It is available here.
- 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 here.
- Gobject Introspection is a framework for making introspection data available to language bindings. It is available here.

External dependencies

- 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 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.
- Cairo is a graphics library that supports vector graphics and image compositing. Both Pango and GTK+ use cairo for all of their drawing.
- libepoxy is a library that abstracts the differences between different OpenGL libraries. GTK+ uses it for cross-platform GL support.
- The Wayland libraries are needed to build GTK+ with the Wayland backend.
- The shared-mime-info package is not a hard dependency of GTK+, but it contains definitions for mime types that are used by GIO and, indirectly, by GTK+. gdk-pixbuf will use GIO for mime type detection if possible. For this to work, shared-mime-info needs to be installed and `XDG_DATA_DIRS` set accordingly at configure time. Otherwise, gdk-pixbuf falls back to its built-in mime type detection.

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 **gtk3-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-immodules=MODULE1,MODULE2,...]]
[--enable-debug=[no/minimum/yes]]
[--disable-Bsymbolic | --enable-Bsymbolic]
[--disable-xkb | --enable-xkb]
[--disable-xinerama | --enable-xinerama]
[--disable-gtk-doc | --enable-gtk-doc]
[--disable-cups | --enable-cups]
[--disable-papi | --enable-papi]
[--enable-xinput | --disable-xinput]
[--enable-packagekit | --disable-packagekit]
[--enable-x11-backend | --disable-x11-backend]
[--enable-win32-backend | --disable-win32-backend]
[--enable-quartz-backend | --disable-quartz-backend]
[--enable-broadway-backend | --disable-broadway-backend]
[--enable-wayland-backend | --disable-wayland-backend]
[--enable-mir-backend | --disable-mir-backend]
[--enable-introspection=[no/auto/yes]]
[--enable-installed-tests | --disable-installed-tests]

`--disable-modules` and `--enable-modules`. Normally GTK+ will try to build the input method modules 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-immodules`. This option allows you to specify which input method modules you want to include directly into the GTK+ shared library, as opposed to building them as loadable modules.

`--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-Bsymbolic` and `--enable-Bsymbolic`. The option `--disable-Bsymbolic` turns off the use of the -Bsymbolic-functions linker flag. This is only necessary if you want to override GTK+ functions by using `LD_PRELOAD`.

`--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-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-xinerama` and `--enable-xinerama`. By default the **configure** script will try to link against the Xinerama libraries if they are found. These options can be used to explicitly control whether Xinerama should be used.

`--disable-xinput` and `--enable-xinput`. Controls whether GTK+ is built with support for the XInput or XInput2 extension. These extensions provide an extended interface to 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.

`--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.

`--disable-cups` and `--enable-cups`. By default the **configure** script will try to build the cups print backend if the cups libraries are found. These options can be used to explicitly control whether the cups print backend should be built.

`--disable-papi` and `--enable-papi`. By default the **configure** script will try to build the papi print backend if the papi libraries are found. These options can be used to explicitly control whether the papi print backend should be built.

`--disable-packagekit` and `--enable-packagekit`. By default the **configure** script will try to build the PackageKit support for the open-with dialog if the PackageKit libraries are found. These options can be used to explicitly control whether PackageKit support should be built.

`--enable-x11-backend`, `--disable-x11-backend`, `--enable-win32-backend`, `--disable-win32-backend`, `--enable-quartz-backend`, `--disable-quartz-backend`, `--enable-broadway-backend`, `--disable-broadway-backend`, `--enable-wayland-backend`, `--disable-wayland-backend` `--enable-mir-backend`, and `--disable-mir-backend`. Enables specific backends for GDK. If none of these options are given, the x11 backend will be enabled by default, unless the platform is Windows, in which case the default is win32. If any backend is explicitly enabled or disabled, no other platform will be enabled automatically. Other supported backends are the quartz backend for OS X.

`--enable-introspection`. Build with or without introspection support. The default is 'auto'.

`--enable-installed-tests` or `--disable-installed-tests`. Whether to install tests on the system. If enabled, tests and their data are installed in `${libexecdir}/gtk+/installed-tests`. Metadata for the tests is installed in `${prefix}/share/installed-tests/gtk+`. To run the installed tests, gnome-desktop-testing-runner can be used.

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 (the actual output on your system may be different):

```
$ pkg-config --cflags gtk+-3.0
 -pthread -I/usr/include/gtk-3.0 -I/usr/lib64/gtk-3.0/include -I/usr/include/atk-1.0
-I/usr/include/cairo -I/usr/include/pango-1.0 -I/usr/include/glib-2.0 -I/usr/lib64/glib-
2.0/include -I/usr/include/pixman-1 -I/usr/include/freetype2 -I/usr/include/libpng12
$ pkg-config --libs gtk+-3.0
 -pthread -lgtk-3 -lgdk-3 -latk-1.0 -lgio-2.0 -lpangoft2-1.0 -lgdk_pixbuf-2.0 -
lpangocairo-1.0 -lcairo -lpango-1.0 -lfreetype -lfontconfig -lgobject-2.0 -lgmodule-2.0 -
lgthread-2.0 -lrt -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 gtk+-3.0` hello.c -o hello `pkg-config --libs gtk+-3.0`
```

Deprecated GTK+ functions are annotated to make the compiler emit warnings when they are used (e.g. with gcc, you need to use the -Wdeprecated-declarations option). If these warnings are problematic, they can be turned off by defining the preprocessor symbol `GDK_DISABLE_DEPRECATION_WARNINGS` by using the commandline option `-DGDK_DISABLE_DEPRECATION_WARNINGS`

GTK+ deprecation annotations are versioned; by defining the macros `GDK_VERSION_MIN_REQUIRED` and `GDK_VERSION_MAX_ALLOWED`, you can specify the range of GTK+ versions whose API you want to use. APIs that were deprecated before or introduced after this range will trigger compiler warnings.

Here is how you would compile hello.c if you want to allow it to use symbols that were not deprecated in 3.2:

```
$ cc `pkg-config --cflags gtk+-3.0` -DGDK_VERSION_MIN_REQIRED=GDK_VERSION_3_2 hello.c -o
hello `pkg-config --libs gtk+-3.0`
```

And here is how you would compile hello.c if you don't want it to use any symbols that were introduced after 3.4:

```
$ cc `pkg-config --cflags gtk+-3.0` -DGDK_VERSION_MAX_ALLOWED=GDK_VERSION_3_4 hello.c -o
hello `pkg-config --libs gtk+-3.0`
```

The older deprecation mechanism of hiding deprecated interfaces entirely from the compiler by using the preprocessor symbol GTK_DISABLE_DEPRECATED is still used for deprecated macros, enumeration values, etc. To detect uses of these in your code, use the commandline option `-DGTK_DISABLE_DEPRECATED`. There are similar symbols GDK_DISABLE_DEPRECATED, GDK_PIXBUF_DISABLE_DEPRECATED and G_DISABLE_DEPRECATED for GDK, GdkPixbuf and GLib.

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

Useful autotools macros

GTK+ provides various macros for easily checking version and backends supported. The macros are

AM_PATH_GTK_3_0([minimum-version], [if-found], [if-not-found], [modules])

This macro should be used to check that GTK+ is installed and available for compilation. The four arguments are optional, and they are: *minimum-version*, the minimum version of GTK+ required for compilation; *if-found*, the action to perform if a valid version of GTK+ has been found; *if-not-found*, the action to perform if a valid version of GTK+ has not been found; *modules*, a list of modules to be checked along with GTK+.

GTK_CHECK_BACKEND([backend-name], [minimum-version], [if-found], [if-not-found])

This macro should be used to check if a specific backend is supported by GTK+. The *minimum-version*, *if-found* and *if-not-found* arguments are optional.

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 `GTK3_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 not available if GTK+ has been configured with `--enable-debug=no`.

`--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 Broadway GDK backends use some additional environment variables.

`GTK_DEBUG`**.** Unless GTK+ has been configured with `--enable-debug=no`, this variable can be set to a list of debug options, which cause GTK+ to print out different types of debugging information.

actions	Actions and menu models
baselines	Show baselines
builder	GtkBuilder support
geometry	Size allocation
icontheme	Icon themes
interactive	Open the interactive debugger
keybindings	Keybindings
misc	Miscellaneous information
modules	Loading of modules
no-css-cache	Bypass caching for CSS style properties
no-pixel-cache	Disable the pixel cache
plugsocket	Cross-process embedding
pixel-cache	Pixel cache
printing	Printing support
size-request	Size requests
text	Text widget internals
touchscreen	Pretend the pointer is a touchscreen device
tree	Tree widget internals
updates	Visual feedback about window updates
resize	Highlight resizing widgets
layout	Show layout borders

The special value `all` can be used to turn on all debug options. The special value `help` can be used to obtain a list of all supported debug options.

`GTK3_MODULES`**.** A list of modules to load. Note that GTK+ also allows to specify modules to load via a commandline option (`--gtk-module`) and with the `gtk-modules` setting.

`GTK_MODULES`**.** A list of modules to load in addition to the ones in the `GTK3_MODULES` variable.

Note that this environment variable is read by GTK+ 2.x too, which may not have the same set of modules available for loading. Use `GTK3_MODULES` for modules that are only compatible with GTK+ 3.

`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, input method modules, file system backends and print backends. 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-3.0` in the user's home directory, followed by the system default directory, which is `libdir`/gtk-3.0/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+-3.0` to determine this from a script), `host` is the architecture on which GTK+ was built. (use `pkg-config --variable=gtk_host gtk+-3.0` to determine this from a script), and `type` is a directory specific to the type of modules; currently it can be `modules`, `engines`, `immodules`, `filesystems` or `printbackends`, corresponding to the types of modules mentioned 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.

Note that this environment variable is read by GTK+ 2.x too, which makes it unsuitable for setting it system-wide (or session-wide), since doing so will cause either GTK+ 2.x applications or GTK+ 3 applications to see incompatible modules.

`GTK_IM_MODULE`**.** Specifies an IM module to use in preference to the one determined from the locale. If this isn't set and you are running on the system that enables `XSETTINGS` and has a value in `Gtk/IMModule`, that will be used for the default IM module. This also can be a colon-separated list of input-methods, which GTK+ will try in turn until it finds one available on the system.

`GTK_IM_MODULE_FILE`**.** Specifies the file listing the IM modules to load. This environment variable the default value `libdir`/gtk-3.0/3.0.0/immodules.cache (`libdir` has the same meaning here as explained for `GTK_PATH`). The `immodules.cache` file is generated by the **gtk-query-immodules-3.0** utility.

Note that this environment variable is read by GTK+ 2.x too, which makes it unsuitable for setting it system-wide (or session-wide), since doing so will cause either GTK+ 2.x applications or GTK+ 3 applications to see the wrong list of IM modules.

`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.

`GTK_THEME`**.** If set, makes GTK+ use the named theme instead of the theme that is specified by the gtk-theme-name setting. This is intended mainly for easy debugging of theme issues. It is also possible to specify a theme variant to load, by appending the variant name with a colon, like this: `GTK_THEME=Adwaita:dark`.

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 `libdir`/gtk-3.0/3.0.0/loaders.cache (`libdir` is the sysconfdir specified when GTK+ was configured, usually `/usr/local/lib`.) The `loaders.cache` file is generated by the **gdk-pixbuf-query-loaders** utility.

`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.

cursor	Information about cursor objects (only win32)
dnd	Information about drag-and-drop
draw	Information about drawing operations (only win32)
eventloop	Information about event loop operation (mostly Quartz)
misc	Miscellaneous information
nogl	Turn off OpenGL. GDK will behave as if OpenGL support was not available.
nograbs	Turn off all pointer and keyboard grabs
xinerama	Simulate a multi-monitor setup
xim	Information about XIM support

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

`GDK_RENDERING`**.** If set, selects the way how GDK creates similar surfaces. This affects both the functionality of the function gdk_window_create_similar_surface() as well as the way GDK creates backing surfaces for double buffering. The following values can be used:

similar	Create similar surfaces to the window in use. This is the default behavior when the variable is not set.
image	Always create image surfaces. This essentially turns off all hardware acceleration inside GTK.
recording	Always create recording surfaces. This causes bare rendering to the backend without the creation of intermediate surfaces (Pixmaps in X) and will likely cause flicker.

All other values will be ignored and fall back to the default behavior. More values might be added in the future.

`GDK_BACKEND`**.** If set, selects the GDK backend to use. Selecting a backend requires that GTK+ is compiled with support for that backend. The following backends can be selected, provided they are included in the GDK libraries you are using:

quartz	Selects the native Quartz backend
win32	Selects the native backend for Microsoft Windows
x11	Selects the native backend for connecting to X11 servers.
broadway	Selects the Broadway backend for display in web browsers
wayland	Selects the Wayland backend for connecting to Wayland display servers
mir	Selects the Mir backend for connecting to Mir display servers

Since 3.10, this environment variable can contain a comma-separated list of backend names, which are tried in order. The list may also contain a *, which means: try all remaining backends. The special value "help" can be used to make GDK print out a list of all available backends. For more information about selecting backends, see the gdk_display_manager_get() function.

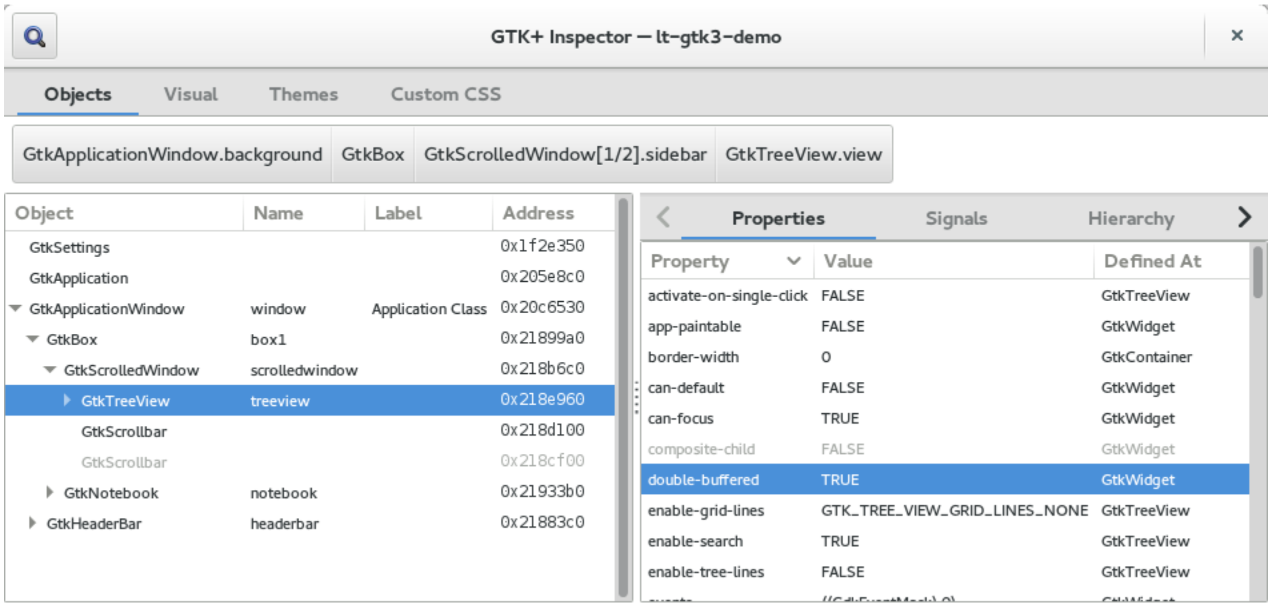
`GTK_CSD`**.** The default value of this environment variable is 1. If changed to 0, this disables the default use of client-side decorations on GTK+ windows, thus making the window manager responsible for drawing the decorations of windows that do not have a custom titlebar widget. CSD is always used for windows with a custom titlebar widget set, as the WM should not draw another titlebar or other decorations around the custom one.

`GTK_OVERLAY_SCROLLING`. The default value of this environment variable is 1, which means that each instance of GtkScrolledWindow will choose whether to use overlay or full- size scrollbars via its own GtkScrolledWindow:overlay-scrolling property, which defaults to `TRUE`. If this variable is set to 0, all GtkScrolledWindow instances are made to use full/non-overlay scrollbars.

`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.

`DESKTOP_STARTUP_ID`. GTK+ uses this environment variable to provide startup notification according to the Startup Notification Spec. Following the specification, GTK+ unsets this variable after reading it (to keep it from leaking to child processes). So, if you need its value for your own purposes, you have to read it before calling gtk_init().

Interactive debugging


GTK+ Inspector — lt-gtk3-demo
Objects
Visual
Themes
Custom CSS
GtkApplicationWindow.background
GtkBox
GtkScrolledWindow[1/2].sidebar
GtkTreeView.view
Properties
Signals
Hierarchy

Object	Name	Label	Address
GtkSettings			0x1f2e350
GtkApplication			0x205e8c0
GtkApplicationWindow	window	Application Class	0x20c6530
GtkBox	box1		0x21899a0
GtkScrolledWindow	scrolledwindow		0x218b6c0
GtkTreeView	treeview		0x218e960
GtkScrollbar			0x218d100
GtkScrollbar			0x218cf00
GtkNotebook	notebook		0x21933b0
GtkHeaderBar	headerbar		0x21883c0

Property	Value	Defined At
activate-on-single-click	FALSE	GtkTreeView
app-paintable	FALSE	GtkWidget
border-width	0	GtkContainer
can-default	FALSE	GtkWidget
can-focus	TRUE	GtkWidget
composite-child	FALSE	GtkWidget
double-buffered	TRUE	GtkWidget
enable-grid-lines	GTK_TREE_VIEW_GRID_LINES_NONE	GtkTreeView
enable-search	TRUE	GtkTreeView
enable-tree-lines	FALSE	GtkTreeView

GTK+ includes an interactive debugger, called the GTK+ Inspector, which lets you explore the widget tree of any GTK+ application at runtime, as well as tweak the theme and trigger visual debugging aids. You can easily try out changes at runtime before putting them into the code.

Note that the GTK+ inspector can only show GTK+ internals. It can not understand the application-specific logic of a GTK+ application. Also, the fact that the GTK+ inspector is running in the application process limits what it can do. It is meant as a complement to full-blown debuggers and system tracing facilities such as DTrace, not as a replacement.

To enable the GTK+ inspector, you can use the Control-Shift-I or Control-Shift-D keyboard shortcuts, or set the `GTK_DEBUG=interactive` environment variable.

In some situations, it may be inappropriate to give users access to the GTK+ inspector. The keyboard shortcuts can be disabled with the \`enable-inspector-keybinding\` key in the \`org.gtk.Settings.Debug\` GSettings schema.

Using GTK+ on the X Window System

Using GTK+ on the X Window System — X11-specific 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.

GTK+ includes an cross-process embedding facility in the form of the #GtkSocket and #GtkPlug widgets. These are X11-specific, and you have to include the `gtk/gtkx.h` header to use them.

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.

X11-specific environment variables

The X11 GDK backend can be influenced with some additional environment variables.

`GDK_SYNCHRONIZE`**.** If set, GDK makes all X requests synchronously. This is a useful option for debugging, but it will slow down the performance considerably.

`GDK_CORE_DEVICE_EVENTS`**.** If set, GDK makes does not use the XInput extension, and only reacts to core X input events.

`GDK_SCALE`**.** Must be set to an integer, typically 2. If set, GDK will scale all windows by the specified factor. Scaled output is meant to be used on high-dpi displays. Normally, GDK will pick up a suitable scale factor for each monitor from the display system. This environment variable allows to override that.

`GDK_DPI_SCALE`**.** This can be useful when using scale-aware GTK+ applications together with scale-unaware applications on a high-dpi display. In that case, the font resolution can be doubled to make scale-unaware applications readable, and `GDK_DPI_SCALE=0.5` can be set to compensate for that in GTK+ applications which are already scaled by setting `GDK_SCALE=2`.

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 Windows

The Windows port of GTK+ is an implementation of GDK (and therefore GTK+) on top of the Win32 API. When compiling GTK+ on Windows, this backend is the default.

Windows-specific commandline options

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

`--sync`**.** Don't batch GDI requests. This might be a marginally useful option for debugging.

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

`--use-wintab`**.** Use the Wintab API for tablet support. This is the default.

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

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.

`GDK_USE_WINTAB`**.** If this variable is set, GTK+ uses the Wintab API for tablet support. This is the default.

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

Windows-specific handling of cursors

By default the "system" cursor theme is used. This makes GTK prefer cursors that Windows currently uses, falling back to Adwaita cursors and (as the last resort) built-in X cursors.

When any other cursor theme is used, GTK will prefer cursors from that theme, falling back to Windows cursors and built-in X cursors.

Theme can be changed by setting `gtk-cursor-theme-name` GTK+ setting. Users can override GTK+ settings in the `settings.ini` file or at runtime in the GTK+ Inspector.

Themes are loaded from normal Windows variants of the XDG locations: `%HOME%/icons/THEME/cursors`, `%APPDATA%/icons/THEME/cursors`, `RUNTIME_PREFIX/share/icons/THEME/cursors`.

The `gtk-cursor-theme-size` setting is ignored, GTK will use the cursor size that Windows tells it to use.

More information about GTK+ on Windows, including detailed build instructions, binary downloads, etc, can be found online.

Using GTK+ on Mac OS X

The Mac OS X port of GTK+ is an implementation of GDK (and therefore GTK+) on top of the Quarz API. Currently, the Mac OS X port does not use any additional commandline options or environment variables.

For up-to-date information about the current status of this port, see the project page.

Using GTK+ with Broadway

Using GTK+ with Broadway — HTML-specific aspects of using GTK+

Using GTK+ with Broadway

The GDK Broadway backend provides support for displaying GTK+ applications in a web browser, using HTML5 and web sockets. To run your application in this way, select the Broadway backend by setting `GDK_BACKEND=broadway`. Then you can make your application appear in a web browser by pointing it at `http://127.0.0.1:8080`. Note that you need to enable web sockets in your web browser.

You can choose a different port from the default 8080 by setting the `BROADWAY_DISPLAY` environment variable to the port that you want to use.

It is also possible to use multiple GTK+ applications in the same web browser window, by using the Broadway server, **broadwayd**, that ships with GTK+. To use broadwayd, start it like this:

```
broadwayd :5
```

Then point your web browser at `http://127.0.0.1:8085`. Start your applications like this:

```
GDK_BACKEND=broadway BROADWAY_DISPLAY=:5 gtk3-demo
```

Broadway-specific environment variables

`BROADWAY_DISPLAY`**.** Specifies the Broadway display number. The default display is 0. The display number determines the port to use when connecting to a Broadway application via the following formula:

```
        port = 8080 + display
```

Using GTK+ with Wayland

Using GTK+ with Wayland — Wayland-specific aspects of using GTK+

Using GTK+ with Wayland

The GDK Wayland backend provides support for running GTK+ applications under the Wayland display server. To run your application in this way, select the Wayland backend by setting `GDK_BACKEND=wayland`.

Currently, the Wayland backend does not use any additional commandline options or environment variables.

For up-to-date information about the current status of this backend, see the project page.

Using GTK+ with Mir

Using GTK+ with Mir — Mir-specific aspects of using GTK+

Using GTK+ with Mir

The GDK Mir backend provides support for running GTK+ applications under Mir based display servers. To run your application in this way, select the Mir backend by setting `GDK_BACKEND=mir`.

Currently, the Mir backend does not use any additional commandline options or environment variables.