GLib Reference Manual

for GLib 2.5.7

Table of Contents

GLib Overview
- Compiling the GLib package - How to compile GLib itself
- Cross-compiling the GLib package - How to cross-compile GLib
- Compiling GLib Applications - How to compile your GLib application
- Running GLib Applications - How to run and debug your GLib application
- Changes to GLib - Incompatible changes made between successing versions of GLib
- Mailing lists and bug reports - Getting help with GLib

GLib Fundamentals
- Version Information - Variables and functions to check the GLib version
- Basic Types - standard GLib types, defined for ease-of-use and portability.
- Limits of Basic Types - portable method of determining the limits of the standard types.
- Standard Macros - commonly-used macros.
- Type Conversion Macros - portably storing integers in pointer variables.
- Byte Order Macros - a portable way to convert between different byte orders.
- Numerical Definitions - mathematical constants, and floating point decomposition.
- Miscellaneous Macros - specialized macros which are not used often.
- Atomic Operations - basic atomic integer and pointer operations

GLib Core Application Support
- The Main Event Loop - manages all available sources of events.
- Threads - thread abstraction; including threads, different mutexes, conditions and thread private data.
- Thread Pools - pools of threads to execute work concurrently.
- Asynchronous Queues - asynchronous communication between threads.
- Dynamic Loading of Modules - portable method for dynamically loading 'plug-ins'.
- Memory Allocation - general memory-handling.
- IO Channels - portable support for using files, pipes and sockets.
- Error Reporting - a system for reporting errors.
- Message Output and Debugging Functions - functions to output messages and help debug applications.
- Message Logging - versatile support for logging messages with different levels of importance.

GLib Utilities
- String Utility Functions - various string-related functions.
- Character Set Conversion - convert strings between different character sets using iconv().
- Unicode Manipulation - functions operating on Unicode characters and UTF-8 strings.
- Internationalization - gettext support macros.
- Date and Time Functions - calendrical calculations and miscellaneous time stuff.
- Random Numbers - pseudo-random number generator.
- Hook Functions - support for manipulating lists of hook functions.
- Miscellaneous Utility Functions - a selection of portable utility functions.
- Lexical Scanner - a general purpose lexical scanner.
- Automatic String Completion - support for automatic completion using a group of target strings.
- Timers - keep track of elapsed time.
- Spawning Processes - process launching with fork()/exec().
- File Utilities - various file-related functions.

- Shell-related Utilities - shell-like commandline handling.
- Commandline option parser - parses commandline options
- Glob-style pattern matching - matches strings against patterns containing '*' (wildcard) and '?' (joker).
- Simple XML Subset Parser - parses a subset of XML.
- Key-value file parser - parses .ini-like config files
- Windows Compatibility Functions - UNIX emulation on Windows.

GLib Data Types
- Memory Chunks - efficient way to allocate groups of equal-sized chunks of memory.
- Doubly-Linked Lists - linked lists containing integer values or pointers to data, with the ability to iterate over the list in both directions.
- Singly-Linked Lists - linked lists containing integer values or pointers to data, limited to iterating over the list in one direction.
- Double-ended Queues - double-ended queue data structure.
- Trash Stacks - maintain a stack of unused allocated memory chunks.
- Hash Tables - associations between keys and values so that given a key the value can be found quickly.
- Strings - text buffers which grow automatically as text is added.
- String Chunks - efficient storage of groups of strings.
- Arrays - arrays of arbitrary elements which grow automatically as elements are added.
- Pointer Arrays - arrays of pointers to any type of data, which grow automatically as new elements are added.
- Byte Arrays - arrays of bytes, which grow automatically as elements are added.
- Balanced Binary Trees - a sorted collection of key/value pairs optimized for searching and traversing in order.
- N-ary Trees - trees of data with any number of branches.
- Quarks - a 2-way association between a string and a unique integer identifier.
- Keyed Data Lists - lists of data elements which are accessible by a string or GQuark identifier.
- Datasets - associate groups of data elements with particular memory locations.
- Relations and Tuples - tables of data which can be indexed on any number of fields.
- Caches - caches allow sharing of complex data structures to save resources.
- Memory Allocators - allocates chunks of memory for GList, GSList and GNode.

GLib Tools
- glib-gettextize - gettext internationalization utility

Index
Index of deprecated symbols
Index of new symbols in 2.2
Index of new symbols in 2.4
Index of new symbols in 2.6

GLib Overview >>

GLib Reference Manual

GLib Overview

Compiling the GLib package - How to compile GLib itself
Cross-compiling the GLib package - How to cross-compile GLib
Compiling GLib Applications - How to compile your GLib application
Running GLib Applications - How to run and debug your GLib application
Changes to GLib - Incompatible changes made between successing versions of GLib
Mailing lists and bug reports - Getting help with GLib

GLib is a general-purpose utility library, which provides many useful data types, macros, type conversions, string utilities, file utilities, a main loop abstraction, and so on. It works on many UNIX-like platforms, Windows, OS/2 and BeOS. GLib is released under the GNU Library General Public License (GNU LGPL).

The general policy of GLib is that all functions are invisibly threadsafe with the exception of data structure manipulation functions, where, if you have two threads manipulating the *same* data structure, they must use a lock to synchronize their operation.

<< GLib Reference Manual Compiling the GLib package >>

Compiling the GLib package

Compiling the GLib Package — How to compile GLib itself

Building the Library on UNIX

On UNIX, GLib 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. The normal sequence for compiling and installing the GLib library is thus:

```
./configure
make
make install
```

The standard options provided by GNU autoconf may be passed to the **configure** script. Please see the autoconf documentation or run **./configure --help** for information about the standard options.

The GTK+ documentation contains further details about the build process and ways to influence it.

Dependencies

Before you can compile the GLib library, 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 GLib 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 GLib library. (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 GLib 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**.)

GLib depends on a number of other libraries.

- 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()`, however many older systems lack an `iconv()` implementation. On such systems, you must install the libiconv library. This can be found at: http://www.gnu.org/software/libiconv.

  If your system has an `iconv()` implementation but you want to use libiconv instead, you can pass the --with-libiconv option to configure. This forces libiconv to be used.

  Note that if you have libiconv installed in your default include search path (for instance, in `/usr/local/`), but don't enable it, you will get an error while compiling GLib because the `iconv.h` that libiconv installs hides the system iconv.

  If you are using the native iconv implementation on Solaris instead of libiconv, you'll need to make sure that you have the converters between locale encodings and UTF-8 installed. At a minimum you'll need the SUNWuiu8 package. You probably should also install the SUNWciu8, SUNWhiu8, SUNWjiu8, and SUNWkiu8 packages.

  The native iconv on Compaq Tru64 doesn't contain support for UTF-8, so you'll need to use GNU libiconv instead. (When using GNU libiconv for GLib, you'll need to use GNU libiconv for GNU gettext as well.) This probably applies to related operating systems as well.

- The libintl library from the GNU gettext package is needed if your system doesn't have the `gettext()` functionality for handling message translation databases.

- A thread implementation is needed, unless you want to compile GLib without thread support, which is not recommended. The thread support in GLib can be based upon several native thread implementations, e.g. POSIX threads, DCE threads or Solaris threads.

Extra Configuration Options

In addition to the normal options, the **configure** script in the GLib library supports these additional arguments:

`configure` [[--enable-debug=[no|minimum|yes]]] [[--disable-gc-friendly] | [--enable-gc-friendly]] [[--disable-mem-pools] | [--enable-mem-pools]] [[--disable-threads] | [--enable-threads]] [[--with-threads=[none|posix|dce|solaris|win32]]] [[--disable-included-printf] | [--enable-included-printf]] [[--disable-gtk-doc] | [--enable-gtk-doc]]

`--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 GLib.

`--disable-gc-friendly` and `--enable-gc-friendly`. When enabled all memory freed by the application, but retained by GLib for performance reasons is set to zero, thus making deployed garbage collection or memory profiling tools detect unlinked memory correctly. This will make GLib slightly slower and is thus disabled by default.

`--disable-mem-pools` and `--enable-mem-pools`. Many small chunks of memory are often allocated via collective pools in GLib and are cached after release to speed up reallocations. For sparse memory systems this behaviour is often inferior, so memory pools can be disabled to avoid excessive caching and force atomic maintenance of chunks through the `g_malloc()` and `g_free()` functions. Code currently affected by this:

- GList, GSList, GNode, GHash allocations

- GMemChunks become basically non-effective

- GSignal disables all caching (potentially very slow)
- GType doesn't honour the GTypeInfo *n_preallocs* field anymore
- the GBSearchArray flag `G_BSEARCH_ALIGN_POWER2` becomes non-functional

`--disable-threads` and `--enable-threads`. Do not compile GLib to be multi thread safe. GLib will be slightly faster then. This is however not recommended, as many programs rely on GLib being multi thread safe.

`--with-threads`. Specify a thread implementation to use.

- 'posix' and 'dce' can be used interchangeable to mean the different versions of Posix threads. configure tries to find out, which one is installed.
- 'solaris' uses the native Solaris thread implementation.
- 'none' means that GLib will be thread safe, but does not have a default thread implementation. This has to be supplied to `g_thread_init()` by the programmer.

`--disable-included-printf` and `--enable-included-printf`. By default the **configure** script will try to auto-detect whether the C library provides a suitable set of `printf()` functions. In detail, **configure** checks that the semantics of `snprintf()` are as specified by C99 and that positional parameters as specified in the Single Unix Specification are supported. If this not the case, GLib will include an implementation of the `printf()` family. These options can be used to explicitly control whether an implementation fo the `printf()` family should be included or not.

`--disable-gtk-doc` and `--enable-gtk-doc`. By default the **configure** script will try to auto-detect whether the gtk-doc package is installed. If it is, then it will use it to extract and build the documentation for the GLib library. These options can be used to explicitly control whether gtk-doc should be used or not. If it is not used, the distributed, pre-generated HTML files will be installed instead of building them on your machine.

<< GLib Overview **Cross-compiling the GLib package >>**

GLib Reference Manual

Cross-compiling the GLib package

Cross-compiling the GLib Package — How to cross-compile GLib

Building the Library for a different architecture

Cross-compilation is the process of compiling a program or library on a different architecture or operating system then it will be run upon. GLib is slightly more difficult to cross-compile than many packages because much of GLib is about hiding differences between different systems.

These notes cover things specific to cross-compiling GLib; for general information about cross-compilation, see the autoconf info pages.

GLib tries to detect as much information as possible about the target system by compiling and linking programs without actually running anything; however, some information GLib needs is not available this way. This information needs to be provided to the configure script via a "cache file" or by setting the cache variables in your environment.

As an example of using a cache file, to cross compile for the "MingW32" Win32 runtine environment on a Linux system, create a file 'win32.cache' with the following contents:

```
glib_cv_long_long_format=ll
glib_cv_stack_grows=no
```

Then execute the following commands:

```
PATH=/path/to/mingw32-compiler/bin:$PATH
chmod a-w win32.cache   # prevent configure from changing it
./configure --cache-file=win32.cache --host=mingw32
```

The complete list of cache file variables follows. Most of these won't need to be set in most cases.

Cache file variables

glib_cv_long_long_format=[ll/q/I64]. Format used by `printf()` and `scanf()` for 64 bit integers. "ll" is the C99 standard, and what is used by the 'trio' library that GLib builds if your `printf()` is insufficiently capable. Doesn't need to be set if you are compiling using trio.

glib_cv_stack_grows=[yes/no]. Whether the stack grows up or down. Most places will want "no", A few architectures, such as PA-RISC need "yes".

glib_cv_working_bcopy=[yes/no]. Whether your `bcopy()` can handle overlapping copies. Only needs to be set if you don't have `memmove()`. (Very unlikely)

glib_cv_sane_realloc=[yes/np]. Whether your `realloc()` conforms to ANSI C and can handle `NULL` as the first argument. Defaults to "yes" and probably doesn't need to be set.

glib_cv_have_strlcpy=[yes/no]. Whether you have `strlcpy()` that matches OpenBSD. Defaults to "no", which is safe, since GLib uses a built-in version in that case.

glib_cv_va_val_copy=[yes/no]. Whether va_list can be copied as a pointer. If set to "no", then `memcopy()` will be used. Only matters if you don't have `va_copy()` or `__va_copy()`. (So, doesn't matter for GCC.) Defaults to "yes" which is slightly more common than "no".

glib_cv_rtldglobal_broken=[yes/no]. Whether you have a bug found in OSF/1 v5.0. Defaults to "no".

glib_cv_uscore=[yes/no]. Whether an underscore needs to be prepended to symbols when looking them up via `dlsym()`. Only needs to be set if your system uses `dlopen()`/`dlsym()`.

ac_cv_func_posix_getpwuid_r=[yes/no]. Whether you have a getpwuid_r function (in your C library, not your thread library) that conforms to the POSIX spec. (Takes a 'struct passwd **' as the final argument)

ac_cv_func_nonposix_getpwuid_r=[yes/no]. Whether you have some variant of `getpwuid_r()` that doesn't conform to to the POSIX spec, but GLib might be able to use (or might segfault.) Only needs to be set if `ac_cv_func_posix_getpwuid_r` is not set. It's safest to set this to "no".

glib_cv_use_pid_surrogate=[yes/no]. Whether to use a `setpriority()` on the PID of the thread as a method for setting the priority of threads. This only needs to be set when using POSIX threads.

ac_cv_func_printf_unix98=[yes/no]. Whether your `printf()` family supports Unix98 style `%N$` positional parameters. Defaults to "no".

ac_cv_func_vsnprintf_c99=[yes/no]. Whether you have a `vsnprintf()` with C99 semantics. (C99 semantics means returning the number of bytes that would have been written had the output buffer had enough space.) Defaults to "no".

<< Compiling the GLib package Compiling GLib Applications >>

Compiling GLib Applications

Compiling GLib Applications — How to compile your GLib application

Compiling GLib Applications on UNIX

To compile a GLib application, you need to tell the compiler where to find the GLib 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 glib-2.0
 -I/usr/include/glib-2.0 -I/usr/lib/glib-2.0/include
$ pkg-config --libs glib-2.0
 -L/usr/lib -lm -lglib-1.3
```

If your application uses modules, threads or GObject features, it must be compiled and linked with the options returned by the following pkg-config invokations:

```
$ pkg-config --cflags --libs gmodule-2.0
$ pkg-config --cflags --libs gthread-2.0
$ pkg-config --cflags --libs gobject-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 GLib Hello, World, you would type the following:

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

<< Cross-compiling the GLib package **Running GLib Applications >>**

GLib Reference Manual

Running GLib Applications

Running GLib Applications — How to run and debug your GLib application

Running and debugging GLib Applications

Environment variables

GLib inspects a few of environment variables in addition to standard variables like `LANG`, `PATH` or `HOME`.

`G_FILENAME_ENCODING`. This environment variable can be set to a comma-separated list of character set names. GLib assumes that filenames are encoded in the first character set from that list rather than in UTF-8. The special token "@locale" can be used to specify the character set for the current locale.

`G_BROKEN_FILENAMES`. If this environment variable is set, GLib assumes that filenames are in the locale encoding rather than in UTF-8. G_FILENAME_ENCODING takes priority over G_BROKEN_FILENAMES.

`G_MESSAGES_PREFIXED`. A list of log levels for which messages should be prefixed by the program name and PID of the application. The default is to prefix everything except `G_LOG_LEVEL_MESSAGE` and `G_LOG_LEVEL_INFO`.

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

fatal_warnings Causes GLib to abort the program at the first call to g_warning(). This option is special in that it doesn't require GLib to be configured with debugging support.

`G_RANDOM_VERSION`. If this environment variable is set to '2.0', the outdated pseudo-random number seeding and generation algorithms from GLib-2.0 are used instead of the new better ones. Use the GLib-2.0 algorithms only if you have sequences of numbers generated with Glib-2.0 that you need to reproduce exactly.

`LIBCHARSET_ALIAS_DIR`. Allows to specify a nonstandard location for the `charset.aliases` file that is used by the character set conversion routines. The default location is the *`libdir`* specified at compilation time.

`G_WIN32_PRETEND_WIN9X`. Setting this variable to any value forces g_win32_get_windows_version () to return a version code for Windows 9x. This is mainly an internal debugging aid for GTK+ and GLib developers, to be able to check the code paths for Windows 9x.

Traps and traces

Some code portions contain trap variables that can be set during debugging time if GLib has been configured with `--enable-debug=yes`. Such traps lead to immediate code halts to examine the current program state and backtrace.

Currently, the following trap variables exist:

```
static volatile gulong g_trap_free_size;
static volatile gulong g_trap_realloc_size;
static volatile gulong g_trap_malloc_size;
```

If set to a size > 0, g_free(), g_realloc() and g_malloc() will be intercepted if the size matches the size of the corresponding memory block. This will only work with `g_mem_set_vtable (glib_mem_profiler_table)` upon startup though, because memory profiling is required to match on the memory block sizes.

<< Compiling GLib Applications **Changes to GLib >>**

GLib Reference Manual

Changes to GLib

Changes to GLib — Incompatible changes made between successing versions of GLib

Incompatible changes from 2.0 to 2.2

- GLib changed the seeding algorithm for the pseudo-random number generator Mersenne Twister, as used by GRand and GRandom. This was necessary, because some seeds would yield very bad pseudo-random streams. Also the pseudo-random integers generated by `g_rand*_int_range()` will have a slightly better equal distribution with the new version of GLib.

  Further information can be found at the website of the Mersenne Twister random number generator at http://www.math.keio.ac.jp/~matumoto/emt.html.

  The original seeding and generation algorithms, as found in GLib 2.0.x, can be used instead of the new ones by setting the environment variable `G_RANDOM_VERSION` to the value of '2.0'. Use the GLib-2.0 algorithms only if you have sequences of numbers generated with Glib-2.0 that you need to reproduce exactly.

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 section on GLib.

- The event loop functionality GMain has extensively been revised to support multiple separate main loops in separate threads. All sources (timeouts, idle functions, etc.) are associated with a GMainContext.

  Compatibility functions exist so that most application code dealing with the main loop will continue to work. However, code that creates new custom types of sources will require modification.

  The main changes here are:

  - Sources are now exposed as GSource *, rather than simply as numeric ids.
  - New types of sources are created by structure "derivation" from GSource, so the `source_data` parameter to the GSource virtual functions has been replaced with a GSource *.
  - Sources are first created, then later added to a specific GMainContext.
  - Dispatching has been modified so both the callback and data are passed in to the `dispatch()` virtual function.

  To go along with this change, the vtable for GIOChannel has changed and `add_watch()` has been replaced by `create_watch()`.

- `g_list_foreach()` and `g_slist_foreach()` have been changed so they are now safe against removal of the current item, not the next item.

  It's not recommended to mutate the list in the callback to these functions in any case.

- GDate now works in UTF-8, not in the current locale. If you want to use it with the encoding of the locale, you need to convert strings using `g_locale_to_utf8()` first.

- `g_strsplit()` has been fixed to:

  - include trailing empty tokens, rather than stripping them
  - split into a maximum of `max_tokens` tokens, rather than `max_tokens + 1`

  Code depending on either of these bugs will need to be fixed.

- Deprecated functions that got removed: `g_set_error_handler()`, `g_set_warning_handler()`, `g_set_message_handler()`, use `g_log_set_handler()` instead.

<< Running GLib Applications **Mailing lists and bug reports >>**

Mailing lists and bug reports

Mailing lists and bug reports — Getting help with GLib

Filing a bug report or feature request

If you encounter a bug, misfeature, or missing feature in GLib, please file a bug report on http://bugzilla.gnome.org. We'd also appreciate reports of incomplete or misleading information in the GLib documentation; file those against the "docs" component of the "glib" 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 GLib development in Bugzilla, so it's the way to be sure the GLib developers won't forget about an issue.

Submitting Patches

If you develop a bugfix or enhancement for GLib, 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 generally takes place on these lists. You can subscribe or view the archives of these lists on http://mail.gnome.org.

gtk-list@gnome.org	gtk-list covers general GTK+ (and GLib) topics; questions about using GLib in programs, GLib from a user standpoint, announcements of GLib-related projects would all be on-topic. The bulk of the traffic consists of GTK+ programming questions.
gtk-devel-list@gnome.org	gtk-devel-list is for discussion of work on GTK+ (and GLib) itself, it is *not* for asking questions about how to use GTK+ (or GLib) in applications. gtk-devel-list is appropriate for discussion of patches, bugs, proposed features, and so on.
gtk-doc-list@gnome.org	gtk-doc-list is for discussion of the gtk-doc documentation system (used to document GTK+ and Glib), and for work on the GTK+ (and GLib) documentation.

<< Changes to GLib **GLib Fundamentals >>**

GLib Fundamentals

Version Information - Variables and functions to check the GLib version
Basic Types - standard GLib types, defined for ease-of-use and portability.
Limits of Basic Types - portable method of determining the limits of the standard types.
Standard Macros - commonly-used macros.
Type Conversion Macros - portably storing integers in pointer variables.
Byte Order Macros - a portable way to convert between different byte orders.
Numerical Definitions - mathematical constants, and floating point decomposition.
Miscellaneous Macros - specialized macros which are not used often.
Atomic Operations - basic atomic integer and pointer operations

<< Mailing lists and bug reports **Version Information >>**

Version Information

Version Information — Variables and functions to check the GLib version

Synopsis

```
#include <glib.h>


extern      const guint glib_major_version;
extern      const guint glib_minor_version;
extern      const guint glib_micro_version;
extern      const guint glib_binary_age;
extern      const guint glib_interface_age;
const gchar* glib_check_version             (guint required_major,
                                             guint required_minor,
                                             guint required_micro);

#define     GLIB_MAJOR_VERSION
#define     GLIB_MINOR_VERSION
#define     GLIB_MICRO_VERSION
#define     GLIB_CHECK_VERSION              (major,minor,micro)
```

Description

GLib 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

glib_major_version

```
extern const guint glib_major_version;
```

The major version number of the GLib library. (e.g. in GLib version 1.2.5 this is 1.)

This variable is in the library, so represents the GLib library you have linked against. Contrast with the GLIB_MAJOR_VERSION macro, which represents the major version of the GLib headers you have included.

---

glib_minor_version

```
extern const guint glib_minor_version;
```


The minor version number of the GLib library. (e.g. in GLib version 1.2.5 this is 2.)

This variable is in the library, so represents the GLib library you have linked against. Contrast with the GLIB_MINOR_VERSION macro, which represents the minor version of the GLib headers you have included.

---

glib_micro_version

```
extern const guint glib_micro_version;
```

The micro version number of the GLib library. (e.g. in GLib version 1.2.5 this is 5.)

This variable is in the library, so represents the GLib library you have linked against. Contrast with the GLIB_MICRO_VERSION macro, which represents the micro version of the GLib headers you have included.

---

glib_binary_age

```
extern const guint glib_binary_age;
```

This is the binary age passed to libtool. If libtool means nothing to you, don't worry about it. ;-)

---

glib_interface_age

```
extern const guint glib_interface_age;
```

This is the interface age passed to libtool. If libtool means nothing to you, don't worry about it. ;-)

---

glib_check_version ()

```
const gchar* glib_check_version             (guint required_major,
                                             guint required_minor,
                                             guint required_micro);
```

Checks that the GLib library in use is compatible with the given version. Generally you would pass in the constants GLIB_MAJOR_VERSION, GLIB_MINOR_VERSION, GLIB_MICRO_VERSION as the three arguments to this function; that produces a check that the library in use is compatible with the version of GLib 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.)

required_major : the required major version.
required_minor : the required major version.
required_micro : the required major version.
Returns : `NULL` if the GLib library is compatible with the given version, or a string describing the version mismatch. The returned string is owned by GLib and must not be modified or freed.

Since 2.6

GLIB_MAJOR_VERSION

```
#define GLIB_MAJOR_VERSION 2
```

The major version number of the GLib library. Like glib_major_version, but from the headers used at application compile time, rather than from the library linked against at application run time.

GLIB_MINOR_VERSION

```
#define GLIB_MINOR_VERSION 5
```

The minor version number of the GLib library. Like gtk_minor_version, but from the headers used at application compile time, rather than from the library linked against at application run time.

GLIB_MICRO_VERSION

```
#define GLIB_MICRO_VERSION 7
```

The micro version number of the GLib library. Like gtk_micro_version, but from the headers used at application compile time, rather than from the library linked against at application run time.

GLIB_CHECK_VERSION()

```
#define     GLIB_CHECK_VERSION(major,minor,micro)
```

Checks the version of the GLib library. It returns `TRUE` if the GLib library is the same or newer than the given version.

Example 1. Checking the version of the GLib library


```
if (!GLIB_CHECK_VERSION (1, 2, 0))
  g_error ("GLib version 1.2.0 or above is needed");
```

major : the major version number.
minor : the minor version number.
micro : the micro version number.

<< GLib Fundamentals **Basic Types >>**

Basic Types

Basic Types — standard GLib types, defined for ease-of-use and portability.

Synopsis

```
#include <glib.h>


typedef     gboolean;
typedef     gpointer;
typedef     gconstpointer;
typedef     gchar;
typedef     guchar;

typedef     gint;
typedef     guint;
typedef     gshort;
typedef     gushort;
typedef     glong;
typedef     gulong;

typedef     gint8;
typedef     guint8;
typedef     gint16;
typedef     guint16;
typedef     gint32;
typedef     guint32;

#define     G_HAVE_GINT64
typedef     gint64;
typedef     guint64;
#define     G_GINT64_CONSTANT               (val)

typedef     gfloat;
typedef     gdouble;

typedef     gsize;
typedef     gssize;
```

Description

GLib defines a number of commonly used types, which can be divided into 4 groups:

- New types which are not part of standard C - gboolean, gsize, gssize.
- Integer types which are guaranteed to be the same size across all platforms - gint8, guint8, gint16, guint16, gint32, guint32, gint64, guint64.
- Types which are easier to use than their standard C counterparts - gpointer, gconstpointer, guchar, guint, gushort, gulong.
- Types which correspond exactly to standard C types, but are included for completeness - gchar, gint, gshort, glong, gfloat, gdouble.

Details

gboolean

```
typedef gint   gboolean;
```

A standard boolean type. Variables of this type should only contain the value TRUE or FALSE.

---

gpointer

```
typedef void* gpointer;
```

An untyped pointer. gpointer looks better and is easier to use than void*.

---

gconstpointer

```
typedef const void *gconstpointer;
```

An untyped pointer to constant data. The data pointed to should not be changed.

This is typically used in function prototypes to indicate that the data pointed to will not be altered by the function.

---

gchar

```
typedef char   gchar;
```

Corresponds to the standard C char type.

---

guchar

```
typedef unsigned char   guchar;
```

Corresponds to the standard C unsigned char type.

---

gint

```
typedef int    gint;
```

Corresponds to the standard C int type. Values of this type can range from G_MININT to G_MAXINT.

---

guint

```
typedef unsigned int    guint;
```

Corresponds to the standard C unsigned int type. Values of this type can range from 0 to G_MAXUINT.

---

gshort

```
typedef short  gshort;
```

Corresponds to the standard C short type. Values of this type can range from G_MINSHORT to G_MAXSHORT.

---

gushort

```
typedef unsigned short  gushort;
```

Corresponds to the standard C unsigned short type. Values of this type can range from 0 to G_MAXUSHORT.

---

glong

```
typedef long   glong;
```

Corresponds to the standard C long type. Values of this type can range from G_MINLONG to G_MAXLONG.

---

gulong

```
typedef unsigned long   gulong;
```


Corresponds to the standard C unsigned long type. Values of this type can range from 0 to G_MAXULONG.

---

gint8

```
typedef signed char gint8;
```

A signed integer guaranteed to be 8 bits on all platforms. Values of this type can range from -128 to 127.

---

guint8

```
typedef unsigned char guint8;
```

An unsigned integer guaranteed to be 8 bits on all platforms. Values of this type can range from 0 to 255.

---

gint16

```
typedef signed short gint16;
```

A signed integer guaranteed to be 16 bits on all platforms. Values of this type can range from -32,768 to 32,767.

---

guint16

```
typedef unsigned short guint16;
```

An unsigned integer guaranteed to be 16 bits on all platforms. Values of this type can range from 0 to 65,535.

---

gint32

```
typedef signed int gint32;
```

A signed integer guaranteed to be 32 bits on all platforms. Values of this type can range from -2,147,483,648 to 2,147,483,647.

---

guint32

```
typedef unsigned int guint32;
```

An unsigned integer guaranteed to be 32 bits on all platforms. Values of this type can range from 0 to 4,294,967,295.

G_HAVE_GINT64

```
#define G_HAVE_GINT64 1          /* deprecated, always true */
```

This macro is defined if 64-bit signed and unsigned integers are available on the platform.

---

gint64

```
G_GNUC_EXTENSION typedef signed long long gint64;
```

A signed integer guaranteed to be 64 bits on all platforms on which it is available (see G_HAVE_GINT64). Values of this type can range from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807.

---

guint64

```
G_GNUC_EXTENSION typedef unsigned long long guint64;
```

An unsigned integer guaranteed to be 64 bits on all platforms on which it is available (see G_HAVE_GINT64). Values of this type can range from 0 to 18,446,744,073,709,551,615.

G_GINT64_CONSTANT()

```
#define G_GINT64_CONSTANT(val)  (G_GNUC_EXTENSION (val##LL))
```

This macro is used to insert 64-bit integer literals into the source code.

val : a literal integer value, e.g. 0x1d636b02300a7aa7U.

---

gfloat

```
typedef float   gfloat;
```


Corresponds to the standard C float type. Values of this type can range from -G_MAXFLOAT to G_MAXFLOAT.

---

gdouble

```
typedef double  gdouble;
```

Corresponds to the standard C double type. Values of this type can range from -G_MAXDOUBLE to G_MAXDOUBLE.

---

gsize

```
typedef unsigned int gsize;
```

An unsigned 32-bit integer intended to represent sizes of data structures.

---

gssize

```
typedef signed int gssize;
```

A signed 32-bit integer intended to represent sizes of data structures.

<< Version Information **Limits of Basic Types >>**

Limits of Basic Types

Limits of Basic Types — portable method of determining the limits of the standard types.

Synopsis

```
#include <glib.h>


#define     G_MININT
#define     G_MAXINT
#define     G_MAXUINT

#define     G_MINSHORT
#define     G_MAXSHORT
#define     G_MAXUSHORT

#define     G_MINLONG
#define     G_MAXLONG
#define     G_MAXULONG

#define     G_MININT8
#define     G_MAXINT8
#define     G_MAXUINT8

#define     G_MININT16
#define     G_MAXINT16
#define     G_MAXUINT16

#define     G_MININT32
#define     G_MAXINT32
#define     G_MAXUINT32

#define     G_MININT64
#define     G_MAXINT64
#define     G_MAXUINT64

#define     G_MAXSIZE

#define     G_MINFLOAT
#define     G_MAXFLOAT

#define     G_MINDOUBLE
#define     G_MAXDOUBLE
```

Description

These macros provide a portable method to determine the limits of some of the standard integer and floating point types.

Details

G_MININT

```
#define G_MININT        INT_MIN
```

The minimum value which can be held in a gint.

G_MAXINT

```
#define G_MAXINT        INT_MAX
```

The maximum value which can be held in a gint.

G_MAXUINT

```
#define G_MAXUINT       UINT_MAX
```

The maximum value which can be held in a guint.

G_MINSHORT

```
#define G_MINSHORT      SHRT_MIN
```

The minimum value which can be held in a gshort.

G_MAXSHORT

```
#define G_MAXSHORT      SHRT_MAX
```

The maximum value which can be held in a gshort.

G_MAXUSHORT

```
#define G_MAXUSHORT     USHRT_MAX
```

The maximum value which can be held in a gushort.

G_MINLONG

```
#define G_MINLONG       LONG_MIN
```

The minimum value which can be held in a glong.

G_MAXLONG

```
#define G_MAXLONG       LONG_MAX
```

The maximum value which can be held in a glong.

G_MAXULONG

```
#define G_MAXULONG      ULONG_MAX
```

The maximum value which can be held in a gulong.

G_MININT8

```
#define G_MININT8       ((gint8)  0x80)
```

The minimum value which can be held in a gint8.

Since 2.4

G_MAXINT8

```
#define G_MAXINT8       ((gint8)  0x7f)
```

The maximum value which can be held in a gint8.

Since 2.4

G_MAXUINT8

```
#define G_MAXUINT8      ((guint8) 0xff)
```


The maximum value which can be held in a guint8.

Since 2.4

G_MININT16

```
#define G_MININT16      ((gint16)  0x8000)
```

The minimum value which can be held in a gint16.

Since 2.4

G_MAXINT16

```
#define G_MAXINT16      ((gint16)  0x7fff)
```

The maximum value which can be held in a gint16.

Since 2.4

G_MAXUINT16

```
#define G_MAXUINT16     ((guint16) 0xffff)
```

The maximum value which can be held in a guint16.

Since 2.4

G_MININT32

```
#define G_MININT32      ((gint32)  0x80000000)
```

The minimum value which can be held in a gint32.

Since 2.4

G_MAXINT32

```
#define G_MAXINT32      ((gint32)  0x7fffffff)
```

The maximum value which can be held in a gint32.

Since 2.4

G_MAXUINT32

```
#define G_MAXUINT32     ((guint32) 0xffffffff)
```

The maximum value which can be held in a guint32.

Since 2.4

G_MININT64

```
#define G_MININT64      G_GINT64_CONSTANT(0x8000000000000000)
```

The minimum value which can be held in a gint64.

G_MAXINT64

```
#define G_MAXINT64      G_GINT64_CONSTANT(0x7fffffffffffffff)
```

The maximum value which can be held in a gint64.

G_MAXUINT64

```
#define G_MAXUINT64     G_GINT64_CONSTANT(0xffffffffffffffffU)
```

The maximum value which can be held in a guint64.

G_MAXSIZE

```
#define G_MAXSIZE       G_MAXUINT
```

The maximum value which can be held in a gsize.

Since 2.4

G_MINFLOAT

```
#define G_MINFLOAT      FLT_MIN
```

The minimum positive value which can be held in a gfloat.

If you are interested in the smallest value which can be held in a gfloat, use -G_MAX_FLOAT.

G_MAXFLOAT

```
#define G_MAXFLOAT      FLT_MAX
```

The maximum value which can be held in a gfloat.

G_MINDOUBLE

```
#define G_MINDOUBLE     DBL_MIN
```

The minimum positive value which can be held in a gdouble.

If you are interested in the smallest value which can be held in a gdouble, use -G_MAXDOUBLE.

G_MAXDOUBLE

```
#define G_MAXDOUBLE     DBL_MAX
```

The maximum value which can be held in a gdouble.

<< Basic Types **Standard Macros >>**

Standard Macros

Standard Macros — commonly-used macros.

Synopsis

```
#include <glib.h>


#define     G_OS_WIN32
#define     G_OS_BEOS
#define     G_OS_UNIX

#define     G_DIR_SEPARATOR
#define     G_DIR_SEPARATOR_S
#define     G_IS_DIR_SEPARATOR              (c)
#define     G_SEARCHPATH_SEPARATOR
#define     G_SEARCHPATH_SEPARATOR_S

#define     TRUE
#define     FALSE

#define     NULL

#define     MIN                             (a, b)
#define     MAX                             (a, b)

#define     ABS                             (a)
#define     CLAMP                           (x, low, high)

#define     G_STRUCT_MEMBER                 (member_type, struct_p, struct_offse
#define     G_STRUCT_MEMBER_P               (struct_p, struct_offset)
#define     G_STRUCT_OFFSET                 (struct_type, member)

#define     G_MEM_ALIGN

#define     G_CONST_RETURN
```

Description

These macros provide a few commonly-used features.

Details

G_OS_WIN32

```
#define G_OS_WIN32
```

This macro is defined only on Windows. So you can bracket Windows-specific code in "#ifdef G_OS_WIN32".

G_OS_BEOS

```
#define G_OS_BEOS
```

This macro is defined only on BeOS. So you can bracket BeOS-specific code in "#ifdef G_OS_BEOS".

G_OS_UNIX

```
#define G_OS_UNIX
```

This macro is defined only on UNIX. So you can bracket UNIX-specific code in "#ifdef G_OS_UNIX".

G_DIR_SEPARATOR

```
#define     G_DIR_SEPARATOR
```

The directory separator character. This is '/' on UNIX machines and '\' under Windows.

G_DIR_SEPARATOR_S

```
#define     G_DIR_SEPARATOR_S
```

The directory separator as a string. This is "/" on UNIX machines and "\" under Windows.

G_IS_DIR_SEPARATOR()

```
#define     G_IS_DIR_SEPARATOR(c)
```

Checks whether a character is a directory separator. It returns `TRUE` for '/' on UNIX machines and for '\' or '/' under Windows.

c : a character

Since 2.6

G_SEARCHPATH_SEPARATOR

```
#define     G_SEARCHPATH_SEPARATOR
```

The search path separator character. This is ':' on UNIX machines and ';' under Windows.

G_SEARCHPATH_SEPARATOR_S

```
#define     G_SEARCHPATH_SEPARATOR_S
```

The search path separator as a string. This is ":" on UNIX machines and ";" under Windows.

TRUE

```
#define TRUE    (!FALSE)
```

Defines the `TRUE` value for the gboolean type.

FALSE

```
#define FALSE   (0)
```

Defines the `FALSE` value for the gboolean type.

NULL

```
#define     NULL
```

Defines the standard `NULL` pointer.

MIN()

```
#define MIN(a, b)  (((a) < (b)) ? (a) : (b))
```

Calculates the minimum of *a* and *b*.

a : a numeric value.
b : a numeric value.
Returns : the minimum of *a* and *b*.

MAX()

```
#define MAX(a, b)  (((a) > (b)) ? (a) : (b))
```

Calculates the maximum of *a* and *b*.

a : a numeric value.
b : a numeric value.
Returns : the maximum of *a* and *b*.

ABS()

```
#define ABS(a)     (((a) < 0) ? -(a) : (a))
```

Calculates the absolute value of *a*. The absolute value is simply the number with any negative sign taken away.

For example,

- ABS(-10) is 10.
- ABS(10) is also 10.

a : a numeric value.
Returns : the absolute value of *a*.

CLAMP()

```
#define CLAMP(x, low, high)  (((x) > (high)) ? (high) : (((x) < (low)) ? (low) :
```

Ensures that *x* is between the limits set by *low* and *high*.

For example,

- CLAMP(5, 10, 15) is 10.
- CLAMP(15, 5, 10) is 10.
- CLAMP(20, 15, 25) is 20.

x : the value to clamp.
low : the minimum value allowed.
high : the maximum value allowed.
Returns : the value of *x* clamped to the range between *low* and *high*.

G_STRUCT_MEMBER()

```
#define     G_STRUCT_MEMBER(member_type, struct_p, struct_offset)
```

Returns a member of a structure at a given offset, using the given type.

member_type : the type of the struct field.
struct_p : a pointer to a struct.
struct_offset : the offset of the field from the start of the struct, in bytes.
Returns : the struct member.

G_STRUCT_MEMBER_P()

```
#define     G_STRUCT_MEMBER_P(struct_p, struct_offset)
```

Returns an untyped pointer to a given offset of a struct.

struct_p : a pointer to a struct.
struct_offset : the offset from the start of the struct, in bytes.
Returns : an untyped pointer to *struct_p* plus *struct_offset* bytes.

G_STRUCT_OFFSET()

```
#define     G_STRUCT_OFFSET(struct_type, member)
```

Returns the offset, in bytes, of a member of a struct.

struct_type : a structure type, e.g. GtkWidget.
member : a field in the structure, e.g. *window*.
Returns : the offset of *member* from the start of *struct_type*.

G_MEM_ALIGN

```
#define     G_MEM_ALIGN
```

Indicates the number of bytes to which memory will be aligned on the current platform.

G_CONST_RETURN

```
#define     G_CONST_RETURN
```

If `G_DISABLE_CONST_RETURNS` is defined, this macro expands to nothing. By default, the macro expands to `const`. The macro should be used in place of `const` for functions that return a value that should not be modified. The purpose of this macro is to allow us to turn on `const` for returned constant strings by default, while allowing programmers who find that annoying to turn it off. This macro should only be used for return values and for *out* parameters, it doesn't make sense for *in* parameters.

<< Limits of Basic Types **Type Conversion Macros >>**

Type Conversion Macros

Type Conversion Macros — portably storing integers in pointer variables.

Synopsis

```
#include <glib.h>


#define     GINT_TO_POINTER                 (i)
#define     GPOINTER_TO_INT                 (p)

#define     GUINT_TO_POINTER                (u)
#define     GPOINTER_TO_UINT                (p)
#define     GSIZE_TO_POINTER                (s)
#define     GPOINTER_TO_SIZE                (p)
```

Description

Many times GLib, GTK+, and other libraries allow you to pass "user data" to a callback, in the form of a void pointer. From time to time you want to pass an integer instead of a pointer. You could allocate an integer, with something like:

```
int *ip = g_new (int, 1);
*ip = 42;
```

But this is inconvenient, and it's annoying to have to free the memory at some later time.

Pointers are always at least 32 bits in size (on all platforms GLib intends to support). Thus you can store at least 32-bit integer values in a pointer value. Naively, you might try this, but it's incorrect:

```
gpointer p;
int i;
p = (void*) 42;
i = (int) p;
```

Again, that example was *not* correct, don't copy it. The problem is that on some systems you need to do this:

```
gpointer p;
int i;
p = (void*) (long) 42;
i = (int) (long) p;
```

So GPOINTER_TO_INT(), GINT_TO_POINTER(), etc. do the right thing on the current platform.

Warning

YOU MAY NOT STORE POINTERS IN INTEGERS. THIS IS NOT PORTABLE IN ANY WAY SHAPE OR FORM. These macros *ONLY* allow storing integers in pointers, and only preserve 32 bits of the integer; values outside the range of a 32-bit integer will be mangled.

Details

GINT_TO_POINTER()

```
#define GINT_TO_POINTER(i)      ((gpointer)  (i))
```

Stuffs an integer into a pointer type.

Remember, YOU MAY NOT STORE POINTERS IN INTEGERS. THIS IS NOT PORTABLE IN ANY WAY SHAPE OR FORM. These macros *ONLY* allow storing integers in pointers, and only preserve 32 bits of the integer; values outside the range of a 32-bit integer will be mangled.

i : integer to stuff into a pointer.

GPOINTER_TO_INT()

```
#define GPOINTER_TO_INT(p)      ((gint)   (p))
```

Extracts an integer from a pointer. The integer must have been stored in the pointer with GINT_TO_POINTER().

Remember, YOU MAY NOT STORE POINTERS IN INTEGERS. THIS IS NOT PORTABLE IN ANY WAY SHAPE OR FORM. These macros *ONLY* allow storing integers in pointers, and only preserve 32 bits of the integer; values outside the range of a 32-bit integer will be mangled.

p : pointer containing an integer.

GUINT_TO_POINTER()

```
#define GUINT_TO_POINTER(u)     ((gpointer)  (u))
```

Stuffs an unsigned integer into a pointer type.

u : unsigned integer to stuff into the pointer.

GPOINTER_TO_UINT()

```
#define GPOINTER_TO_UINT(p)     ((guint)  (p))
```

Extracts an unsigned integer from a pointer. The integer must have been stored in the pointer with GUINT_TO_POINTER().

p : pointer to extract an unsigned integer from.

GSIZE_TO_POINTER()

```
#define GSIZE_TO_POINTER(s)     ((gpointer) (gsize) (s))
```

Stuffs a gsize into a pointer type.

s : gsize to stuff into the pointer.

GPOINTER_TO_SIZE()

```
#define GPOINTER_TO_SIZE(p)     ((gsize) (p))
```

Extracts a gsize from a pointer. The gsize must have been stored in the pointer with GSIZE_TO_POINTER().

p : pointer to extract a gsize from.

<< Standard Macros **Byte Order Macros >>**

Byte Order Macros

Byte Order Macros — a portable way to convert between different byte orders.

Synopsis

```
#include <glib.h>


#define     G_BYTE_ORDER
#define     G_LITTLE_ENDIAN
#define     G_BIG_ENDIAN
#define     G_PDP_ENDIAN

#define     g_htonl                         (val)
#define     g_htons                         (val)
#define     g_ntohl                         (val)
#define     g_ntohs                         (val)

#define     GINT_FROM_BE                    (val)
#define     GINT_FROM_LE                    (val)
#define     GINT_TO_BE                      (val)
#define     GINT_TO_LE                      (val)

#define     GUINT_FROM_BE                   (val)
#define     GUINT_FROM_LE                   (val)
#define     GUINT_TO_BE                     (val)
#define     GUINT_TO_LE                     (val)

#define     GLONG_FROM_BE                   (val)
#define     GLONG_FROM_LE                   (val)
#define     GLONG_TO_BE                     (val)
#define     GLONG_TO_LE                     (val)

#define     GULONG_FROM_BE                  (val)
#define     GULONG_FROM_LE                  (val)
#define     GULONG_TO_BE                    (val)
#define     GULONG_TO_LE                    (val)

#define     GINT16_FROM_BE                  (val)
#define     GINT16_FROM_LE                  (val)
#define     GINT16_TO_BE                    (val)
#define     GINT16_TO_LE                    (val)

#define     GUINT16_FROM_BE                 (val)
#define     GUINT16_FROM_LE                 (val)
#define     GUINT16_TO_BE                   (val)
#define     GUINT16_TO_LE                   (val)

#define     GINT32_FROM_BE                  (val)
#define     GINT32_FROM_LE                  (val)
#define     GINT32_TO_BE                    (val)
#define     GINT32_TO_LE                    (val)

#define     GUINT32_FROM_BE                 (val)
```



```
#define     GUINT32_FROM_LE                 (val)
#define     GUINT32_TO_BE                   (val)
#define     GUINT32_TO_LE                   (val)

#define     GINT64_FROM_BE                  (val)
#define     GINT64_FROM_LE                  (val)
#define     GINT64_TO_BE                    (val)
#define     GINT64_TO_LE                    (val)

#define     GUINT64_FROM_BE                 (val)
#define     GUINT64_FROM_LE                 (val)
#define     GUINT64_TO_BE                   (val)
#define     GUINT64_TO_LE                   (val)

#define     GUINT16_SWAP_BE_PDP             (val)
#define     GUINT16_SWAP_LE_BE              (val)
#define     GUINT16_SWAP_LE_PDP             (val)

#define     GUINT32_SWAP_BE_PDP             (val)
#define     GUINT32_SWAP_LE_BE              (val)
#define     GUINT32_SWAP_LE_PDP             (val)

#define     GUINT64_SWAP_LE_BE              (val)
```

Description

These macros provide a portable way to determine the host byte order and to convert values between different byte orders.

The byte order is the order in which bytes are stored to create larger data types such as the gint and glong values. The host byte order is the byte order used on the current machine.

Some processors store the most significant bytes (i.e. the bytes that hold the largest part of the value) first. These are known as big-endian processors.

Other processors (notably the x86 family) store the most significant byte last. These are known as little-endian processors.

Finally, to complicate matters, some other processors store the bytes in a rather curious order known as PDP-endian. For a 4-byte word, the 3rd most significant byte is stored first, then the 4th, then the 1st and finally the 2nd.

Obviously there is a problem when these different processors communicate with each other, for example over networks or by using binary file formats. This is where these macros come in. They are typically used to convert values into a byte order which has been agreed on for use when communicating between different processors. The Internet uses what is known as 'network byte order' as the standard byte order (which is in fact the big-endian byte order).

Details

G_BYTE_ORDER

```
#define G_BYTE_ORDER G_LITTLE_ENDIAN
```

The host byte order. This can be either G_LITTLE_ENDIAN or G_BIG_ENDIAN (support for G_PDP_ENDIAN may be added in future.)

G_LITTLE_ENDIAN

```
#define G_LITTLE_ENDIAN 1234
```

Specifies one of the possible types of byte order. See G_BYTE_ORDER.

G_BIG_ENDIAN

```
#define G_BIG_ENDIAN    4321
```

Specifies one of the possible types of byte order. See G_BYTE_ORDER.

G_PDP_ENDIAN

```
#define G_PDP_ENDIAN    3412            /* unused, need specific PDP check */
```

Specifies one of the possible types of byte order (currently unused). See G_BYTE_ORDER.

---

g_htonl()

```
#define     g_htonl(val)
```

Converts a 32-bit integer value from host to network byte order.

val : a 32-bit integer value in host byte order.
Returns : *val* converted to network byte order.

---

g_htons()

```
#define     g_htons(val)
```

Converts a 16-bit integer value from host to network byte order.

val : a 16-bit integer value in host byte order.
Returns : *val* converted to network byte order.

---

g_ntohl()

```
#define     g_ntohl(val)
```

Converts a 32-bit integer value from network to host byte order.

val : a 32-bit integer value in network byte order.
Returns : *val* converted to host byte order.

---

g_ntohs()

```
#define     g_ntohs(val)
```

Converts a 16-bit integer value from network to host byte order.

val : a 16-bit integer value in network byte order.
Returns : *val* converted to host byte order.

GINT_FROM_BE()

```
#define GINT_FROM_BE(val)       (GINT_TO_BE (val))
```

Converts a gint value from big-endian to host byte order.

val : a gint value in big-endian byte order.
Returns : *val* converted to host byte order.

GINT_FROM_LE()

```
#define GINT_FROM_LE(val)       (GINT_TO_LE (val))
```

Converts a gint value from little-endian to host byte order.

val : a gint value in little-endian byte order.
Returns : *val* converted to host byte order.

GINT_TO_BE()

```
#define GINT_TO_BE(val)         ((gint) GINT32_TO_BE (val))
```

Converts a gint value from host byte order to big-endian.

val : a gint value in host byte order.
Returns : *val* converted to big-endian byte order.

GINT_TO_LE()

```
#define GINT_TO_LE(val)         ((gint) GINT32_TO_LE (val))
```

Converts a gint value from host byte order to little-endian.

val : a gint value in host byte order.
Returns : *val* converted to little-endian byte order.

GUINT_FROM_BE()

```
#define GUINT_FROM_BE(val)      (GUINT_TO_BE (val))
```

Converts a guint value from big-endian to host byte order.

val : a guint value in big-endian byte order.
Returns : *val* converted to host byte order.

GUINT_FROM_LE()

```
#define GUINT_FROM_LE(val)      (GUINT_TO_LE (val))
```

Converts a guint value from little-endian to host byte order.

val : a guint value in little-endian byte order.
Returns : *val* converted to host byte order.

GUINT_TO_BE()

```
#define GUINT_TO_BE(val)        ((guint) GUINT32_TO_BE (val))
```

Converts a guint value from host byte order to big-endian.

val : a guint value in host byte order.
Returns : *val* converted to big-endian byte order.

GUINT_TO_LE()

```
#define GUINT_TO_LE(val)        ((guint) GUINT32_TO_LE (val))
```

Converts a guint value from host byte order to little-endian.

val : a guint value in host byte order.
Returns : *val* converted to little-endian byte order.

GLONG_FROM_BE()

```
#define GLONG_FROM_BE(val)      (GLONG_TO_BE (val))
```

Converts a glong value from big-endian to the host byte order.

val : a glong value in big-endian byte order.
Returns : *val* converted to host byte order.

GLONG_FROM_LE()

```
#define GLONG_FROM_LE(val)      (GLONG_TO_LE (val))
```

Converts a glong value from little-endian to host byte order.

val : a glong value in little-endian byte order.
Returns : *val* converted to host byte order.

GLONG_TO_BE()

```
#define GLONG_TO_BE(val)        ((glong) GINT32_TO_BE (val))
```

Converts a glong value from host byte order to big-endian.

val : a glong value in host byte order.
Returns : *val* converted to big-endian byte order.

GLONG_TO_LE()

```
#define GLONG_TO_LE(val)        ((glong) GINT32_TO_LE (val))
```

Converts a glong value from host byte order to little-endian.

val : a glong value in host byte order.
Returns : *val* converted to little-endian.

GULONG_FROM_BE()

```
#define GULONG_FROM_BE(val)     (GULONG_TO_BE (val))
```

Converts a gulong value from big-endian to host byte order.

val : a gulong value in big-endian byte order.
Returns : *val* converted to host byte order.

GULONG_FROM_LE()

```
#define GULONG_FROM_LE(val)     (GULONG_TO_LE (val))
```

Converts a gulong value from little-endian to host byte order.

val : a gulong value in little-endian byte order.
Returns : *val* converted to host byte order.

GULONG_TO_BE()

```
#define GULONG_TO_BE(val)       ((gulong) GUINT32_TO_BE (val))
```

Converts a gulong value from host byte order to big-endian.

val : a gulong value in host byte order.
Returns : *val* converted to big-endian.

GULONG_TO_LE()

```
#define GULONG_TO_LE(val)       ((gulong) GUINT32_TO_LE (val))
```

Converts a gulong value from host byte order to little-endian.

val : a gulong value in host byte order.
Returns : *val* converted to little-endian.

GINT16_FROM_BE()


```
#define GINT16_FROM_BE(val)     (GINT16_TO_BE (val))
```

Converts a gint16 value from big-endian to host byte order.

val : a gint16 value in big-endian byte order.
Returns : *val* converted to host byte order.

GINT16_FROM_LE()

```
#define GINT16_FROM_LE(val)     (GINT16_TO_LE (val))
```

Converts a gint16 value from little-endian to host byte order.

val : a gint16 value in little-endian byte order.
Returns : *val* converted to host byte order.

GINT16_TO_BE()

```
#define GINT16_TO_BE(val)       ((gint16) GUINT16_SWAP_LE_BE (val))
```

Converts a gint16 value from host byte order to big-endian.

val : a gint16 value in host byte order.
Returns : *val* converted to big-endian.

GINT16_TO_LE()

```
#define GINT16_TO_LE(val)       ((gint16) (val))
```

Converts a gint16 value from host byte order to little-endian.

val : a gint16 value in host byte order.
Returns : *val* converted to little-endian.

GUINT16_FROM_BE()

```
#define GUINT16_FROM_BE(val)    (GUINT16_TO_BE (val))
```

Converts a guint16 value from big-endian to host byte order.

val : a guint16 value in big-endian byte order.

Returns : *val* converted to host byte order.

GUINT16_FROM_LE()

```
#define GUINT16_FROM_LE(val)    (GUINT16_TO_LE (val))
```

Converts a guint16 value from little-endian to host byte order.

val : a guint16 value in little-endian byte order.
Returns : *val* converted to host byte order.

GUINT16_TO_BE()

```
#define GUINT16_TO_BE(val)      (GUINT16_SWAP_LE_BE (val))
```

Converts a guint16 value from host byte order to big-endian.

val : a guint16 value in host byte order.
Returns : *val* converted to big-endian.

GUINT16_TO_LE()

```
#define GUINT16_TO_LE(val)      ((guint16) (val))
```

Converts a guint16 value from host byte order to little-endian.

val : a guint16 value in host byte order.
Returns : *val* converted to little-endian.

GINT32_FROM_BE()

```
#define GINT32_FROM_BE(val)     (GINT32_TO_BE (val))
```

Converts a gint32 value from big-endian to host byte order.

val : a gint32 value in big-endian byte order.
Returns : *val* converted to host byte order.

GINT32_FROM_LE()


```
#define GINT32_FROM_LE(val)     (GINT32_TO_LE (val))
```

Converts a gint32 value from little-endian to host byte order.

val : a gint32 value in little-endian byte order.
Returns : *val* converted to host byte order.

GINT32_TO_BE()

```
#define GINT32_TO_BE(val)       ((gint32) GUINT32_SWAP_LE_BE (val))
```

Converts a gint32 value from host byte order to big-endian.

val : a gint32 value in host byte order.
Returns : *val* converted to big-endian.

GINT32_TO_LE()

```
#define GINT32_TO_LE(val)       ((gint32) (val))
```

Converts a gint32 value from host byte order to little-endian.

val : a gint32 value in host byte order.
Returns : *val* converted to little-endian.

GUINT32_FROM_BE()

```
#define GUINT32_FROM_BE(val)    (GUINT32_TO_BE (val))
```

Converts a guint32 value from big-endian to host byte order.

val : a guint32 value in big-endian byte order.
Returns : *val* converted to host byte order.

GUINT32_FROM_LE()

```
#define GUINT32_FROM_LE(val)    (GUINT32_TO_LE (val))
```

Converts a guint32 value from little-endian to host byte order.

val : a guint32 value in little-endian byte order.

Returns : `val` converted to host byte order.

GUINT32_TO_BE()

```
#define GUINT32_TO_BE(val)      (GUINT32_SWAP_LE_BE (val))
```

Converts a guint32 value from host byte order to big-endian.

`val` : a guint32 value in host byte order.
Returns : `val` converted to big-endian.

GUINT32_TO_LE()

```
#define GUINT32_TO_LE(val)      ((guint32) (val))
```

Converts a guint32 value from host byte order to little-endian.

`val` : a guint32 value in host byte order.
Returns : `val` converted to little-endian.

GINT64_FROM_BE()

```
#define GINT64_FROM_BE(val)     (GINT64_TO_BE (val))
```

Converts a gint64 value from big-endian to host byte order.

`val` : a gint64 value in big-endian byte order.
Returns : `val` converted to host byte order.

GINT64_FROM_LE()

```
#define GINT64_FROM_LE(val)     (GINT64_TO_LE (val))
```

Converts a gint64 value from little-endian to host byte order.

`val` : a gint64 value in little-endian byte order.
Returns : `val` converted to host byte order.

GINT64_TO_BE()


```
#define GINT64_TO_BE(val)       ((gint64) GUINT64_SWAP_LE_BE (val))
```

Converts a gint64 value from host byte order to big-endian.

`val` : a gint64 value in host byte order.
Returns : `val` converted to big-endian.

GINT64_TO_LE()

```
#define GINT64_TO_LE(val)       ((gint64) (val))
```

Converts a gint64 value from host byte order to little-endian.

`val` : a gint64 value in host byte order.
Returns : `val` converted to little-endian.

GUINT64_FROM_BE()

```
#define GUINT64_FROM_BE(val)    (GUINT64_TO_BE (val))
```

Converts a guint64 value from big-endian to host byte order.

`val` : a guint64 value in big-endian byte order.
Returns : `val` converted to host byte order.

GUINT64_FROM_LE()

```
#define GUINT64_FROM_LE(val)    (GUINT64_TO_LE (val))
```

Converts a guint64 value from little-endian to host byte order.

`val` : a guint64 value in little-endian byte order.
Returns : `val` converted to host byte order.

GUINT64_TO_BE()

```
#define GUINT64_TO_BE(val)      (GUINT64_SWAP_LE_BE (val))
```

Converts a guint64 value from host byte order to big-endian.

`val` : a guint64 value in host byte order.

Returns : `val` converted to big-endian.

GUINT64_TO_LE()

```
#define GUINT64_TO_LE(val)      ((guint64) (val))
```

Converts a guint64 value from host byte order to little-endian.

`val` : a guint64 value in host byte order.
Returns : `val` converted to little-endian.

GUINT16_SWAP_BE_PDP()

```
#define GUINT16_SWAP_BE_PDP(val)        (GUINT16_SWAP_LE_BE (val))
```

Converts a guint16 value between big-endian and pdp-endian byte order. The conversion is symmetric so it can be used both ways.

`val` : a guint16 value in big-endian or pdp-endian byte order.
Returns : `val` converted to the opposite byte order.

GUINT16_SWAP_LE_BE()

```
#define     GUINT16_SWAP_LE_BE(val)
```

Converts a guint16 value between little-endian and big-endian byte order. The conversion is symmetric so it can be used both ways.

`val` : a guint16 value in little-endian or big-endian byte order.
Returns : `val` converted to the opposite byte order.

GUINT16_SWAP_LE_PDP()

```
#define GUINT16_SWAP_LE_PDP(val)        ((guint16) (val))
```

Converts a guint16 value between little-endian and pdp-endian byte order. The conversion is symmetric so it can be used both ways.

`val` : a guint16 value in little-endian or pdp-endian byte order.
Returns : `val` converted to the opposite byte order.

GUINT32_SWAP_BE_PDP()

```
#define     GUINT32_SWAP_BE_PDP(val)
```

Converts a guint32 value between big-endian and pdp-endian byte order. The conversion is symmetric so it can be used both ways.

`val` : a guint32 value in big-endian or pdp-endian byte order.
Returns : `val` converted to the opposite byte order.

GUINT32_SWAP_LE_BE()

```
#define     GUINT32_SWAP_LE_BE(val)
```

Converts a guint32 value between little-endian and big-endian byte order. The conversion is symmetric so it can be used both ways.

`val` : a guint32 value in little-endian or big-endian byte order.
Returns : `val` converted to the opposite byte order.

GUINT32_SWAP_LE_PDP()

```
#define     GUINT32_SWAP_LE_PDP(val)
```

Converts a guint32 value between little-endian and pdp-endian byte order. The conversion is symmetric so it can be used both ways.

`val` : a guint32 value in little-endian or pdp-endian byte order.
Returns : `val` converted to the opposite byte order.

GUINT64_SWAP_LE_BE()

```
#define     GUINT64_SWAP_LE_BE(val)
```

Converts a guint64 value between little-endian and big-endian byte order. The conversion is symmetric so it can be used both ways.

`val` : a guint64 value in little-endian or big-endian byte order.
Returns : `val` converted to the opposite byte order.

<< Type Conversion Macros **Numerical Definitions >>**

Numerical Definitions

Numerical Definitions — mathematical constants, and floating point decomposition.

Synopsis

```
#include <glib.h>


#define     G_IEEE754_FLOAT_BIAS
#define     G_IEEE754_DOUBLE_BIAS
union       GFloatIEEE754;
union       GDoubleIEEE754;


#define     G_E
#define     G_LN2
#define     G_LN10
#define     G_PI
#define     G_PI_2
#define     G_PI_4
#define     G_SQRT2
#define     G_LOG_2_BASE_10
```

Description

GLib offers mathematical constants such as G_PI for the value of pi; many platforms have these in the C library, but some don't, the GLib versions always exist.

The GFloatIEEE754 and GDoubleIEEE754 unions are used to access the sign, mantissa and exponent of IEEE floats and doubles. These unions are defined as appropriate for a given platform. IEEE floats and doubles are supported (used for storage) by at least Intel, PPC and Sparc, for reference: http://cch.loria.fr/documentation/IEEE754/numerical_comp_guide/ncg_math.doc.html

Details

G_IEEE754_FLOAT_BIAS

```
#define G_IEEE754_FLOAT_BIAS    (127)
```

See http://cch.loria.fr/documentation/IEEE754/numerical_comp_guide/ncg_math.doc.html

---

G_IEEE754_DOUBLE_BIAS


```
#define G_IEEE754_DOUBLE_BIAS   (1023)
```

See http://cch.loria.fr/documentation/IEEE754/numerical_comp_guide/ncg_math.doc.html

---

union GFloatIEEE754

```
union GFloatIEEE754
{
  gfloat v_float;
  struct {
    guint mantissa : 23;
    guint biased_exponent : 8;
    guint sign : 1;
  } mpn;
};
```

The GFloatIEEE754 and GDoubleIEEE754 unions are used to access the sign, mantissa and exponent of IEEE floats and doubles. These unions are defined as appropriate for a given platform. IEEE floats and doubles are supported (used for storage) by at least Intel, PPC and Sparc, for reference: http://cch.loria.fr/documentation/IEEE754/numerical_comp_guide/ncg_math.doc.html

---

union GDoubleIEEE754

```
union GDoubleIEEE754
{
  gdouble v_double;
  struct {
    guint mantissa_low : 32;
    guint mantissa_high : 20;
    guint biased_exponent : 11;
    guint sign : 1;
  } mpn;
};
```

The GFloatIEEE754 and GDoubleIEEE754 unions are used to access the sign, mantissa and exponent of IEEE floats and doubles. These unions are defined as appropriate for a given platform. IEEE floats and doubles are supported (used for storage) by at least Intel, PPC and Sparc, for reference: http://cch.loria.fr/documentation/IEEE754/numerical_comp_guide/ncg_math.doc.html

---

G_E

```
#define G_E     2.7182818284590452353602874713526624977572470937000
```

The base of natural logarithms.

G_LN2

```
#define G_LN2   0.693147180559945309417232121458176568075500134360255
```

The natural logarithm of 2.

G_LN10

```
#define G_LN10  2.302585092994045684017991454684364207601101488628772
```

The natural logarithm of 10.

G_PI

```
#define G_PI    3.141592653589793238462643383279502884197169399375105
```

The value of pi (ratio of circle's circumference to its diameter).

G_PI_2

```
#define G_PI_2  1.570796326794896619231321691639751442098584699687552
```

Pi divided by 2.

G_PI_4

```
#define G_PI_4  0.785398163397448309615660845819875721049292349843776
```

Pi divided by 4.

G_SQRT2

```
#define G_SQRT2 1.414213562373095048801688724209698078569671875376948
```

The square root of two.

G_LOG_2_BASE_10

```
#define G_LOG_2_BASE_10         (0.30102999566398119521)
```

Used for fooling around with float formats, see
http://cch.loria.fr/documentation/IEEE754/numerical_comp_guide/ncg_math.doc.html

See Also

http://cch.loria.fr/documentation/IEEE754/numerical_comp_guide/ncg_math.doc.html

<< Byte Order Macros **Miscellaneous Macros >>**

Miscellaneous Macros

Miscellaneous Macros — specialized macros which are not used often.

Synopsis

```
#include <glib.h>


#define     G_INLINE_FUNC

#define     G_STMT_START
#define     G_STMT_END

#define     G_BEGIN_DECLS
#define     G_END_DECLS

#define     G_N_ELEMENTS                    (arr)

#define     G_VA_COPY

#define     G_STRINGIFY                     (macro_or_string)

#define     G_GNUC_EXTENSION
#define     G_GNUC_CONST
#define     G_GNUC_DEPRECATED
#define     G_GNUC_NORETURN
#define     G_GNUC_UNUSED
#define     G_GNUC_PURE
#define     G_GNUC_PRINTF                   ( format_idx, arg_idx )
#define     G_GNUC_SCANF                    ( format_idx, arg_idx )
#define     G_GNUC_FORMAT                   ( arg_idx )
#define     G_GNUC_FUNCTION
#define     G_GNUC_PRETTY_FUNCTION
#define     G_GNUC_NO_INSTRUMENT
#define     G_GNUC_INTERNAL

#define     G_LIKELY                        (expr)
#define     G_UNLIKELY                      (expr)

#define     G_STRLOC
#define     G_STRFUNC

#define     G_GINT16_MODIFIER
#define     G_GINT16_FORMAT
#define     G_GUINT16_FORMAT
#define     G_GINT32_MODIFIER
#define     G_GINT32_FORMAT
#define     G_GUINT32_FORMAT
#define     G_GINT64_MODIFIER
#define     G_GINT64_FORMAT
#define     G_GUINT64_FORMAT
#define     G_GSIZE_MODIFIER
#define     G_GSIZE_FORMAT
#define     G_GSSIZE_FORMAT
```


Description

These macros provide more specialized features which are not needed so often by application programmers.

Details

G_INLINE_FUNC

```
#define     G_INLINE_FUNC
```

Used to declare inline functions. If inline functions are not supported on the particular platform, the macro evaluates to the empty string.

G_STMT_START

```
#define     G_STMT_START
```

Used within multi-statement macros so that they can be used in places where only one statement is expected by the compiler.

G_STMT_END

```
#define     G_STMT_END
```

Used within multi-statement macros so that they can be used in places where only one statement is expected by the compiler.

G_BEGIN_DECLS

```
#define     G_BEGIN_DECLS
```

Used (along with G_END_DECLS) to bracket header files. If the compiler in use is a C++ compiler, adds `extern "C"` around the header.

G_END_DECLS

```
#define     G_END_DECLS
```

Used (along with G_BEGIN_DECLS) to bracket header files. If the compiler in use is a C++ compiler, adds `extern "C"` around the header.

G_N_ELEMENTS()

```
#define G_N_ELEMENTS(arr)               (sizeof (arr) / sizeof ((arr)[0]))
```

Determines the number of elements in an array. The array must be declared so the compiler knows its size at compile-time; this macro will not work on an array allocated on the heap, only static arrays or arrays on the stack.

arr : the array

G_VA_COPY

```
#define     G_VA_COPY
```

Portable way to copy va_list variables.

In order to use this function, you must include `string.h` yourself, because this macro may use `memmove()` and GLib does not include `string.h` for you.

G_STRINGIFY()

```
#define G_STRINGIFY(macro_or_string)    G_STRINGIFY_ARG (macro_or_string)
```

Accepts a macro or a string and converts it into a string.

macro_or_string : a macro or a string.

G_GNUC_EXTENSION

```
#define     G_GNUC_EXTENSION
```

Expands to __extension__ when **gcc** is used as the compiler. This simply tells **gcc** not to warn about the following non-standard code when compiling with the `-pedantic` option.

G_GNUC_CONST

```
#define     G_GNUC_CONST
```


Expands to the GNU C `const` function attribute if the compiler is **gcc**. Declaring a function as const enables better optimization of the function. A const function doesn't examine any values except its parameters, and has no effects except its return value. See the GNU C documentation for details.

> **Note**
>
> A function that has pointer arguments and examines the data pointed to must *not* be declared const. Likewise, a function that calls a non-const function usually must not be const. It doesn't make sense for a const function to return void.

G_GNUC_DEPRECATED

```
#define     G_GNUC_DEPRECATED
```

Expands to the GNU C `deprecated` attribute if the compiler is **gcc**. It can be used to mark typedefs, variables and functions as deprecated. When called with the `-Wdeprecated` option, the compiler will generate warnings when deprecated interfaces are used. See the GNU C documentation for details.

Since 2.2

G_GNUC_NORETURN

```
#define     G_GNUC_NORETURN
```

Expands to the GNU C `noreturn` function attribute if the compiler is **gcc**. It is used for declaring functions which never return. It enables optimization of the function, and avoids possible compiler warnings. See the GNU C documentation for details.

G_GNUC_UNUSED

```
#define     G_GNUC_UNUSED
```

Expands to the GNU C `unused` function attribute if the compiler is **gcc**. It is used for declaring functions which may never be used. It avoids possible compiler warnings. See the GNU C documentation for details.

G_GNUC_PURE

```
#define     G_GNUC_PURE
```

Expands to the GNU C `pure` function attribute if the compiler is **gcc**. Declaring a function as pure enables better optimization of the function. A pure function has no effects except its return value and

the return value depends only on the parameters and/or global variables. See the GNU C documentation for details.

G_GNUC_PRINTF()

```
#define     G_GNUC_PRINTF( format_idx, arg_idx )
```

Expands to the GNU C `format` function attribute if the compiler is **gcc**. This is used for declaring functions which take a variable number of arguments, with the same syntax as `printf()`. It allows the compiler to type-check the arguments passed to the function. See the GNU C documentation for details.

```
gint g_snprintf (gchar  *string,
                 gulong       n,
                 gchar const *format,
                 ...) G_GNUC_PRINTF (3, 4);
```

`format_idx` : the index of the argument corresponding to the format string. (The arguments are numbered from 1).

`arg_idx` : the index of the first of the format arguments.

G_GNUC_SCANF()

```
#define     G_GNUC_SCANF( format_idx, arg_idx )
```

Expands to the GNU C `format` function attribute if the compiler is **gcc**. This is used for declaring functions which take a variable number of arguments, with the same syntax as `scanf()`. It allows the compiler to type-check the arguments passed to the function. See the GNU C documentation for details.

`format_idx` : the index of the argument corresponding to the format string. (The arguments are numbered from 1).

`arg_idx` : the index of the first of the format arguments.

G_GNUC_FORMAT()

```
#define     G_GNUC_FORMAT( arg_idx )
```

Expands to the GNU C `format_arg` function attribute if the compiler is **gcc**. This function attribute specifies that a function takes a format string for a `printf()`, `scanf()`, `strftime()` or `strfmon()` style function and modifies it, so that the result can be passed to a `printf()`, `scanf()`, `strftime()` or `strfmon()` style function (with the remaining arguments to the format function the same as they would have been for the unmodified string). See the GNU C documentation for details.


```
gchar *g_dgettext (gchar *domain_name, gchar *msgid) G_GNUC_FORMAT (2);
```

`arg_idx` : the index of the argument.

G_GNUC_FUNCTION

```
#define     G_GNUC_FUNCTION
```

Expands to the GNU C `__FUNCTION__` variable if the compiler is **gcc**, or "" if it isn't. The GNU C `__FUNCTION__` variable contains the name of the current function. See the GNU C documentation for details.

G_GNUC_PRETTY_FUNCTION

```
#define     G_GNUC_PRETTY_FUNCTION
```

Expands to the GNU C `__PRETTY_FUNCTION__` variable if the compiler is **gcc**, or "" if it isn't. The GNU C `__PRETTY_FUNCTION__` variable contains the name of the current function. For a C program this is the same as the `__FUNCTION__` variable but for C++ it also includes extra information such as the class and function prototype. See the GNU C documentation for details.

G_GNUC_NO_INSTRUMENT

```
#define     G_GNUC_NO_INSTRUMENT
```

Expands to the GNU C `no_instrument_function` function attribute if the compiler is **gcc**. Functions with this attribute will not be instrumented for profiling, when the compiler is called with the `-finstrument-functions` option. See the GNU C documentation for details.

G_GNUC_INTERNAL

```
#define G_GNUC_INTERNAL
```

Expands to the GNU C `visibility(hidden)` attribute if the compiler supports it (currently only **gcc**). This attribute can be used for marking library functions as being used internally to the lib only, to not create inefficient PLT entries. Note that static functions do not need to be marked as internal in this way. See the GNU C documentation for details.

Since: 2.6

G_LIKELY()

```
#define     G_LIKELY(expr)
```

Hints the compiler that the expression is likely to evaluate to a true value. The compiler may use this information for optimizations.

```
if (G_LIKELY (random() != 1))
  g_print ("not one");
```

expr : the expression

Since 2.2

G_UNLIKELY()

```
#define     G_UNLIKELY(expr)
```

Hints the compiler that the expression is unlikely to evaluate to a true value. The compiler may use this information for optimizations.

```
if (G_UNLIKELY (random() == 1))
  g_print ("a random one");
```

expr : the expression

Since 2.2

G_STRLOC

```
#define     G_STRLOC
```

Expands to a string identifying the current code position.

G_STRFUNC

```
#define     G_STRFUNC
```

Expands to a string identifying the current function.

Since 2.4

G_GINT16_MODIFIER

```
#define G_GINT16_MODIFIER "h"
```

The platform dependent length modifier for constructing `printf()` conversion specifiers for values of type gint16 or guint16. It is a string literal, but doesn't include the percent-sign, such that you can add precision and length modifiers between percent-sign and conversion specifier and append a conversion specifier.

The following example prints "0x7b";

```
gint16 value = 123;
g_print ("%#" G_GINT16_MODIFIER "x", value);
```

Since 2.4

G_GINT16_FORMAT

```
#define G_GINT16_FORMAT "hi"
```

This is the platform dependent conversion specifier for scanning and printing values of type gint16. It is a string literal, but doesn't include the percent-sign, such that you can add precision and length modifiers between percent-sign and conversion specifier.

```
gint16 in;
gint32 out;
sscanf ("42", "%" G_GINT16_FORMAT, &in)
out = in * 1000;
g_print ("%" G_GINT32_FORMAT, out);
```

G_GUINT16_FORMAT

```
#define G_GUINT16_FORMAT "hu"
```

This is the platform dependent conversion specifier for scanning and printing values of type guint16. See also G_GINT16_FORMAT.

G_GINT32_MODIFIER

```
#define G_GINT32_MODIFIER ""
```

The platform dependent length modifier for constructing `printf()` conversion specifiers for values of type gint32 or guint32. See also G_GINT16_MODIFIER.

Since 2.4

G_GINT32_FORMAT

```
#define G_GINT32_FORMAT "i"
```

This is the platform dependent conversion specifier for scanning and printing values of type gint32. See also G_GINT16_FORMAT.

G_GUINT32_FORMAT

```
#define G_GUINT32_FORMAT "u"
```

This is the platform dependent conversion specifier for scanning and printing values of type guint32. See also G_GINT16_FORMAT.

G_GINT64_MODIFIER

```
#define G_GINT64_MODIFIER "ll"
```

The platform dependent length modifier for constructing `printf()` conversion specifiers for values of type gint64 or guint64. See also G_GINT16_MODIFIER.

Note

Some platforms do not support printing 64 bit integers, even though the types are supported. On such platforms G_GINT64_MODIFIER is not defined.

Since 2.4

G_GINT64_FORMAT

```
#define G_GINT64_FORMAT "lli"
```

This is the platform dependent conversion specifier for scanning and printing values of type gint64. See also G_GINT16_FORMAT.

Note

Some platforms do not support scanning and printing 64 bit integers, even though the types are supported. On such platforms G_GINT64_FORMAT is not defined. Note that `scanf()` may not support 64 bit integers, even if G_GINT64_FORMAT is defined. Due to its weak error handling, `scanf()` is not recommended for parsing anyway; consider using `g_strtoull()` instead.

G_GUINT64_FORMAT

```
#define G_GUINT64_FORMAT "llu"
```

This is the platform dependent conversion specifier for scanning and printing values of type guint64. See also G_GINT16_FORMAT.

Note

Some platforms do not support scanning and printing 64 bit integers, even though the types are supported. On such platforms G_GUINT64_FORMAT is not defined. Note that `scanf()` may not support 64 bit integers, even if G_GINT64_FORMAT is defined. Due to its weak error handling, `scanf()` is not recommended for parsing anyway; consider using `g_strtoull()` instead.

G_GSIZE_MODIFIER

```
#define G_GSIZE_MODIFIER ""
```

The platform dependent length modifier for constructing `printf()` conversion specifiers for values of type gsize or gssize. See also G_GINT16_MODIFIER.

Since 2.6

G_GSIZE_FORMAT

```
#define G_GSIZE_FORMAT "u"
```

This is the platform dependent conversion specifier for scanning and printing values of type gsize. See also G_GINT16_FORMAT.

Since 2.6

G_GSSIZE_FORMAT

```
#define G_GSSIZE_FORMAT "i"
```

This is the platform dependent conversion specifier for scanning and printing values of type gssize.
See also G_GINT16_FORMAT.

Since 2.6

<< Numerical Definitions **Atomic Operations >>**

Atomic Operations

Atomic Operations — basic atomic integer and pointer operations

Synopsis

```
#include <glib.h>


gint        g_atomic_int_get                (gint *atomic);
void        g_atomic_int_add                (gint *atomic,
                                             gint val);
gint        g_atomic_int_exchange_and_add   (gint *atomic,
                                             gint val);
gboolean    g_atomic_int_compare_and_exchange
                                            (gint *atomic,
                                             gint oldval,
                                             gint newval);
gpointer    g_atomic_pointer_get            (gpointer *atomic);
gboolean    g_atomic_pointer_compare_and_exchange
                                            (gpointer *atomic,
                                             gpointer oldval,
                                             gpointer newval);
void        g_atomic_int_inc                (gint *atomic);
gboolean    g_atomic_int_dec_and_test       (gint *atomic);
```

Description

The following functions can be used to atomically access integers and pointers. They are implemented as inline assembler function on most platforms and use slower fall-backs otherwise. Using them can sometimes save you from using a performance-expensive GMutex to protect the integer or pointer.

The most important usage is reference counting. Using `g_atomic_int_inc()` and `g_atomic_int_dec_and_test()` makes reference counting a very fast operation.

Note

You must not directly read integers or pointers concurrently accessed by other threads with with the following functions directly. Always use `g_atomic_int_get()` and `g_atomic_pointer_get()` respectively. They are acting as a memory barrier.

Note

If you are using those functions for anything apart from simple reference counting, you should really be aware of the implications of doing that. There are literally thousands of ways to shoot yourself in the foot. So if in doubt, use a GMutex. If you don't know, what memory barriers are, do not use anything but `g_atomic_int_inc()` and `g_atomic_int_dec_and_test()`.

Note

It is not safe to set an integer or pointer just by assigning to it, when it is concurrently accessed by other threads with the following functions. Use `g_atomic_int_compare_and_exchange()` or `g_atomic_pointer_compare_and_exchange()` respectively.

Details

g_atomic_int_get ()

```
gint        g_atomic_int_get                (gint *atomic);
```

Reads the value of the integer pointed to by *atomic*. Also acts as a memory barrier.

atomic : a pointer to an integer.
Returns : the value of **atomic*.

Since 2.4

---

g_atomic_int_add ()

```
void        g_atomic_int_add                (gint *atomic,
                                             gint val);
```

Atomically adds *val* to the integer pointed to by *atomic*. Also acts as a memory barrier.

atomic : a pointer to an integer.
val : the value to add to **atomic*.

Since 2.4

---

g_atomic_int_exchange_and_add ()

```
gint        g_atomic_int_exchange_and_add   (gint *atomic,
                                             gint val);
```

Atomically adds *val* to the integer pointed to by *atomic*. It returns the value of **atomic* just before the addition took place. Also acts as a memory barrier.

atomic : a pointer to an integer.
val : the value to add to **atomic*.
Returns : the value of **atomic* before the addition.

Since 2.4

---

g_atomic_int_compare_and_exchange ()

```
gboolean    g_atomic_int_compare_and_exchange
                                        (gint *atomic,
                                         gint oldval,
                                         gint newval);
```

Compares *oldval* with the integer pointed to by *atomic* and if they are equal, atomically exchanges **atomic* with *newval*. Also acts as a memory barrier.

atomic : a pointer to an integer.
oldval : the assumed old value of **atomic*.
newval : the new value of **atomic*.
Returns : TRUE, if **atomic* was equal *oldval*. FALSE otherwise.

Since 2.4

---

g_atomic_pointer_get ()

```
gpointer    g_atomic_pointer_get            (gpointer *atomic);
```

Reads the value of the pointer pointed to by *atomic*. Also acts as a memory barrier.

atomic : a pointer to a gpointer.
Returns : the value to add to **atomic*.

Since 2.4

---

g_atomic_pointer_compare_and_exchange ()

```
gboolean    g_atomic_pointer_compare_and_exchange
                                        (gpointer *atomic,
                                         gpointer oldval,
                                         gpointer newval);
```

Compares *oldval* with the pointer pointed to by *atomic* and if they are equal, atomically exchanges **atomic* with *newval*. Also acts as a memory barrier.

atomic : a pointer to a gpointer.
oldval : the assumed old value of **atomic*.

newval : the new value of **atomic*.
Returns : TRUE, if **atomic* was equal *oldval*. FALSE otherwise.

Since 2.4

---

g_atomic_int_inc ()

```
void        g_atomic_int_inc                (gint *atomic);
```

Atomically increments the integer pointed to by *atomic* by 1.

atomic : a pointer to an integer.

Since 2.4

---

g_atomic_int_dec_and_test ()

```
gboolean    g_atomic_int_dec_and_test       (gint *atomic);
```

Atomically decrements the integer pointed to by *atomic* by 1.

atomic : a pointer to an integer.
Returns : TRUE, if the integer pointed to by *atomic* is 0 after decrementing it.

Since 2.4

See Also

GMutex GLib mutual exclusions.

<< Miscellaneous Macros **GLib Core Application Support >>**

GLib Core Application Support

The Main Event Loop - manages all available sources of events.
Threads - thread abstraction; including threads, different mutexes, conditions and thread private data.
Thread Pools - pools of threads to execute work concurrently.
Asynchronous Queues - asynchronous communication between threads.
Dynamic Loading of Modules - portable method for dynamically loading 'plug-ins'.
Memory Allocation - general memory-handling.
IO Channels - portable support for using files, pipes and sockets.
Error Reporting - a system for reporting errors.
Message Output and Debugging Functions - functions to output messages and help debug applications.
Message Logging - versatile support for logging messages with different levels of importance.

<< Atomic Operations **The Main Event Loop >>**

The Main Event Loop

The Main Event Loop — manages all available sources of events.

Synopsis

```
#include <glib.h>


            GMainLoop;
GMainLoop*  g_main_loop_new                 (GMainContext *context,
                                             gboolean is_running);
GMainLoop*  g_main_loop_ref                 (GMainLoop *loop);
void        g_main_loop_unref               (GMainLoop *loop);
void        g_main_loop_run                 (GMainLoop *loop);
void        g_main_loop_quit                (GMainLoop *loop);
gboolean    g_main_loop_is_running          (GMainLoop *loop);
GMainContext* g_main_loop_get_context       (GMainLoop *loop);
#define     g_main_new                      (is_running)
#define     g_main_destroy                  (loop)
#define     g_main_run                      (loop)
#define     g_main_quit                     (loop)
#define     g_main_is_running               (loop)

#define     G_PRIORITY_HIGH
#define     G_PRIORITY_DEFAULT
#define     G_PRIORITY_HIGH_IDLE
#define     G_PRIORITY_DEFAULT_IDLE
#define     G_PRIORITY_LOW

            GMainContext;
GMainContext* g_main_context_new            (void);
GMainContext* g_main_context_ref            (GMainContext *context);
void        g_main_context_unref            (GMainContext *context);
GMainContext* g_main_context_default        (void);
gboolean    g_main_context_iteration        (GMainContext *context,
                                             gboolean may_block);
#define     g_main_iteration                (may_block)
gboolean    g_main_context_pending          (GMainContext *context);
#define     g_main_pending                  ()
GSource*    g_main_context_find_source_by_id
                                            (GMainContext *context,
                                             guint source_id);
GSource*    g_main_context_find_source_by_user_data
                                            (GMainContext *context,
                                             gpointer user_data);
GSource*    g_main_context_find_source_by_funcs_user_data
                                            (GMainContext *context,
                                             GSourceFuncs *funcs,
                                             gpointer user_data);
void        g_main_context_wakeup           (GMainContext *context);
gboolean    g_main_context_acquire          (GMainContext *context);
void        g_main_context_release          (GMainContext *context);
gboolean    g_main_context_wait             (GMainContext *context,
                                             GCond *cond,
```



```
                                             GMutex *mutex);
gboolean    g_main_context_prepare          (GMainContext *context,
                                             gint *priority);
gint        g_main_context_query            (GMainContext *context,
                                             gint max_priority,
                                             gint *timeout_,
                                             GPollFD *fds,
                                             gint n_fds);
gint        g_main_context_check            (GMainContext *context,
                                             gint max_priority,
                                             GPollFD *fds,
                                             gint n_fds);
void        g_main_context_dispatch         (GMainContext *context);
void        g_main_context_set_poll_func    (GMainContext *context,
                                             GPollFunc func);
GPollFunc   g_main_context_get_poll_func    (GMainContext *context);
gint        (*GPollFunc)                    (GPollFD *ufds,
                                             guint nfsd,
                                             gint timeout_);
void        g_main_context_add_poll         (GMainContext *context,
                                             GPollFD *fd,
                                             gint priority);
void        g_main_context_remove_poll      (GMainContext *context,
                                             GPollFD *fd);
int         g_main_depth                    (void);
#define     g_main_set_poll_func            (func)

GSource*    g_timeout_source_new            (guint interval);
guint       g_timeout_add                   (guint interval,
                                             GSourceFunc function,
                                             gpointer data);
guint       g_timeout_add_full              (gint priority,
                                             guint interval,
                                             GSourceFunc function,
                                             gpointer data,
                                             GDestroyNotify notify);

GSource*    g_idle_source_new               (void);
guint       g_idle_add                      (GSourceFunc function,
                                             gpointer data);
guint       g_idle_add_full                 (gint priority,
                                             GSourceFunc function,
                                             gpointer data,
                                             GDestroyNotify notify);
gboolean    g_idle_remove_by_data           (gpointer data);

typedef     GPid;
void        (*GChildWatchFunc)              (GPid pid,
                                             gint status,
                                             gpointer data);
GSource*    g_child_watch_source_new        (GPid pid);
guint       g_child_watch_add               (GPid pid,
                                             GChildWatchFunc function,
                                             gpointer data);
guint       g_child_watch_add_full          (gint priority,
                                             GPid pid,
                                             GChildWatchFunc function,
                                             gpointer data,
                                             GDestroyNotify notify);

            GPollFD;

            GSource;
void        (*GSourceDummyMarshal)          (void);
```

```
            GSourceFuncs;
            GSourceCallbackFuncs;
GSource*    g_source_new                    (GSourceFuncs *source_funcs,
                                             guint struct_size);
GSource*    g_source_ref                    (GSource *source);
void        g_source_unref                  (GSource *source);
guint       g_source_attach                 (GSource *source,
                                             GMainContext *context);
void        g_source_destroy                (GSource *source);
void        g_source_set_priority           (GSource *source,
                                             gint priority);
gint        g_source_get_priority           (GSource *source);
void        g_source_set_can_recurse        (GSource *source,
                                             gboolean can_recurse);
gboolean    g_source_get_can_recurse        (GSource *source);
guint       g_source_get_id                 (GSource *source);
GMainContext* g_source_get_context          (GSource *source);
void        g_source_set_callback           (GSource *source,
                                             GSourceFunc func,
                                             gpointer data,
                                             GDestroyNotify notify);
gboolean    (*GSourceFunc)                  (gpointer data);
void        g_source_set_callback_indirect  (GSource *source,
                                             gpointer callback_data,
                                             GSourceCallbackFuncs *callback_funcs
void        g_source_add_poll               (GSource *source,
                                             GPollFD *fd);
void        g_source_remove_poll            (GSource *source,
                                             GPollFD *fd);
void        g_source_get_current_time       (GSource *source,
                                             GTimeVal *timeval);
gboolean    g_source_remove                 (guint tag);
gboolean    g_source_remove_by_funcs_user_data
                                            (GSourceFuncs *funcs,
                                             gpointer user_data);
gboolean    g_source_remove_by_user_data    (gpointer user_data);
```

Description

The main event loop manages all the available sources of events for GLib and GTK+ applications. These events can come from any number of different types of sources such as file descriptors (plain files, pipes or sockets) and timeouts. New types of event sources can also be added using `g_source_attach()`.

To allow multiple independent sets of sources to be handled in different threads, each source is associated with a GMainContext. A GMainContext can only be running in a single thread, but sources can be added to it and removed from it from other threads.

Each event source is assigned a priority. The default priority, G_PRIORITY_DEFAULT, is 0. Values less than 0 denote higher priorities. Values greater than 0 denote lower priorities. Events from high priority sources are always processed before events from lower priority sources.

Idle functions can also be added, and assigned a priority. These will be run whenever no events with a higher priority are ready to be processed.

The GMainLoop data type represents a main event loop. A GMainLoop is created with `g_main_loop_new()`. After adding the initial event sources, `g_main_loop_run()` is called. This continuously checks for new events from each of the event sources and dispatches them. Finally, the

processing of an event from one of the sources leads to a call to `g_main_loop_quit()` to exit the main loop, and `g_main_loop_run()` returns.

It is possible to create new instances of GMainLoop recursively. This is often used in GTK+ applications when showing modal dialog boxes. Note that event sources are associated with a particular GMainContext, and will be checked and dispatched for all main loops associated with that GMainContext.

GTK+ contains wrappers of some of these functions, e.g. `gtk_main()`, `gtk_main_quit()` and `gtk_events_pending()`.

Creating new sources types

One of the unusual features of the GTK+ main loop functionality is that new types of event source can be created and used in addition to the builtin type of event source. A new event source type is used for handling GDK events. A new source type is created by *deriving* from the GSource structure. The derived type of source is represented by a structure that has the GSource structure as a first element, and other elements specific to the new source type. To create an instance of the new source type, call `g_source_new()` passing in the size of the derived structure and a table of functions. These GSourceFuncs determine the behavior of the new source types.

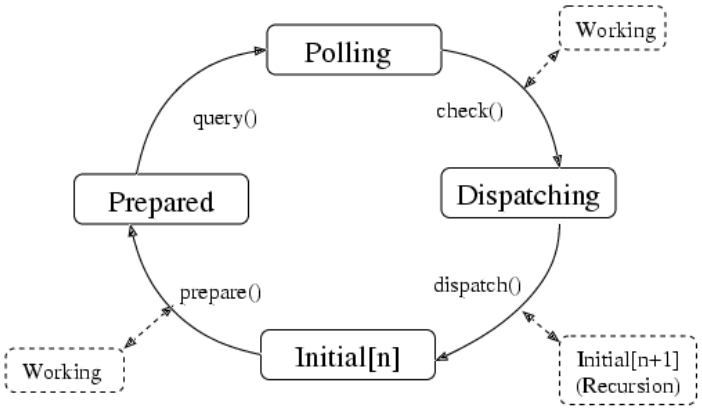
New source types basically interact with with the main context in two ways. Their prepare function in GSourceFuncs can set a timeout to determine the maximum amount of time that the main loop will sleep before checking the source again. In addition, or as well, the source can add file descriptors to the set that the main context checks using `g_source_add_poll()`.

Customizing the main loop iteration

Single iterations of a GMainContext can be run with `g_main_context_iteration()`. In some cases, more detailed control of exactly how the details of the main loop work is desired, for instance, when integrating the GMainLoop with an external main loop. In such cases, you can call the component functions of `g_main_context_iteration()` directly. These functions are `g_main_context_prepare()`, `g_main_context_query()`, `g_main_context_check()` and `g_main_context_dispatch()`.

The operation of these functions can best be seen in terms of a state diagram, as shown in Figure 1, “States of a Main Context”.

Figure 1. States of a Main Context



Details

GMainLoop

```
typedef struct _GMainLoop GMainLoop;
```

The GMainLoop struct is an opaque data type representing the main event loop of a GLib or GTK+ application.

---

g_main_loop_new ()

```
GMainLoop*  g_main_loop_new                 (GMainContext *context,
                                             gboolean is_running);
```

Creates a new GMainLoop structure.

context : a GMainContext (if NULL, the default context will be used).
is_running : set to TRUE to indicate that the loop is running. This is not very important since calling g_main_loop_run() will set this to TRUE anyway.
Returns : a new GMainLoop.

---

g_main_loop_ref ()

```
GMainLoop*  g_main_loop_ref                 (GMainLoop *loop);
```

Increases the reference count on a GMainLoop object by one.

loop : a GMainLoop
Returns :

loop

---

g_main_loop_unref ()

```
void        g_main_loop_unref               (GMainLoop *loop);
```

Decreases the reference count on a GMainLoop object by one. If the result is zero, free the loop and free all associated memory.

loop : a GMainLoop

---

g_main_loop_run ()

```
void        g_main_loop_run                 (GMainLoop *loop);
```

Runs a main loop until g_main_loop_quit() is called on the loop. If this is called for the thread of the loop's GMainContext, it will process events from the loop, otherwise it will simply wait.

loop : a GMainLoop

---

g_main_loop_quit ()

```
void        g_main_loop_quit                (GMainLoop *loop);
```

Stops a GMainLoop from running. Any calls to g_main_loop_run() for the loop will return.

loop : a GMainLoop

---

g_main_loop_is_running ()

```
gboolean    g_main_loop_is_running          (GMainLoop *loop);
```

Checks to see if the main loop is currently being run via g_main_loop_run().

loop : a GMainLoop.
Returns : TRUE if the mainloop is currently being run.

---

g_main_loop_get_context ()

```
GMainContext* g_main_loop_get_context       (GMainLoop *loop);
```

Returns the GMainContext of *loop*.

loop : a GMainLoop.
Returns : the GMainContext of *loop*

---

g_main_new()

```
#define     g_main_new(is_running)
```

Warning

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

Creates a new GMainLoop for the default main loop.

is_running : set to `TRUE` to indicate that the loop is running. This is not very important since calling `g_main_run()` will set this to `TRUE` anyway.
Returns : a new GMainLoop.

---

g_main_destroy()

```
#define     g_main_destroy(loop)
```

Warning

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

Frees the memory allocated for the GMainLoop.

loop : a GMainLoop.

---

g_main_run()

```
#define     g_main_run(loop)
```

Warning

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

Runs a main loop until it stops running.

loop : a GMainLoop.

---

g_main_quit()

```
#define     g_main_quit(loop)
```

Warning

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

Stops the GMainLoop. If `g_main_run()` was called to run the GMainLoop, it will now return.

loop : a GMainLoop.

---

g_main_is_running()

```
#define     g_main_is_running(loop)
```

Warning

`g_main_is_running` is deprecated and should not be used in newly-written code. USe `g_main_loop_is_running()` instead.

Checks if the main loop is running.

loop : a GMainLoop.
Returns : `TRUE` if the main loop is running.

G_PRIORITY_HIGH

```
#define G_PRIORITY_HIGH            -100
```

Use this for high priority event sources. It is not used within GLib or GTK+.

G_PRIORITY_DEFAULT

```
#define G_PRIORITY_DEFAULT          0
```

Use this for default priority event sources. In GLib this priority is used when adding timeout functions with `g_timeout_add()`. In GDK this priority is used for events from the X server.

G_PRIORITY_HIGH_IDLE

```
#define G_PRIORITY_HIGH_IDLE        100
```

Use this for high priority idle functions. GTK+ uses G_PRIORITY_HIGH_IDLE + 10 for resizing operations, and G_PRIORITY_HIGH_IDLE + 20 for redrawing operations. (This is done to ensure that any pending resizes are processed before any pending redraws, so that widgets are not redrawn twice unnecessarily.)

G_PRIORITY_DEFAULT_IDLE

```
#define G_PRIORITY_DEFAULT_IDLE     200
```

Use this for default priority idle functions. In GLib this priority is used when adding idle functions with `g_idle_add()`.

G_PRIORITY_LOW

```
#define G_PRIORITY_LOW              300
```

Use this for very low priority background tasks. It is not used within GLib or GTK+.

GMainContext

```
typedef struct _GMainContext GMainContext;
```

The GMainContext struct is an opaque data type representing a set of sources to be handled in a main loop.

---

g_main_context_new ()

```
GMainContext* g_main_context_new            (void);
```

Creates a new GMainContext strcuture

Returns : the new GMainContext

---

g_main_context_ref ()

```
GMainContext* g_main_context_ref            (GMainContext *context);
```

Increases the reference count on a GMainContext object by one.

context : a GMainContext
Returns : the *context* that was passed in (since 2.6)

---

g_main_context_unref ()

```
void        g_main_context_unref            (GMainContext *context);
```

Decreases the reference count on a GMainContext object by one. If the result is zero, free the context and free all associated memory.

context : a GMainContext

---

g_main_context_default ()

```
GMainContext* g_main_context_default        (void);
```

Returns the default main context. This is the main context used for main loop functions when a main loop is not explicitly specified.

Returns : the default main context.

---

g_main_context_iteration ()

```
gboolean    g_main_context_iteration        (GMainContext *context,
                                             gboolean may_block);
```

Runs a single iteration for the given main loop. This involves checking to see if any event sources are ready to be processed, then if no events sources are ready and *may_block* is TRUE, waiting for a source to become ready, then dispatching the highest priority events sources that are ready. Note that even when *may_block* is TRUE, it is still possible for `g_main_context_iteration()` to return FALSE, since the the wait may be interrupted for other reasons than an event source becoming ready.

context : a GMainContext (if NULL, the default context will be used)
may_block : whether the call may block.
Returns : TRUE if events were dispatched.

---

g_main_iteration()

```
#define     g_main_iteration(may_block)
```

Warning

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

Runs a single iteration for the default GMainContext.

may_block : set to `TRUE` if it should block (i.e. wait) until an event source becomes ready. It will return after an event source has been processed. If set to `FALSE` it will return immediately if no event source is ready to be processed.
Returns : `TRUE` if more events are pending.

---

g_main_context_pending ()

```
gboolean    g_main_context_pending          (GMainContext *context);
```

Checks if any sources have pending events for the given context.

context : a GMainContext (if `NULL`, the default context will be used)
Returns : `TRUE` if events are pending.

---

g_main_pending()

```
#define     g_main_pending()
```

Warning

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

Checks if any events are pending for the default GMainContext (i.e. ready to be processed).

Returns : `TRUE` if any events are pending.

---

g_main_context_find_source_by_id ()

```
GSource*    g_main_context_find_source_by_id
                                        (GMainContext *context,
                                         guint source_id);
```


Finds a GSource given a pair of context and ID

context : a GMainContext (if `NULL`, the default context will be used)
source_id : the source ID, as returned by `g_source_get_id()`
Returns : the GSource if found, otherwise, `NULL`

---

g_main_context_find_source_by_user_data ()

```
GSource*    g_main_context_find_source_by_user_data
                                        (GMainContext *context,
                                         gpointer user_data);
```

Finds a source with the given user data for the callback. If multiple sources exist with the same user data, the first one found will be returned.

context : a GMainContext
user_data : the user_data for the callback.
Returns : the source, if one was found, otherwise `NULL`

---

g_main_context_find_source_by_funcs_user_data ()

```
GSource*    g_main_context_find_source_by_funcs_user_data
                                        (GMainContext *context,
                                         GSourceFuncs *funcs,
                                         gpointer user_data);
```

Finds a source with the given source functions and user data. If multiple sources exist with the same source function and user data, the first one found will be returned.

context : a GMainContext (if `NULL`, the default context will be used).
funcs : the *source_funcs* passed to `g_source_new()`.
user_data : the user data from the callback.
Returns : the source, if one was found, otherwise `NULL`

---

g_main_context_wakeup ()

```
void        g_main_context_wakeup           (GMainContext *context);
```

If *context* is currently waiting in a `poll()`, interrupt the `poll()`, and continue the iteration process.

context : a GMainContext

---

g_main_context_acquire ()

```
gboolean    g_main_context_acquire          (GMainContext *context);
```

Tries to become the owner of the specified context. If some other context is the owner of the context, returns FALSE immediately. Ownership is properly recursive: the owner can require ownership again and will release ownership when g_main_context_release() is called as many times as g_main_context_acquire().

You must be the owner of a context before you can call g_main_context_prepare(), g_main_context_query(), g_main_context_check(), g_main_context_dispatch().

context : a GMainContext
Returns : TRUE if the operation succeeded, and this thread is now the owner of *context*.

---

g_main_context_release ()

```
void        g_main_context_release          (GMainContext *context);
```

Releases ownership of a context previously acquired by this thread with g_main_context_acquire(). If the context was acquired multiple times, the only release ownership when g_main_context_release() is called as many times as it was acquired.

context : a GMainContext

---

g_main_context_wait ()

```
gboolean    g_main_context_wait             (GMainContext *context,
                                             GCond *cond,
                                             GMutex *mutex);
```

Tries to become the owner of the specified context, as with g_main_context_acquire(). But if another thread is the owner, atomically drop *mutex* and wait on *cond* until that owner releases ownership or until *cond* is signaled, then try again (once) to become the owner.

context : a GMainContext
cond : a condition variable
mutex : a mutex, currently held
Returns : TRUE if the operation succeeded, and this thread is now the owner of *context*.

---

g_main_context_prepare ()

```
gboolean    g_main_context_prepare          (GMainContext *context,
                                             gint *priority);
```

Prepares to poll sources within a main loop. The resulting information for polling is determined by

calling g_main_context_query().

context : a GMainContext
priority : location to store priority of highest priority source already ready.
Returns : TRUE if some source is ready to be dispatched prior to polling.

---

g_main_context_query ()

```
gint        g_main_context_query            (GMainContext *context,
                                             gint max_priority,
                                             gint *timeout_,
                                             GPollFD *fds,
                                             gint n_fds);
```

Determines information necessary to poll this main loop.

context : a GMainContext
max_priority : maximum priority source to check
*timeout_* : location to store timeout to be used in polling
fds : location to store GPollFD records that need to be polled.
n_fds : length of *fds*.
Returns : the number of records actually stored in *fds*, or, if more than *n_fds* records need to be stored, the number of records that need to be stored.

---

g_main_context_check ()

```
gint        g_main_context_check            (GMainContext *context,
                                             gint max_priority,
                                             GPollFD *fds,
                                             gint n_fds);
```

Passes the results of polling back to the main loop.

context : a GMainContext
max_priority : the maximum numerical priority of sources to check
fds : array of GPollFD's that was passed to the last call to g_main_context_query()
n_fds : return value of g_main_context_query()
Returns : TRUE if some sources are ready to be dispatched.

---

g_main_context_dispatch ()

```
void        g_main_context_dispatch         (GMainContext *context);
```

Dispatches all pending sources.

context : a GMainContext

---

g_main_context_set_poll_func ()

```
void        g_main_context_set_poll_func    (GMainContext *context,
                                             GPollFunc func);
```

Sets the function to use to handle polling of file descriptors. It will be used instead of the `poll()` system call (or GLib's replacement function, which is used where `poll()` isn't available).

This function could possibly be used to integrate the GLib event loop with an external event loop.

context : a GMainContext
func : the function to call to poll all file descriptors

---

g_main_context_get_poll_func ()

```
GPollFunc   g_main_context_get_poll_func    (GMainContext *context);
```

Gets the poll function set by `g_main_context_set_poll_func()`.

context : a GMainContext
Returns : the poll function

GPollFunc ()

```
gint        (*GPollFunc)                    (GPollFD *ufds,
                                             guint nfsd,
                                             gint timeout_);
```

Specifies the type of function passed to `g_main_context_set_poll_func()`. The semantics of the function should match those of the `poll()` system call.

ufds : an array of GPollFD elements.
nfsd : the number of elements in *ufds*.
*timeout_* : the maximum time to wait for an event of the file descriptors. A negative value indicates an infinite timeout.
Returns : the number of GPollFD elements which have events or errors reported, or -1 if an error occurred.

---

g_main_context_add_poll ()


```
void        g_main_context_add_poll         (GMainContext *context,
                                             GPollFD *fd,
                                             gint priority);
```

Adds a file descriptor to the set of file descriptors polled for this context. This will very seldomly be used directly. Instead a typical event source will use `g_source_add_poll()` instead.

context : a GMainContext (or `NULL` for the default context)
fd : a GPollFD structure holding information about a file descriptor to watch.
priority : the priority for this file descriptor which should be the same as the priority used for `g_source_attach()` to ensure that the file descriptor is polled whenever the results may be needed.

---

g_main_context_remove_poll ()

```
void        g_main_context_remove_poll      (GMainContext *context,
                                             GPollFD *fd);
```

Removes file descriptor from the set of file descriptors to be polled for a particular context.

context : a GMainContext
fd : a GPollFD descriptor previously added with `g_main_context_add_poll()`

---

g_main_depth ()

```
int         g_main_depth                    (void);
```

Return value: The main loop recursion level in the current thread

Returns : the depth of the stack of calls to `g_main_context_dispatch()` on any GMainContext in the current thread. That is, when called from the toplevel, it gives 0. When called from within a callback from `g_main_context_iteration()` (or `g_main_loop_run()`, etc.) it returns 1. When called from within a callback to a recursive call to `g_main_context_iterate()`, it returns 2. And so forth. This function is useful in a situation like the following: Imagine an extremely simple "garbage collected" system.

Example 1.

static GList *free_list; gpointer allocate_memory (gsize size) { gpointer result = g_malloc (size); free_list = g_list_prepend (free_list, result); return result; } void free_allocated_memory (void) { GList *l; for (l = free_list; l; l = l->next); g_free (l->data); g_list_free (free_list); free_list = NULL; } [...] while (TRUE); { g_main_context_iteration (NULL, TRUE); `free_allocated_memory()`; } This works from an application, however, if you want to do the same thing from a library, it gets more difficult, since you no longer control the main loop. You might think you can simply use an idle function to make the call to

`free_allocated_memory()`, but that doesn't work, since the idle function could be called from a recursive callback. This can be fixed by using `g_main_depth()`

Example 2.

gpointer allocate_memory (gsize size) { FreeListBlock *block = g_new (FreeListBlock, 1);\ block->mem = g_malloc (size); block->depth = `g_main_depth()`; free_list = g_list_prepend (free_list, block); return block->mem; } void free_allocated_memory (void) { GList *l; int depth = `g_main_depth()`; for (l = free_list; l;); { GList *next = l->next; FreeListBlock *block = l->data; if (block->depth > depth) { g_free (block->mem); g_free (block); free_list = g_list_delete_link (free_list, l); } l = next; } }
There is a temptation to use `g_main_depth()` to solve problems with reentrancy. For instance, while waiting for data to be received from the network in response to a menu item, the menu item might be selected again. It might seem that one could make the menu item's callback return immediately and do nothing if `g_main_depth()` returns a value greater than 1. However, this should be avoided since the user then sees selecting the menu item do nothing. Furthermore, you'll find yourself adding these checks all over your code, since there are doubtless many, many things that the user could do. Instead, you can use the following techniques:

1. Use `gtk_widget_set_sensitive()` or modal dialogs to prevent the user from interacting with elements while the main loop is recursing.

2. Avoid main loop recursion in situations where you can't handle arbitrary callbacks. Instead, structure your code so that you simply return to the main loop and then get called again when there is more work to do.

---

g_main_set_poll_func()

```
#define     g_main_set_poll_func(func)
```

Warning

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

Sets the function to use for the handle polling of file descriptors for the default main context.

`func` : the function to call to poll all file descriptors.

---

g_timeout_source_new ()

```
GSource*    g_timeout_source_new            (guint interval);
```

Creates a new timeout source.

The source will not initially be associated with any GMainContext and must be added to one with `g_source_attach()` before it will be executed.

`interval` : the timeout interval in milliseconds.
Returns : the newly-created timeout source

---

g_timeout_add ()

```
guint       g_timeout_add                   (guint interval,
                                             GSourceFunc function,
                                             gpointer data);
```

Sets a function to be called at regular intervals, with the default priority, G_PRIORITY_DEFAULT. The function is called repeatedly until it returns `FALSE`, at which point the timeout is automatically destroyed and the function will not be called again. The first call to the function will be at the end of the first *`interval`*.

Note that timeout functions may be delayed, due to the processing of other event sources. Thus they should not be relied on for precise timing. After each call to the timeout function, the time of the next timeout is recalculated based on the current time and the given interval (it does not try to 'catch up' time lost in delays).

`interval` : the time between calls to the function, in milliseconds (1/1000ths of a second)
`function` : function to call
`data` : data to pass to *`function`*
Returns : the id of event source.

---

g_timeout_add_full ()

```
guint       g_timeout_add_full              (gint priority,
                                             guint interval,
                                             GSourceFunc function,
                                             gpointer data,
                                             GDestroyNotify notify);
```

Sets a function to be called at regular intervals, with the given priority. The function is called repeatedly until it returns `FALSE`, at which point the timeout is automatically destroyed and the function will not be called again. The *`notify`* function is called when the timeout is destroyed. The first call to the function will be at the end of the first *`interval`*.

Note that timeout functions may be delayed, due to the processing of other event sources. Thus they should not be relied on for precise timing. After each call to the timeout function, the time of the next timeout is recalculated based on the current time and the given interval (it does not try to 'catch up' time lost in delays).

`priority` : the priority of the idle source. Typically this will be in the range between G_PRIORITY_DEFAULT_IDLE and G_PRIORITY_HIGH_IDLE.
`interval` : the time between calls to the function, in milliseconds (1/1000ths of a second)

function : function to call
data : data to pass to *function*
notify : function to call when the idle is removed, or NULL
Returns : the id of event source.

---

g_idle_source_new ()

```
GSource*    g_idle_source_new               (void);
```

Creates a new idle source.

The source will not initially be associated with any GMainContext and must be added to one with g_source_attach() before it will be executed. Note that the default priority for idle sources is G_PRIORITY_DEFAULT_IDLE, as compared to other sources which have a default priority of G_PRIORITY_DEFAULT.

Returns : the newly-created idle source

---

g_idle_add ()

```
guint       g_idle_add                      (GSourceFunc function,
                                             gpointer data);
```

Adds a function to be called whenever there are no higher priority events pending to the default main loop. The function is given the default idle priority, G_PRIORITY_DEFAULT_IDLE. If the function returns FALSE it is automatically removed from the list of event sources and will not be called again.

function : function to call
data : data to pass to *function*.
Returns : the id of the event source.

---

g_idle_add_full ()

```
guint       g_idle_add_full                 (gint priority,
                                             GSourceFunc function,
                                             gpointer data,
                                             GDestroyNotify notify);
```

Adds a function to be called whenever there are no higher priority events pending. If the function returns FALSE it is automatically removed from the list of event sources and will not be called again.

priority : the priority of the idle source. Typically this will be in the range btweeen G_PRIORITY_DEFAULT_IDLE and G_PRIORITY_HIGH_IDLE.
function : function to call

data : data to pass to *function*
notify : function to call when the idle is removed, or NULL
Returns : the id of the event source.

---

g_idle_remove_by_data ()

```
gboolean    g_idle_remove_by_data           (gpointer data);
```

Removes the idle function with the given data.

data : the data for the idle source's callback.
Returns : TRUE if an idle source was found and removed.

GPid

```
typedef int GPid;
```

A type which is used to hold a process identification. On Unix, processes are identified by a process id (an integer), while Windows uses process handles (which are pointers).

GChildWatchFunc ()

```
void        (*GChildWatchFunc)              (GPid pid,
                                             gint status,
                                             gpointer data);
```

The type of functions to be called when a child exists.

pid : the process id of the child process
status : Status information about the child process, see waitpid(2) for more information about this field
data : user data passed to g_child_watch_add()

---

g_child_watch_source_new ()

```
GSource*    g_child_watch_source_new        (GPid pid);
```

Creates a new child_watch source.

The source will not initially be associated with any GMainContext and must be added to one with g_source_attach() before it will be executed.

Note that on platforms where GPid must be explicitely closed (see g_spawn_close_pid()) *pid* must not be closed while the source is still active. Typically, you will want to call g_spawn_close_pid() in the callback function for the source.

pid : process id of a child process to watch. On Windows, a HANDLE for the process to watch (which actually doesn't have to be a child).

Returns : the newly-created child watch source

Since 2.4

---

g_child_watch_add ()

```
guint       g_child_watch_add               (GPid pid,
                                             GChildWatchFunc function,
                                             gpointer data);
```

Sets a function to be called when the child indicated by *pid* exits, at a default priority, G_PRIORITY_DEFAULT.

Note that on platforms where GPid must be explicitely closed (see g_spawn_close_pid()) *pid* must not be closed while the source is still active. Typically, you will want to call g_spawn_close_pid() in the callback function for the source.

GLib supports only a single callback per process id.

pid : process id of a child process to watch

function : function to call

data : data to pass to *function*

Returns : the id of event source.

Since 2.4

---

g_child_watch_add_full ()

```
guint       g_child_watch_add_full          (gint priority,
                                             GPid pid,
                                             GChildWatchFunc function,
                                             gpointer data,
                                             GDestroyNotify notify);
```

Sets a function to be called when the child indicated by *pid* exits, at a default priority, G_PRIORITY_DEFAULT.

Note that on platforms where GPid must be explicitely closed (see g_spawn_close_pid()) *pid* must not be closed while the source is still active. Typically, you will want to call g_spawn_close_pid() in the callback function for the source.

GLib supports only a single callback per process id.

priority : the priority of the idle source. Typically this will be in the range between G_PRIORITY_DEFAULT_IDLE and G_PRIORITY_HIGH_IDLE.

pid : process id of a child process to watch

function : function to call

data : data to pass to *function*

notify : function to call when the idle is removed, or NULL

Returns : the id of event source.

Since 2.4

GPollFD

```
typedef struct {
  gint          fd;
  gushort       events;
  gushort       revents;
} GPollFD;
```

gint fd; the file descriptor to poll (or a HANDLE on Win32 platforms).

gushort events; a bitwise combination of flags from GIOCondition, specifying which events should be polled for. Typically for reading from a file descriptor you would use G_IO_IN | G_IO_HUP | G_IO_ERR, and for writing you would use G_IO_OUT | G_IO_ERR.

gushort revents; a bitwise combination of flags from GIOCondition, returned from the poll() function to indicate which events occurred.

GSource

```
typedef struct {
} GSource;
```

The GSource struct is an opaque data type representing an event source.

GSourceDummyMarshal ()

```
void        (*GSourceDummyMarshal)          (void);
```

This is just a placeholder for GClosureMarshal, which cannot be used here for dependency reasons.

GSourceFuncs

```
typedef struct {
  gboolean (*prepare)  (GSource    *source,
                        gint       *timeout_);
  gboolean (*check)    (GSource    *source);
  gboolean (*dispatch) (GSource    *source,
                        GSourceFunc callback,
                        gpointer    user_data);
  void     (*finalize) (GSource    *source); /* Can be NULL */

  /* For use by g_source_set_closure */
  GSourceFunc     closure_callback;
  GSourceDummyMarshal closure_marshal; /* Really is of type GClosureMarshal */
} GSourceFuncs;
```

The GSourceFuncs struct contains a table of functions used to handle event sources in a generic manner.

prepare	Called before all the file descriptors are polled. If the source can determine that it is ready here (without waiting for the results of the `poll()` call) it should return `TRUE`. It can also return a *`timeout_`* value which should be the maximum timeout (in milliseconds) which should be passed to the `poll()` call. The actual timeout used will be -1 if all sources returned -1, or it will be the minimum of all the *`timeout_`* values returned which were >= 0.
check	Called after all the file descriptors are polled. The source should return `TRUE` if it is ready to be dispatched. Note that some time may have passed since the previous prepare function was called, so the source should be checked again here.
dispatch	Called to dispatch the event source, after it has returned `TRUE` in either its *`prepare`* or its *`check`* function. The *`dispatch`* function is passed in a callback function and data. The callback function may be `NULL` if the source was never connected to a callback using `g_source_set_callback()`. The *`dispatch`* function should call the callback function with *`user_data`* and whatever additional parameters are needed for this type of event source.
finalize	Called when the source is finalized.

For idle sources, the prepare and check functions always return `TRUE` to indicate that the source is always ready to be processed. The prepare function also returns a timeout value of 0 to ensure that the `poll()` call doesn't block (since that would be time wasted which could have been spent running the idle function).

For timeout sources, the prepare and check functions both return `TRUE` if the timeout interval has expired. The prepare function also returns a timeout value to ensure that the `poll()` call doesn't block too long and miss the next timeout.

For file descriptor sources, the prepare function typically returns `FALSE`, since it must wait until `poll()` has been called before it knows whether any events need to be processed. It sets the returned timeout to -1 to indicate that it doesn't mind how long the `poll()` call blocks. In the check function, it tests the results of the `poll()` call to see if the required condition has been met, and returns `TRUE` if so.

GSourceCallbackFuncs


```
typedef struct {
  void (*ref)   (gpointer     cb_data);
  void (*unref) (gpointer     cb_data);
  void (*get)   (gpointer     cb_data,
                 GSource     *source,
                 GSourceFunc *func,
                 gpointer    *data);
} GSourceCallbackFuncs;
```

The GSourceCallbackFuncs struct contains functions for managing callback objects.

`ref` ()	Called when a reference is added to the callback object.
`unref` ()	Called when a reference to the callback object is dropped.
`get` ()	Called to extract the callback function and data from the callback object.

---

g_source_new ()

```
GSource*    g_source_new                    (GSourceFuncs *source_funcs,
                                             guint struct_size);
```

Creates a new GSource structure. The size is specified to allow creating structures derived from GSource that contain additional data. The size passed in must be at least `sizeof (GSource)`.

The source will not initially be associated with any GMainContext and must be added to one with `g_source_attach()` before it will be executed.

`source_funcs` :	structure containing functions that implement the sources behavior.
`struct_size` :	size of the GSource structure to create.
Returns :	the newly-created GSource.

---

g_source_ref ()

```
GSource*    g_source_ref                    (GSource *source);
```

Increases the reference count on a source by one.

`source` : a GSource
Returns : *`source`*

---

g_source_unref ()

```
void        g_source_unref                  (GSource *source);
```

Decreases the reference count of a source by one. If the resulting reference count is zero the source and associated memory will be destroyed.

source : a GSource

---

g_source_attach ()

```
guint       g_source_attach                 (GSource *source,
                                             GMainContext *context);
```

Adds a GSource to a *context* so that it will be executed within that context.

source : a GSource
context : a GMainContext (if NULL, the default context will be used)
Returns : the ID for the source within the GMainContext

---

g_source_destroy ()

```
void        g_source_destroy                (GSource *source);
```

Removes a source from its GMainContext, if any, and mark it as destroyed. The source cannot be subsequently added to another context.

source : a GSource

---

g_source_set_priority ()

```
void        g_source_set_priority           (GSource *source,
                                             gint priority);
```

Sets the priority of a source. While the main loop is being run, a source will be dispatched if it is ready to be dispatched and no sources at a higher (numerically smaller) priority are ready to be dispatched.

source : a GSource
priority : the new priority.

---

g_source_get_priority ()

```
gint        g_source_get_priority           (GSource *source);
```

Gets the priority of a source.

source : a GSource
Returns : the priority of the source

---

g_source_set_can_recurse ()

```
void        g_source_set_can_recurse        (GSource *source,
                                             gboolean can_recurse);
```

Sets whether a source can be called recursively. If *can_recurse* is TRUE, then while the source is being dispatched then this source will be processed normally. Otherwise, all processing of this source is blocked until the dispatch function returns.

source : a GSource
can_recurse : whether recursion is allowed for this source

---

g_source_get_can_recurse ()

```
gboolean    g_source_get_can_recurse        (GSource *source);
```

Checks whether a source is allowed to be called recursively. see g_source_set_can_recurse().

source : a GSource
Returns : whether recursion is allowed.

---

g_source_get_id ()

```
guint       g_source_get_id                 (GSource *source);
```

Returns the numeric ID for a particular source. The ID of a source is unique within a particular main loop context. The reverse mapping from ID to source is done by g_main_context_find_source_by_id().

source : a GSource
Returns : the ID for the source

---

g_source_get_context ()

```
GMainContext* g_source_get_context          (GSource *source);
```

Gets the GMainContext with which the source is associated. Calling this function on a destroyed source is an error.

source : a GSource

Returns : the GMainContext with which the source is associated, or NULL if the context has not yet been added to a source.

---

g_source_set_callback ()

```
void        g_source_set_callback           (GSource *source,
                                             GSourceFunc func,
                                             gpointer data,
                                             GDestroyNotify notify);
```

Sets the callback function for a source. The callback for a source is called from the source's dispatch function.

The exact type of *func* depends on the type of source; ie. you should not count on *func* being called with *data* as its first parameter.

Typically, you won't use this function. Instead use functions specific to the type of source you are using.

source : the source

func : a callback function

data : the data to pass to callback function

notify : a function to call when *data* is no longer in use, or NULL.

GSourceFunc ()

```
gboolean    (*GSourceFunc)                  (gpointer data);
```

Specifies the type of function passed to g_timeout_add(), g_timeout_add_full(), g_idle_add(), and g_idle_add_full().

data : data passed to the function, set when the source was created with one of the above functions.

Returns : it should return FALSE if the source should be removed.

---

g_source_set_callback_indirect ()

```
void        g_source_set_callback_indirect  (GSource *source,
                                             gpointer callback_data,
                                             GSourceCallbackFuncs *callback_func
```

Sets the callback function storing the data as a refcounted callback "object". This is used internally. Note that calling g_source_set_callback_indirect() assumes an initial reference count on *callback_data*, and thus *callback_funcs->unref* will eventually be called once more than

callback_funcs->ref.

source : the source

callback_data : pointer to callback data "object"

callback_funcs : functions for reference counting *callback_data* and getting the callback and data

---

g_source_add_poll ()

```
void        g_source_add_poll               (GSource *source,
                                             GPollFD *fd);
```

Adds a file descriptor to the set of file descriptors polled for this source. This is usually combined with g_source_new() to add an event source. The event source's check function will typically test the *revents* field in the GPollFD struct and return TRUE if events need to be processed.

source : a GSource

fd : a GPollFD structure holding information about a file descriptor to watch.

---

g_source_remove_poll ()

```
void        g_source_remove_poll            (GSource *source,
                                             GPollFD *fd);
```

Removes a file descriptor from the set of file descriptors polled for this source.

source : a GSource

fd : a GPollFD structure previously passed to g_source_add_poll().

---

g_source_get_current_time ()

```
void        g_source_get_current_time       (GSource *source,
                                             GTimeVal *timeval);
```

Gets the "current time" to be used when checking this source. The advantage of calling this function over calling g_get_current_time() directly is that when checking multiple sources, GLib can cache a single value instead of having to repeatedly get the system time.

source : a GSource

timeval : GTimeVal structure in which to store current time.

---

g_source_remove ()

```
gboolean    g_source_remove                 (guint tag);
```

Removes the source with the given id from the default main context. The id of a GSource is given by g_source_get_id(), or will be returned by the functions g_source_attach(), g_idle_add(), g_idle_add_full(), g_timeout_add(), g_timeout_add_full(), g_child_watch_add(), g_child_watch_add_full(), g_io_add_watch(), and g_io_add_watch_full().

See also g_source_destroy().

tag : the id of the source to remove.
Returns : TRUE if the source was found and removed.

---

g_source_remove_by_funcs_user_data ()

```
gboolean    g_source_remove_by_funcs_user_data
                                        (GSourceFuncs *funcs,
                                         gpointer user_data);
```

Removes a source from the default main loop context given the source functions and user data. If multiple sources exist with the same source functions and user data, only one will be destroyed.

funcs : The *source_funcs* passed to g_source_new()
user_data : the user data for the callback
Returns : TRUE if a source was found and removed.

---

g_source_remove_by_user_data ()

```
gboolean    g_source_remove_by_user_data    (gpointer user_data);
```

Removes a source from the default main loop context given the user data for the callback. If multiple sources exist with the same user data, only one will be destroyed.

user_data : the user_data for the callback.
Returns : TRUE if a source was found and removed.

<< GLib Core Application Support **Threads >>**

Threads

Threads — thread abstraction; including threads, different mutexes, conditions and thread private data.

Synopsis

```
#include <glib.h>


#define     G_THREADS_ENABLED
#define     G_THREADS_IMPL_POSIX
#define     G_THREADS_IMPL_SOLARIS
#define     G_THREADS_IMPL_NONE

#define     G_THREAD_ERROR
enum        GThreadError;

            GThreadFunctions;
void        g_thread_init                   (GThreadFunctions *vtable);
gboolean    g_thread_supported              ();

gpointer    (*GThreadFunc)                  (gpointer data);
enum        GThreadPriority;
            GThread;
GThread*    g_thread_create                 (GThreadFunc func,
                                             gpointer data,
                                             gboolean joinable,
                                             GError **error);
GThread*    g_thread_create_full            (GThreadFunc func,
                                             gpointer data,
                                             gulong stack_size,
                                             gboolean joinable,
                                             gboolean bound,
                                             GThreadPriority priority,
                                             GError **error);
GThread*    g_thread_self                   (void);
gpointer    g_thread_join                   (GThread *thread);
void        g_thread_set_priority           (GThread *thread,
                                             GThreadPriority priority);
void        g_thread_yield                  ();
void        g_thread_exit                   (gpointer retval);

            GMutex;
GMutex*     g_mutex_new                     ();
void        g_mutex_lock                    (GMutex *mutex);
gboolean    g_mutex_trylock                 (GMutex *mutex);
void        g_mutex_unlock                  (GMutex *mutex);
void        g_mutex_free                    (GMutex *mutex);

            GStaticMutex;
#define     G_STATIC_MUTEX_INIT
void        g_static_mutex_init             (GStaticMutex *mutex);
void        g_static_mutex_lock             (GStaticMutex *mutex);
```



```
gboolean    g_static_mutex_trylock          (GStaticMutex *mutex);
void        g_static_mutex_unlock           (GStaticMutex *mutex);
GMutex*     g_static_mutex_get_mutex        (GStaticMutex *mutex);
void        g_static_mutex_free             (GStaticMutex *mutex);

#define     G_LOCK_DEFINE                   (name)
#define     G_LOCK_DEFINE_STATIC            (name)
#define     G_LOCK_EXTERN                   (name)
#define     G_LOCK                          (name)
#define     G_TRYLOCK                       (name)
#define     G_UNLOCK                        (name)

            GStaticRecMutex;
#define     G_STATIC_REC_MUTEX_INIT
void        g_static_rec_mutex_init         (GStaticRecMutex *mutex);
void        g_static_rec_mutex_lock         (GStaticRecMutex *mutex);
gboolean    g_static_rec_mutex_trylock      (GStaticRecMutex *mutex);
void        g_static_rec_mutex_unlock       (GStaticRecMutex *mutex);
void        g_static_rec_mutex_lock_full    (GStaticRecMutex *mutex,
                                             guint depth);
guint       g_static_rec_mutex_unlock_full  (GStaticRecMutex *mutex);
void        g_static_rec_mutex_free         (GStaticRecMutex *mutex);

            GStaticRWLock;
#define     G_STATIC_RW_LOCK_INIT
void        g_static_rw_lock_init           (GStaticRWLock *lock);
void        g_static_rw_lock_reader_lock    (GStaticRWLock *lock);
gboolean    g_static_rw_lock_reader_trylock (GStaticRWLock *lock);
void        g_static_rw_lock_reader_unlock  (GStaticRWLock *lock);
void        g_static_rw_lock_writer_lock    (GStaticRWLock *lock);
gboolean    g_static_rw_lock_writer_trylock (GStaticRWLock *lock);
void        g_static_rw_lock_writer_unlock  (GStaticRWLock *lock);
void        g_static_rw_lock_free           (GStaticRWLock *lock);

            GCond;
GCond*      g_cond_new                      ();
void        g_cond_signal                   (GCond *cond);
void        g_cond_broadcast                (GCond *cond);
void        g_cond_wait                     (GCond *cond,
                                             GMutex *mutex);
gboolean    g_cond_timed_wait               (GCond *cond,
                                             GMutex *mutex,
                                             GTimeVal *abs_time);
void        g_cond_free                     (GCond *cond);

            GPrivate;
GPrivate*   g_private_new                   (GDestroyNotify destructor);
gpointer    g_private_get                   (GPrivate *private_key);
void        g_private_set                   (GPrivate *private_key,
                                             gpointer data);

            GStaticPrivate;
#define     G_STATIC_PRIVATE_INIT
void        g_static_private_init           (GStaticPrivate *private_key);
gpointer    g_static_private_get            (GStaticPrivate *private_key);
void        g_static_private_set            (GStaticPrivate *private_key,
                                             gpointer data,
                                             GDestroyNotify notify);
void        g_static_private_free           (GStaticPrivate *private_key);

            GOnce;
enum        GOnceStatus;
#define     G_ONCE_INIT
#define     g_once                          (once, func, arg)
```

Description

Threads act almost like processes, but unlike processes all threads of one process share the same memory. This is good, as it provides easy communication between the involved threads via this shared memory, and it is bad, because strange things (so called Heisenbugs) might happen, when the program is not carefully designed. Especially bad is, that due to the concurrent nature of threads no assumptions on the order of execution of different threads can be done unless explicitly forced by the programmer through synchronization primitives.

The aim of the thread related functions in GLib is to provide a portable means for writing multi-threaded software. There are primitives for mutexes to protect the access to portions of memory (GMutex, GStaticMutex, G_LOCK_DEFINE, GStaticRecMutex and GStaticRWLock), there are primitives for condition variables to allow synchronization of threads (GCond) and finally there are primitives for thread-private data, that every thread has a private instance of (GPrivate, GStaticPrivate). Last but definitely not least there are primitives to portably create and manage threads (GThread).

Details

G_THREADS_ENABLED

```
#define G_THREADS_ENABLED
```

This macro is defined, if GLib was compiled with thread support. This does not necessarily mean, that there is a thread implementation available, but the infrastructure is in place and once you provide a thread implementation to `g_thread_init()`, GLib will be multi-thread safe. It isn't and cannot be, if G_THREADS_ENABLED is not defined.

G_THREADS_IMPL_POSIX

```
#define G_THREADS_IMPL_POSIX
```

This macro is defined, if POSIX style threads are used.

G_THREADS_IMPL_SOLARIS

```
#define G_THREADS_IMPL_SOLARIS
```

This macro is defined, if the Solaris thread system is used.

G_THREADS_IMPL_NONE


```
#define G_THREADS_IMPL_NONE
```

This macro is defined, if no thread implementation is used. You can however provide one to `g_thread_init()` to make GLib multi-thread safe.

G_THREAD_ERROR

```
#define G_THREAD_ERROR g_thread_error_quark ()
```

The error domain of the GLib thread subsystem.

---

enum GThreadError

```
typedef enum
{
  G_THREAD_ERROR_AGAIN /* Resource temporarily unavailable */
} GThreadError;
```

Possible errors of thread related functions.

`G_THREAD_ERROR_AGAIN`	a thread couldn't be created due to resource shortage. Try again later.

GThreadFunctions

```
typedef struct {
  GMutex*  (*mutex_new)          (void);
  void     (*mutex_lock)         (GMutex               *mutex);
  gboolean (*mutex_trylock)      (GMutex               *mutex);
  void     (*mutex_unlock)       (GMutex               *mutex);
  void     (*mutex_free)         (GMutex               *mutex);
  GCond*   (*cond_new)           (void);
  void     (*cond_signal)        (GCond                *cond);
  void     (*cond_broadcast)     (GCond                *cond);
  void     (*cond_wait)          (GCond                *cond,
                                  GMutex               *mutex);
  gboolean (*cond_timed_wait)    (GCond                *cond,
                                  GMutex               *mutex,
                                  GTimeVal             *end_time);
  void      (*cond_free)         (GCond                *cond);
  GPrivate* (*private_new)       (GDestroyNotify        destructor);
  gpointer  (*private_get)       (GPrivate             *private_key);
  void      (*private_set)       (GPrivate             *private_key,
                                  gpointer              data);
  void      (*thread_create)     (GThreadFunc          func,
                                  gpointer             data,
                                  gulong               stack_size,
                                  gboolean             joinable,
                                  gboolean             bound,
```

```
                                GThreadPriority       priority,
                                gpointer              thread,
                                GError              **error);
  void      (*thread_yield)     (void);
  void      (*thread_join)      (gpointer              thread);
  void      (*thread_exit)      (void);
  void      (*thread_set_priority)(gpointer           thread,
                                GThreadPriority       priority);
  void      (*thread_self)      (gpointer              thread);
  gboolean  (*thread_equal)     (gpointer              thread1,
                                gpointer              thread2);
} GThreadFunctions;
```

This function table is used by g_thread_init() to initialize the thread system. The functions in that table are directly used by their g_* prepended counterparts, that are described here, e.g. if you call g_mutex_new() then mutex_new() from the table provided to g_thread_init() will be called.

Note

This struct should only be used, if you know, what you are doing.

---

g_thread_init ()

```
void        g_thread_init                   (GThreadFunctions *vtable);
```

If you use GLib from more than one thread, you must initialize the thread system by calling g_thread_init(). Most of the time you will only have to call g_thread_init (NULL).

Note

You should only call g_thread_init() with a non-NULL parameter if you really know what you are doing.

Note

g_thread_init() must not be called directly or indirectly as a callback from GLib. Also no mutexes may be currently locked, while calling g_thread_init().

g_thread_init() might only be called once. On the second call it will abort with an error. If you want to make sure, that the thread system is initialized, you can do that too:

```
if (!g_thread_supported ()) g_thread_init (NULL);
```

After that line either the thread system is initialized or the program will abort, if no thread system is available in GLib, i.e. either G_THREADS_ENABLED is not defined or G_THREADS_IMPL_NONE is defined.

If no thread system is available and *vtable* is NULL or if not all elements of *vtable* are non-NULL, then g_thread_init() will abort.

Note

To use g_thread_init() in your program, you have to link with the libraries that the command **pkg-config --libs gthread-2.0** outputs. This is not the case for all the other thread related functions of GLib. Those can be used without having to link with the thread libraries.

vtable : a function table of type GThreadFunctions, that provides the entry points to the thread system to be used.

---

g_thread_supported ()

```
gboolean    g_thread_supported              ();
```

This function returns, whether the thread system is initialized or not.

Note

This function is actually a macro. Apart from taking the address of it you can however use it as if it was a function.

Returns : TRUE, if the thread system is initialized.

GThreadFunc ()

```
gpointer    (*GThreadFunc)                  (gpointer data);
```

Specifies the type of the *func* functions passed to g_thread_create() or g_thread_create_full().

data : data passed to the thread.
Returns : the return value of the thread, which will be returned by g_thread_join().

---

enum GThreadPriority

```
typedef enum
{
  G_THREAD_PRIORITY_LOW,
  G_THREAD_PRIORITY_NORMAL,
  G_THREAD_PRIORITY_HIGH,
  G_THREAD_PRIORITY_URGENT
} GThreadPriority;
```

Specifies the priority of a thread.

Note

It is not guaranteed, that threads with different priorities really behave accordingly.

On some systems (e.g. Linux) there are no thread priorities. On other systems (e.g. Solaris) there doesn't seem to be different scheduling for different priorities. All in all try to avoid being dependent on priorities.

`G_THREAD_PRIORITY_LOW`	a priority lower than normal
`G_THREAD_PRIORITY_NORMAL`	the default priority
`G_THREAD_PRIORITY_HIGH`	a priority higher than normal
`G_THREAD_PRIORITY_URGENT`	the highest priority

GThread

```
typedef struct {
} GThread;
```

The GThread struct represents a running thread. It has three public read-only members, but the underlying struct is bigger, so you must not copy this struct.

Note

Resources for a joinable thread are not fully released until `g_thread_join()` is called for that thread.

---

g_thread_create ()

```
GThread*    g_thread_create                 (GThreadFunc func,
                                             gpointer data,
                                             gboolean joinable,
                                             GError **error);
```

This function creates a new thread with the default priority.

If *`joinable`* is `TRUE`, you can wait for this threads termination calling `g_thread_join()`. Otherwise the thread will just disappear, when ready.

The new thread executes the function *`func`* with the argument *`data`*. If the thread was created successfully, it is returned.

`error` can be `NULL` to ignore errors, or non-`NULL` to report errors. The error is set, if and only if the function returns `NULL`.

`func` :	a function to execute in the new thread.
`data` :	an argument to supply to the new thread.
`joinable` :	should this thread be joinable?
`error` :	return location for error.
Returns :	the new GThread on success.

---

g_thread_create_full ()

```
GThread*    g_thread_create_full            (GThreadFunc func,
                                             gpointer data,
                                             gulong stack_size,
                                             gboolean joinable,
                                             gboolean bound,
                                             GThreadPriority priority,
                                             GError **error);
```

This function creates a new thread with the priority *`priority`*. The stack gets the size *`stack_size`* or the default value for the current platform, if *`stack_size`* is 0.

If *`joinable`* is `TRUE`, you can wait for this threads termination calling `g_thread_join()`. Otherwise the thread will just disappear, when ready. If *`bound`* is `TRUE`, this thread will be scheduled in the system scope, otherwise the implementation is free to do scheduling in the process scope. The first variant is more expensive resource-wise, but generally faster. On some systems (e.g. Linux) all threads are bound.

The new thread executes the function *`func`* with the argument *`data`*. If the thread was created successfully, it is returned.

`error` can be `NULL` to ignore errors, or non-`NULL` to report errors. The error is set, if and only if the function returns `NULL`.

Note

It is not guaranteed, that threads with different priorities really behave accordingly. On some systems (e.g. Linux) there are no thread priorities. On other systems (e.g. Solaris) there doesn't seem to be different scheduling for different priorities. All in all try to avoid being dependent on priorities. Use `G_THREAD_PRIORITY_NORMAL` here as a default.

Note

Only use `g_thread_create_full()`, when you really can't use `g_thread_create()` instead. `g_thread_create()` does not take *`stack_size`*, *`bound`* and *`priority`* as arguments, as they should only be used for cases, where it is inevitable.

`func` :	a function to execute in the new thread.
`data` :	an argument to supply to the new thread.
`stack_size` :	a stack size for the new thread.
`joinable` :	should this thread be joinable?
`bound` :	should this thread be bound to a system thread?
`priority` :	a priority for the thread.
`error` :	return location for error.
Returns :	the new GThread on success.

---

g_thread_self ()

```
GThread*    g_thread_self                   (void);
```

This functions returns the GThread corresponding to the calling thread.

Returns : the current thread.

---

g_thread_join ()

```
gpointer    g_thread_join                   (GThread *thread);
```

Waits until *thread* finishes, i.e. the function *func*, as given to g_thread_create(), returns or g_thread_exit() is called by *thread*. All resources of *thread* including the GThread struct are released. *thread* must have been created with *joinable*=TRUE in g_thread_create(). The value returned by *func* or given to g_thread_exit() by *thread* is returned by this function.

thread : a GThread to be waited for.
Returns : the return value of the thread.

---

g_thread_set_priority ()

```
void        g_thread_set_priority           (GThread *thread,
                                             GThreadPriority priority);
```

Changes the priority of *thread* to *priority*.

Note

It is not guaranteed, that threads with different priorities really behave accordingly. On some systems (e.g. Linux) there are no thread priorities. On other systems (e.g. Solaris) there doesn't seem to be different scheduling for different priorities. All in all try to avoid being dependent on priorities.

thread : a GThread.
priority : a new priority for *thread*.

---

g_thread_yield ()

```
void        g_thread_yield                  ();
```

Gives way to other threads waiting to be scheduled.

This function is often used as a method to make busy wait less evil. But in most cases, you will encounter, there are better methods to do that. So in general you shouldn't use that function.

---

g_thread_exit ()

```
void        g_thread_exit                   (gpointer retval);
```

Exits the current thread. If another thread is waiting for that thread using g_thread_join() and the current thread is joinable, the waiting thread will be woken up and getting *retval* as the return value of g_thread_join(). If the current thread is not joinable, *retval* is ignored. Calling

```
g_thread_exit (retval);
```

is equivalent to calling

```
return retval;
```

in the function *func*, as given to g_thread_create().

Note

Never call g_thread_exit() from within a thread of a GThreadPool, as that will mess up the bookkeeping and lead to funny and unwanted results.

retval : the return value of this thread.

GMutex

```
typedef struct _GMutex GMutex;
```

The GMutex struct is an opaque data structure to represent a mutex (mutual exclusion). It can be used to protect data against shared access. Take for example the following function:

Example 3. A function which will not work in a threaded environment

```
int give_me_next_number ()
{
  static int current_number = 0;

  /* now do a very complicated calculation to calculate the new number,
     this might for example be a random number generator */
  current_number = calc_next_number (current_number);
  return current_number;
}
```

It is easy to see, that this won't work in a multi-threaded application. There current_number must be protected against shared access. A first naive implementation would be:

Example 4. The wrong way to write a thread-safe function

```
int give_me_next_number ()
{
  static int current_number = 0;
  int ret_val;
  static GMutex * mutex = NULL;

  if (!mutex)
    mutex = g_mutex_new ();
  g_mutex_lock (mutex);
  ret_val = current_number = calc_next_number (current_number);
  g_mutex_unlock (mutex);
  return ret_val;
}
```

This looks like it would work, but there is a race condition while constructing the mutex and this code cannot work reliable. So please do not use such constructs in your own programs. One working solution is:

Example 5. A correct thread-safe function

```
static GMutex *give_me_next_number_mutex = NULL;

/* this function must be called before any call to give_me_next_number ()
   it must be called exactly once. */
void init_give_me_next_number ()
{
  g_assert (give_me_next_number_mutex == NULL);
  give_me_next_number_mutex = g_mutex_new ();
}

int give_me_next_number ()
{
  static int current_number = 0;
  int ret_val;

  g_mutex_lock (give_me_next_number_mutex);
  ret_val = current_number = calc_next_number (current_number);
  g_mutex_unlock (give_me_next_number_mutex);
  return ret_val;
}
```

GStaticMutex provides a simpler and safer way of doing this.

If you want to use a mutex, but your code should also work without calling `g_thread_init()` first, you can not use a GMutex, as `g_mutex_new()` requires that. Use a GStaticMutex instead.

A GMutex should only be accessed via the following functions.

> **Note**
>
> All of the `g_mutex_*` functions are actually macros. Apart from taking their addresses, you can however use them as if they were functions.

---

g_mutex_new ()


```
GMutex*     g_mutex_new                     ();
```

Creates a new GMutex.

> **Note**
>
> This function will abort, if `g_thread_init()` has not been called yet.

Returns : a new GMutex.

---

g_mutex_lock ()

```
void        g_mutex_lock                    (GMutex *mutex);
```

Locks *`mutex`*. If *`mutex`* is already locked by another thread, the current thread will block until *`mutex`* is unlocked by the other thread.

This function can also be used, if `g_thread_init()` has not yet been called and will do nothing then.

> **Note**
>
> GMutex is neither guaranteed to be recursive nor to be non-recursive, i.e. a thread could deadlock while calling `g_mutex_lock()`, if it already has locked *`mutex`*. Use GStaticRecMutex, if you need recursive mutexes.

`mutex` : a GMutex.

---

g_mutex_trylock ()

```
gboolean    g_mutex_trylock                 (GMutex *mutex);
```

Tries to lock *`mutex`*. If *`mutex`* is already locked by another thread, it immediately returns `FALSE`. Otherwise it locks *`mutex`* and returns `TRUE`.

This function can also be used, if `g_thread_init()` has not yet been called and will immediately return `TRUE` then.

> **Note**
>
> GMutex is neither guaranteed to be recursive nor to be non-recursive, i.e. the return value of `g_mutex_trylock()` could be both `FALSE` or `TRUE`, if the current thread already has locked *`mutex`*. Use GStaticRecMutex, if you need recursive mutexes.

`mutex` : a GMutex.
Returns : `TRUE`, if *`mutex`* could be locked.

g_mutex_unlock ()

```
void        g_mutex_unlock                  (GMutex *mutex);
```

Unlocks *mutex*. If another thread is blocked in a g_mutex_lock() call for *mutex*, it will be woken and can lock *mutex* itself.

This function can also be used, if g_thread_init() has not yet been called and will do nothing then.

mutex : a GMutex.

g_mutex_free ()

```
void        g_mutex_free                    (GMutex *mutex);
```

Destroys *mutex*.

mutex : a GMutex.

GStaticMutex

```
typedef struct _GStaticMutex GStaticMutex;
```

A GStaticMutex works like a GMutex, but it has one significant advantage. It doesn't need to be created at run-time like a GMutex, but can be defined at compile-time. Here is a shorter, easier and safer version of our `give_me_next_number()` example:

Example 6. Using GStaticMutex to simplify thread-safe programming

```
int give_me_next_number ()
{
  static int current_number = 0;
  int ret_val;
  static GStaticMutex mutex = G_STATIC_MUTEX_INIT;

  g_static_mutex_lock (&mutex);
  ret_val = current_number = calc_next_number (current_number);
  g_static_mutex_unlock (&mutex);
  return ret_val;
}
```

Sometimes you would like to dynamically create a mutex. If you don't want to require prior calling to g_thread_init(), because your code should also be usable in non-threaded programs, you are not able to use g_mutex_new() and thus GMutex, as that requires a prior call to g_thread_init(). In theses cases you can also use a GStaticMutex. It must be initialized with g_static_mutex_init

() before using it and freed with with g_static_mutex_free() when not needed anymore to free up any allocated resources.

Even though GStaticMutex is not opaque, it should only be used with the following functions, as it is defined differently on different platforms.

All of the `g_static_mutex_*` functions can also be used, if g_thread_init() has not yet been called.

Note

All of the `g_static_mutex_*` functions are actually macros. Apart from taking their addresses, you can however use them as if they were functions.

G_STATIC_MUTEX_INIT

```
#define G_STATIC_MUTEX_INIT
```

A GStaticMutex must be initialized with this macro, before it can be used. This macro can used be to initialize a variable, but it cannot be assigned to a variable. In that case you have to use g_static_mutex_init().

```
GStaticMutex my_mutex = G_STATIC_MUTEX_INIT;
```

g_static_mutex_init ()

```
void        g_static_mutex_init             (GStaticMutex *mutex);
```

Initializes *mutex*. Alternatively you can initialize it with G_STATIC_MUTEX_INIT.

mutex : a GStaticMutex to be initialized.

g_static_mutex_lock ()

```
void        g_static_mutex_lock             (GStaticMutex *mutex);
```

Works like g_mutex_lock(), but for a GStaticMutex.

mutex : a GStaticMutex.

g_static_mutex_trylock ()

```
gboolean    g_static_mutex_trylock          (GStaticMutex *mutex);
```

Works like g_mutex_trylock(), but for a GStaticMutex.

mutex : a GStaticMutex.
Returns : TRUE, if the GStaticMutex could be locked.

---

g_static_mutex_unlock ()

```
void        g_static_mutex_unlock           (GStaticMutex *mutex);
```

Works like g_mutex_unlock(), but for a GStaticMutex.

mutex : a GStaticMutex.

---

g_static_mutex_get_mutex ()

```
GMutex*     g_static_mutex_get_mutex        (GStaticMutex *mutex);
```

For some operations (like g_cond_wait()) you must have a GMutex instead of a GStaticMutex. This function will return the corresponding GMutex for *mutex*.

mutex : a GStaticMutex.
Returns : the GMutex corresponding to *mutex*.

---

g_static_mutex_free ()

```
void        g_static_mutex_free             (GStaticMutex *mutex);
```

Releases all resources allocated to *mutex*.

You don't have to call this functions for a GStaticMutex with an unbounded lifetime, i.e. objects declared 'static', but if you have a GStaticMutex as a member of a structure and the structure is freed, you should also free the GStaticMutex.

mutex : a GStaticMutex to be freed.

G_LOCK_DEFINE()

```
#define     G_LOCK_DEFINE(name)
```

The G_LOCK_* macros provide a convenient interface to GStaticMutex with the advantage that they

will expand to nothing in programs compiled against a thread-disabled GLib, saving code and memory there. G_LOCK_DEFINE defines a lock. It can appear, where variable definitions may appear in programs, i.e. in the first block of a function or outside of functions. The *name* parameter will be mangled to get the name of the GStaticMutex. This means, that you can use names of existing variables as the parameter, e.g. the name of the variable you intent to protect with the lock. Look at our give_me_next_number() example using the G_LOCK_* macros:

Example 7. Using the G_LOCK_* convenience macros

```
G_LOCK_DEFINE (current_number);

int give_me_next_number ()
  {
    static int current_number = 0;
    int ret_val;

    G_LOCK (current_number);
    ret_val = current_number = calc_next_number (current_number);
    G_UNLOCK (current_number);
    return ret_val;
  }
```

name : the name of the lock.

G_LOCK_DEFINE_STATIC()

```
#define     G_LOCK_DEFINE_STATIC(name)
```

This works like G_LOCK_DEFINE, but it creates a static object.

name : the name of the lock.

G_LOCK_EXTERN()

```
#define     G_LOCK_EXTERN(name)
```

This declares a lock, that is defined with G_LOCK_DEFINE in another module.

name : the name of the lock.

G_LOCK()

```
#define     G_LOCK(name)
```

Works like g_mutex_lock(), but for a lock defined with G_LOCK_DEFINE.

name : the name of the lock.

G_TRYLOCK()

```
#define     G_TRYLOCK(name)
```

Works like `g_mutex_trylock()`, but for a lock defined with G_LOCK_DEFINE.

name : the name of the lock.
Returns : `TRUE`, if the lock could be locked.

G_UNLOCK()

```
#define     G_UNLOCK(name)
```

Works like `g_mutex_unlock()`, but for a lock defined with G_LOCK_DEFINE.

name : the name of the lock.

GStaticRecMutex

```
typedef struct {
} GStaticRecMutex;
```

A GStaticRecMutex works like a GStaticMutex, but it can be locked multiple times by one thread. If you enter it n times, however, you have to unlock it n times again to let other threads lock it. An exception is the function `g_static_rec_mutex_unlock_full()`, that allows you to unlock a GStaticRecMutex completely returning the depth, i.e. the number of times this mutex was locked. The depth can later be used to restore the state by calling `g_static_rec_mutex_lock_full()`.

Even though GStaticRecMutex is not opaque, it should only be used with the following functions.

All of the `g_static_rec_mutex_*` functions can also be used, if `g_thread_init()` has not been called.

G_STATIC_REC_MUTEX_INIT

```
#define G_STATIC_REC_MUTEX_INIT { G_STATIC_MUTEX_INIT }
```

A GStaticRecMutex must be initialized with this macro, before it can be used. This macro can used be to initialize a variable, but it cannot be assigned to a variable. In that case you have to use `g_static_rec_mutex_init()`.


```
GStaticRecMutex my_mutex = G_STATIC_REC_MUTEX_INIT;
```

---

g_static_rec_mutex_init ()

```
void        g_static_rec_mutex_init         (GStaticRecMutex *mutex);
```

A GStaticRecMutex must be initialized with this function, before it can be used. Alternatively you can initialize it with G_STATIC_REC_MUTEX_INIT.

mutex : a GStaticRecMutex to be initialized.

---

g_static_rec_mutex_lock ()

```
void        g_static_rec_mutex_lock         (GStaticRecMutex *mutex);
```

Locks *mutex*. If *mutex* is already locked by another thread, the current thread will block until *mutex* is unlocked by the other thread. If *mutex* is already locked by the calling thread, this functions increases the depth of *mutex* and returns immediately.

mutex : a GStaticRecMutex to lock.

---

g_static_rec_mutex_trylock ()

```
gboolean    g_static_rec_mutex_trylock      (GStaticRecMutex *mutex);
```

Tries to lock *mutex*. If *mutex* is already locked by another thread, it immediately returns `FALSE`. Otherwise it locks *mutex* and returns `TRUE`. If *mutex* is already locked by the calling thread, this functions increases the depth of *mutex* and immediately returns `TRUE`.

mutex : a GStaticRecMutex to lock.
Returns : `TRUE`, if *mutex* could be locked.

---

g_static_rec_mutex_unlock ()

```
void        g_static_rec_mutex_unlock       (GStaticRecMutex *mutex);
```

Unlocks *mutex*. Another threads can, however, only lock *mutex* when it has been unlocked as many times, as it had been locked before. If *mutex* is completely unlocked and another thread is blocked in a `g_static_rec_mutex_lock()` call for *mutex*, it will be woken and can lock *mutex* itself.

mutex : a GStaticRecMutex to unlock.

g_static_rec_mutex_lock_full ()

```
void        g_static_rec_mutex_lock_full    (GStaticRecMutex *mutex,
                                             guint depth);
```

Works like calling g_static_rec_mutex_lock() for *mutex* *depth* times.

mutex : a GStaticRecMutex to lock.
depth : number of times this mutex has to be unlocked to be completely unlocked.

g_static_rec_mutex_unlock_full ()

```
guint       g_static_rec_mutex_unlock_full  (GStaticRecMutex *mutex);
```

Completely unlocks *mutex*. If another thread is blocked in a g_static_rec_mutex_lock() call for *mutex*, it will be woken and can lock *mutex* itself. This function returns the number of times, that *mutex* has been locked by the current thread. To restore the state before the call to g_static_rec_mutex_unlock_full() you can call g_static_rec_mutex_lock_full() with the depth returned by this function.

mutex : a GStaticRecMutex to completely unlock.
Returns : number of times *mutex* has been locked by the current thread.

g_static_rec_mutex_free ()

```
void        g_static_rec_mutex_free         (GStaticRecMutex *mutex);
```

Releases all resources allocated to a GStaticRecMutex.

You don't have to call this functions for a GStaticRecMutex with an unbounded lifetime, i.e. objects declared 'static', but if you have a GStaticRecMutex as a member of a structure and the structure is freed, you should also free the GStaticRecMutex.

mutex : a GStaticRecMutex to be freed.

GStaticRWLock

```
typedef struct {
} GStaticRWLock;
```

The GStaticRWLock struct represents a read-write lock. A read-write lock can be used for protecting data, that some portions of code only read from, while others also write. In such situations it is desirable, that several readers can read at once, whereas of course only one writer may write at a

time. Take a look at the following example:

Example 8. An array with access functions

```
GStaticRWLock rwlock = G_STATIC_RW_LOCK_INIT;

GPtrArray *array;

gpointer my_array_get (guint index)
{
  gpointer retval = NULL;

  if (!array)
    return NULL;

  g_static_rw_lock_reader_lock (&rwlock);

  if (index < array->len)
    retval = g_ptr_array_index (array, index);

  g_static_rw_lock_reader_unlock (&rwlock);

  return retval;
}

void my_array_set (guint index, gpointer data)
{
  g_static_rw_lock_writer_lock (&rwlock);

  if (!array)
    array = g_ptr_array_new ();

  if (index >= array->len)
    g_ptr_array_set_size (array, index+1);

  g_ptr_array_index (array, index) = data;

  g_static_rw_lock_writer_unlock (&rwlock);
}
```

This example shows an array, which can be accessed by many readers (the `my_array_get()` function) simultaneously, whereas the writers (the `my_array_set()` function) will only be allowed once a time and only if no readers currently access the array. This is because of the potentially dangerous resizing of the array. Using these functions is fully multi-thread safe now.

Most of the time the writers should have precedence of readers. That means for this implementation, that as soon as a writer wants to lock the data, no other reader is allowed to lock the data, whereas of course the readers, that already have locked the data are allowed to finish their operation. As soon as the last reader unlocks the data, the writer will lock it.

Even though GStaticRWLock is not opaque, it should only be used with the following functions.

All of the `g_static_rw_lock_*` functions can also be used, if g_thread_init() has not been called.

Note

A read-write lock has a higher overhead as a mutex. For example both

g_static_rw_lock_reader_lock() and g_static_rw_lock_reader_unlock() have to lock and unlock a GStaticMutex, so it takes at least twice the time to lock and unlock a GStaticRWLock than to lock and unlock a GStaticMutex. So only data structures, that are accessed by multiple readers, which keep the lock for a considerable time justify a GStaticRWLock. The above example most probably would fare better with a GStaticMutex.

G_STATIC_RW_LOCK_INIT

```
#define G_STATIC_RW_LOCK_INIT { G_STATIC_MUTEX_INIT, NULL, NULL, 0, FALSE, 0, 0
```

A GStaticRWLock must be initialized with this macro, before it can be used. This macro can used be to initialize a variable, but it cannot be assigned to a variable. In that case you have to use g_static_rw_lock_init().

```
GStaticRWLock my_lock = G_STATIC_RW_LOCK_INIT;
```

---

g_static_rw_lock_init ()

```
void        g_static_rw_lock_init           (GStaticRWLock *lock);
```

A GStaticRWLock must be initialized with this function, before it can be used. Alternatively you can initialize it with G_STATIC_RW_LOCK_INIT.

lock : a GStaticRWLock to be initialized.

---

g_static_rw_lock_reader_lock ()

```
void        g_static_rw_lock_reader_lock    (GStaticRWLock *lock);
```

Locks *lock* for reading. There may be unlimited concurrent locks for reading of a GStaticRWLock at the same time. If *lock* is already locked for writing by another thread or if another thread is already waiting to lock *lock* for writing, this function will block until *lock* is unlocked by the other writing thread and no other writing threads want to lock *lock*. This lock has to be unlocked by g_static_rw_lock_reader_unlock().

GStaticRWLock is not recursive. It might seem to be possible to recursively lock for reading, but that can result in a deadlock as well, due to writer preference.

lock : a GStaticRWLock to lock for reading.

---

g_static_rw_lock_reader_trylock ()


```
gboolean    g_static_rw_lock_reader_trylock (GStaticRWLock *lock);
```

Tries to lock *lock* for reading. If *lock* is already locked for writing by another thread or if another thread is already waiting to lock *lock* for writing, it immediately returns FALSE. Otherwise it locks *lock* for reading and returns TRUE. This lock has to be unlocked by g_static_rw_lock_reader_unlock().

lock : a GStaticRWLock to lock for reading.

Returns : TRUE, if *lock* could be locked for reading.

---

g_static_rw_lock_reader_unlock ()

```
void        g_static_rw_lock_reader_unlock  (GStaticRWLock *lock);
```

Unlocks *lock*. If a thread waits to lock *lock* for writing and all locks for reading have been unlocked, the waiting thread is woken up and can lock *lock* for writing.

lock : a GStaticRWLock to unlock after reading.

---

g_static_rw_lock_writer_lock ()

```
void        g_static_rw_lock_writer_lock    (GStaticRWLock *lock);
```

Locks *lock* for writing. If *lock* is already locked for writing or reading by other threads, this function will block until *lock* is completely unlocked and then lock *lock* for writing. While this functions waits to lock *lock*, no other thread can lock *lock* for reading. When *lock* is locked for writing, no other thread can lock *lock* (neither for reading nor writing). This lock has to be unlocked by g_static_rw_lock_writer_unlock().

lock : a GStaticRWLock to lock for writing.

---

g_static_rw_lock_writer_trylock ()

```
gboolean    g_static_rw_lock_writer_trylock (GStaticRWLock *lock);
```

Tries to lock *lock* for writing. If *lock* is already locked (for either reading or writing) by another thread, it immediately returns FALSE. Otherwise it locks *lock* for writing and returns TRUE. This lock has to be unlocked by g_static_rw_lock_writer_unlock().

lock : a GStaticRWLock to lock for writing.

Returns : TRUE, if *lock* could be locked for writing.

---

g_static_rw_lock_writer_unlock ()

```
void        g_static_rw_lock_writer_unlock  (GStaticRWLock *lock);
```

Unlocks *lock*. If a thread waits to lock *lock* for writing and all locks for reading have been unlocked, the waiting thread is woken up and can lock *lock* for writing. If no thread waits to lock *lock* for writing and threads wait to lock *lock* for reading, the waiting threads are woken up and can lock *lock* for reading.

lock : a GStaticRWLock to unlock after writing.

---

g_static_rw_lock_free ()

```
void        g_static_rw_lock_free           (GStaticRWLock *lock);
```

Releases all resources allocated to *lock*.

You don't have to call this functions for a GStaticRWLock with an unbounded lifetime, i.e. objects declared 'static', but if you have a GStaticRWLock as a member of a structure and the structure is freed, you should also free the GStaticRWLock.

lock : a GStaticRWLock to be freed.

GCond

```
typedef struct _GCond GCond;
```

The GCond struct is an opaque data structure to represent a condition. A GCond is an object, that threads can block on, if they find a certain condition to be false. If other threads change the state of this condition they can signal the GCond, such that the waiting thread is woken up.

Example 9. Using GCond to block a thread until a condition is satisfied

```
GCond* data_cond = NULL;   /* Must be initialized somewhere */
GMutex* data_mutex = NULL; /* Must be initialized somewhere */
gpointer current_data = NULL;

void push_data (gpointer data)
{
  g_mutex_lock (data_mutex);
  current_data = data;
  g_cond_signal (data_cond);
  g_mutex_unlock (data_mutex);
}

gpointer pop_data ()
{
  gpointer data;

  g_mutex_lock (data_mutex);
```



```
  while (!current_data)
    g_cond_wait (data_cond, data_mutex);
  data = current_data;
  current_data = NULL;
  g_mutex_unlock (data_mutex);
  return data;
}
```

Whenever a thread calls `pop_data()` now, it will wait until current_data is non-`NULL`, i.e. until some other thread has called `push_data()`.

Note

It is important to use the `g_cond_wait()` and `g_cond_timed_wait()` functions only inside a loop, which checks for the condition to be true as it is not guaranteed that the waiting thread will find it fulfilled, even if the signaling thread left the condition in that state. This is because another thread can have altered the condition, before the waiting thread got the chance to be woken up, even if the condition itself is protected by a GMutex, like above.

A GCond should only be accessed via the following functions.

Note

All of the `g_cond_*` functions are actually macros. Apart from taking their addresses, you can however use them as if they were functions.

---

g_cond_new ()

```
GCond*      g_cond_new                      ();
```

Creates a new GCond. This function will abort, if `g_thread_init()` has not been called yet.

Returns : a new GCond.

---

g_cond_signal ()

```
void        g_cond_signal                   (GCond *cond);
```

If threads are waiting for *cond*, exactly one of them is woken up. It is good practice to hold the same lock as the waiting thread, while calling this function, though not required.

This function can also be used, if `g_thread_init()` has not yet been called and will do nothing then.

cond : a GCond.

---

g_cond_broadcast ()

```
void        g_cond_broadcast                (GCond *cond);
```

If threads are waiting for *cond*, all of them are woken up. It is good practice to lock the same mutex as the waiting threads, while calling this function, though not required.

This function can also be used, if g_thread_init() has not yet been called and will do nothing then.

cond : a GCond.

---

g_cond_wait ()

```
void        g_cond_wait                     (GCond *cond,
                                             GMutex *mutex);
```

Waits until this thread is woken up on *cond*. The *mutex* is unlocked before falling asleep and locked again before resuming.

This function can also be used, if g_thread_init() has not yet been called and will immediately return then.

cond : a GCond.
mutex : a GMutex, that is currently locked.

---

g_cond_timed_wait ()

```
gboolean    g_cond_timed_wait               (GCond *cond,
                                             GMutex *mutex,
                                             GTimeVal *abs_time);
```

Waits until this thread is woken up on *cond*, but not longer than until the time, that is specified by *abs_time*. The *mutex* is unlocked before falling asleep and locked again before resuming.

If *abs_time* is NULL, g_cond_timed_wait() acts like g_cond_wait().

This function can also be used, if g_thread_init() has not yet been called and will immediately return TRUE then.

To easily calculate *abs_time* a combination of g_get_current_time() and g_time_val_add() can be used.

cond : a GCond.
mutex : a GMutex, that is currently locked.
abs_time : a GTimeVal, determining the final time.

Returns : TRUE, if the thread is woken up in time.

---

g_cond_free ()

```
void        g_cond_free                     (GCond *cond);
```

Destroys the GCond.

cond : a GCond.

GPrivate

```
typedef struct _GPrivate GPrivate;
```

The GPrivate struct is an opaque data structure to represent a thread private data key. Threads can thereby obtain and set a pointer, which is private to the current thread. Take our give_me_next_number() example from above. Now we don't want current_number to be shared between the threads, but to be private to each thread. This can be done as follows:

Example 10. Using GPrivate for per-thread data

```
  GPrivate* current_number_key = NULL; /* Must be initialized somewhere */
                                       /* with g_private_new (g_free); */

  int give_me_next_number ()
  {
    int *current_number = g_private_get (current_number_key);

    if (!current_number)
    {
      current_number = g_new (int,1);
      *current_number = 0;
      g_private_set (current_number_key, current_number);
    }
    *current_number = calc_next_number (*current_number);
    return *current_number;
  }
```

Here the pointer belonging to the key current_number_key is read. If it is NULL, it has not been set yet. Then get memory for an integer value, assign this memory to the pointer and write the pointer back. Now we have an integer value, that is private to the current thread.

The GPrivate struct should only be accessed via the following functions.

> **Note**
>
> All of the g_private_* functions are actually macros. Apart from taking their addresses, you can however use them as if they were functions.

---

g_private_new ()

```
GPrivate*   g_private_new                   (GDestroyNotify destructor);
```

Creates a new GPrivate. If *destructor* is non-NULL, it is a pointer to a destructor function. Whenever a thread ends and the corresponding pointer keyed to this instance of GPrivate is non-NULL, the destructor is called with this pointer as the argument.

Note

destructor is working quite differently from *notify* in g_static_private_set().

Note

A GPrivate can not be freed. Reuse it instead, if you can to avoid shortage or use GStaticPrivate.

Note

This function will abort, if g_thread_init() has not been called yet.

destructor : a function to handle the data keyed to GPrivate, when a thread ends.
Returns : a new GPrivate.

---

g_private_get ()

```
gpointer    g_private_get                   (GPrivate *private_key);
```

Returns the pointer keyed to *private_key* for the current thread. This pointer is NULL, when g_private_set() hasn't been called for the current *private_key* and thread yet.

This function can also be used, if g_thread_init() has not yet been called and will return the value of *private_key* casted to gpointer then.

private_key : a GPrivate.
Returns : the corresponding pointer.

---

g_private_set ()

```
void        g_private_set                   (GPrivate *private_key,
                                             gpointer data);
```

Sets the pointer keyed to *private_key* for the current thread.

This function can also be used, if g_thread_init() has not yet been called and will set *private_key* to *data* casted to GPrivate* then.

private_key : a GPrivate.
data : the new pointer.

GStaticPrivate

```
typedef struct {
} GStaticPrivate;
```

A GStaticPrivate works almost like a GPrivate, but it has one significant advantage. It doesn't need to be created at run-time like a GPrivate, but can be defined at compile-time. This is similar to the difference between GMutex and GStaticMutex. Now look at our give_me_next_number() example with GStaticPrivate:

Example 11. Using GStaticPrivate for per-thread data

```
int give_me_next_number ()
{
  static GStaticPrivate current_number_key = G_STATIC_PRIVATE_INIT;
  int *current_number = g_static_private_get (&current_number_key);

  if (!current_number)
  {
    current_number = g_new (int,1);
    *current_number = 0;
    g_static_private_set (&current_number_key, current_number, g_free);
  }
  *current_number = calc_next_number (*current_number);
  return *current_number;
}
```

G_STATIC_PRIVATE_INIT

```
#define G_STATIC_PRIVATE_INIT
```

Every GStaticPrivate must be initialized with this macro, before it can be used.

```
GStaticPrivate my_private = G_STATIC_PRIVATE_INIT;
```

---

g_static_private_init ()

```
void        g_static_private_init           (GStaticPrivate *private_key);
```

Initializes *private_key*. Alternatively you can initialize it with G_STATIC_PRIVATE_INIT.

private_key : a GStaticPrivate to be initialized.

g_static_private_get ()

```
gpointer    g_static_private_get            (GStaticPrivate *private_key);
```

Works like g_private_get() only for a GStaticPrivate.

This function also works, if g_thread_init() has not yet been called.

private_key : a GStaticPrivate.
Returns : the corresponding pointer.

g_static_private_set ()

```
void        g_static_private_set            (GStaticPrivate *private_key,
                                             gpointer data,
                                             GDestroyNotify notify);
```

Sets the pointer keyed to *private_key* for the current thread and the function *notify* to be called with that pointer (NULL or non-NULL), whenever the pointer is set again or whenever the current thread ends.

This function also works, if g_thread_init() has not yet been called. If g_thread_init() is called later, the *data* keyed to *private_key* will be inherited only by the main thread, i.e. the one that called g_thread_init().

Note

notify is working quite differently from *destructor* in g_private_new().

private_key : a GStaticPrivate.
data : the new pointer.
notify : a function to be called with the pointer, whenever the current thread ends or sets this pointer again.

g_static_private_free ()

```
void        g_static_private_free           (GStaticPrivate *private_key);
```

Releases all resources allocated to *private_key*.

You don't have to call this functions for a GStaticPrivate with an unbounded lifetime, i.e. objects declared 'static', but if you have a GStaticPrivate as a member of a structure and the structure is freed, you should also free the GStaticPrivate.

private_key : a GStaticPrivate to be freed.

GOnce

```
typedef struct {
  volatile GOnceStatus status;
  volatile gpointer retval;
} GOnce;
```

A GOnce struct controls a one-time initialization function. Any one-time initialization function must have its own unique GOnce struct.

Since 2.4

enum GOnceStatus

```
typedef enum
{
  G_ONCE_STATUS_NOTCALLED,
  G_ONCE_STATUS_PROGRESS,
  G_ONCE_STATUS_READY
} GOnceStatus;
```

The possible stati of a one-time initialization function controlled by a GOnce struct.

G_ONCE_STATUS_NOTCALLED the function has not been called yet.
G_ONCE_STATUS_PROGRESS the function call is currently in progress.
G_ONCE_STATUS_READY the function has been called.

Since 2.4

G_ONCE_INIT

```
#define G_ONCE_INIT { G_ONCE_STATUS_NOTCALLED, NULL }
```

A GOnce must be initialized with this macro, before it can be used.

```
GOnce my_once = G_ONCE_INIT;
```

Since 2.4

g_once()

```
#define     g_once(once, func, arg)
```

The first call to this routine by a process with a given GOnce struct calls *func* with the given argument. Thereafter, subsequent calls to g_once() with the same GOnce struct do not call *func* again, but return the stored result of the first call. On return from g_once(), the status of *once* will be G_ONCE_STATUS_READY.

For example, a mutex or a thread-specific data key must be created exactly once. In a threaded environment, calling g_once() ensures that the initialization is serialized across multiple threads.

Note

Calling g_once() recursively on the same GOnce struct in *func* will lead to a deadlock.

```
gpointer
get_debug_flags()
{
  static GOnce my_once = G_ONCE_INIT;

  g_once (&my_once, parse_debug_flags, NULL);

  return my_once.retval;
}
```

once : a GOnce structure

func : the function associated to *once*. This function is called only once, regardless of the number of times it and its associated GOnce struct are passed to g_once() .

arg : data to be passed to *func*

Since 2.4

See Also

GThreadPool Thread pools.

GAsyncQueue Send asynchronous messages between threads.

<< The Main Event Loop **Thread Pools >>**

Thread Pools

Thread Pools — pools of threads to execute work concurrently.

Synopsis

```
#include <glib.h>


            GThreadPool;
GThreadPool* g_thread_pool_new              (GFunc func,
                                             gpointer user_data,
                                             gint max_threads,
                                             gboolean exclusive,
                                             GError **error);
void        g_thread_pool_push              (GThreadPool *pool,
                                             gpointer data,
                                             GError **error);
void        g_thread_pool_set_max_threads   (GThreadPool *pool,
                                             gint max_threads,
                                             GError **error);
gint        g_thread_pool_get_max_threads   (GThreadPool *pool);
guint       g_thread_pool_get_num_threads   (GThreadPool *pool);
guint       g_thread_pool_unprocessed       (GThreadPool *pool);
void        g_thread_pool_free              (GThreadPool *pool,
                                             gboolean immediate,
                                             gboolean wait);
void        g_thread_pool_set_max_unused_threads
                                            (gint max_threads);
gint        g_thread_pool_get_max_unused_threads
                                            (void);
guint       g_thread_pool_get_num_unused_threads
                                            (void);
void        g_thread_pool_stop_unused_threads
                                            (void);
```

Description

Sometimes you wish to asyncronously fork out the execution of work and continue working in your own thread. If that will happen often, the overhead of starting and destroying a thread each time might be to high. In such cases reusing already started threads seems like a good idea. And it indeed is, but implementing this can be tedious and error-prone.

Therefore GLib provides thread pools for your convenience. An added advantage is, that the threads can be shared between the different subsystems of your program, when they are using GLib.

To create a new thread pool, you use `g_thread_pool_new()`. It is destroyed by `g_thread_pool_free()`.

If you want to execute a certain task within a thread pool, you call `g_thread_pool_push()`.

To get the current number of running threads you call `g_thread_pool_get_num_threads()`. To get the number of still unprocessed tasks you call `g_thread_pool_unprocessed()`. To control the maximal number of threads for a thread pool, you use `g_thread_pool_get_max_threads()` and `g_thread_pool_set_max_threads()`.

Finally you can control the number of unused threads, that are kept alive by GLib for future use. The current number can be fetched with `g_thread_pool_get_num_unused_threads()`. The maximal number can be controlled by `g_thread_pool_get_max_unused_threads()` and `g_thread_pool_set_max_unused_threads()`. All currently unused threads can be stopped by calling `g_thread_pool_stop_unused_threads()`.

Details

GThreadPool

```
typedef struct {
  GFunc func;
  gpointer user_data;
  gboolean exclusive;
} GThreadPool;
```

The GThreadPool struct represents a thread pool. It has six public read-only members, but the underlying struct is bigger, so you must not copy this struct.

GFunc *func*;	the function to execute in the threads of this pool
gpointer *user_data*;	the user data for the threads of this pool
gboolean *exclusive*;	are all threads exclusive to this pool

---

g_thread_pool_new ()

```
GThreadPool* g_thread_pool_new              (GFunc func,
                                             gpointer user_data,
                                             gint max_threads,
                                             gboolean exclusive,
                                             GError **error);
```

This function creates a new thread pool.

Whenever you call `g_thread_pool_push()`, either a new thread is created or an unused one is reused. At most *max_threads* threads are running concurrently for this thread pool. *max_threads* = -1 allows unlimited threads to be created for this thread pool. The newly created or reused thread now executes the function *func* with the two arguments. The first one is the parameter to `g_thread_pool_push()` and the second one is *user_data*.

The parameter *exclusive* determines, whether the thread pool owns all threads exclusive or whether the threads are shared globally. If *exclusive* is TRUE, *max_threads* threads are started immediately and they will run exclusively for this thread pool until it is destroyed by `g_thread_pool_free()`. If *exclusive* is FALSE, threads are created, when needed and shared between all non-exclusive thread pools. This implies that *max_threads* may not be -1 for exclusive thread pools.

error can be NULL to ignore errors, or non-NULL to report errors. An error can only occur when *exclusive* is set to TRUE and not all *max_threads* threads could be created.

func : a function to execute in the threads of the new thread pool
user_data : user data that is handed over to *func* every time it is called
max_threads : the maximal number of threads to execute concurrently in the new thread pool, -1 means no limit
exclusive : should this thread pool be exclusive?
error : return location for error
Returns : the new GThreadPool

---

g_thread_pool_push ()

```
void        g_thread_pool_push              (GThreadPool *pool,
                                             gpointer data,
                                             GError **error);
```

Inserts *data* into the list of tasks to be executed by *pool*. When the number of currently running threads is lower than the maximal allowed number of threads, a new thread is started (or reused) with the properties given to g_thread_pool_new(). Otherwise *data* stays in the queue until a thread in this pool finishes its previous task and processes *data*.

error can be NULL to ignore errors, or non-NULL to report errors. An error can only occur when a new thread couldn't be created. In that case *data* is simply appended to the queue of work to do.

pool : a GThreadPool
data : a new task for *pool*
error : return location for error

---

g_thread_pool_set_max_threads ()

```
void        g_thread_pool_set_max_threads   (GThreadPool *pool,
                                             gint max_threads,
                                             GError **error);
```

Sets the maximal allowed number of threads for *pool*. A value of -1 means, that the maximal number of threads is unlimited.

Setting *max_threads* to 0 means stopping all work for *pool*. It is effectively frozen until *max_threads* is set to a non-zero value again.

A thread is never terminated while calling *func*, as supplied by g_thread_pool_new(). Instead the maximal number of threads only has effect for the allocation of new threads in g_thread_pool_push(). A new thread is allocated, whenever the number of currently running threads in *pool* is smaller than the maximal number.

error can be NULL to ignore errors, or non-NULL to report errors. An error can only occur when a

new thread couldn't be created.

pool : a GThreadPool
max_threads : a new maximal number of threads for *pool*
error : return location for error

---

g_thread_pool_get_max_threads ()

```
gint        g_thread_pool_get_max_threads   (GThreadPool *pool);
```

Returns the maximal number of threads for *pool*.

pool : a GThreadPool
Returns : the maximal number of threads

---

g_thread_pool_get_num_threads ()

```
guint       g_thread_pool_get_num_threads   (GThreadPool *pool);
```

Returns the number of threads currently running in *pool*.

pool : a GThreadPool
Returns : the number of threads currently running

---

g_thread_pool_unprocessed ()

```
guint       g_thread_pool_unprocessed       (GThreadPool *pool);
```

Returns the number of tasks still unprocessed in *pool*.

pool : a GThreadPool
Returns : the number of unprocessed tasks

---

g_thread_pool_free ()

```
void        g_thread_pool_free              (GThreadPool *pool,
                                             gboolean immediate,
                                             gboolean wait);
```

Frees all resources allocated for *pool*.

If *immediate* is TRUE, no new task is processed for *pool*. Otherwise *pool* is not freed before the last

task is processed. Note however, that no thread of this pool is interrupted, while processing a task. Instead at least all still running threads can finish their tasks before the *pool* is freed.

If *wait* is TRUE, the functions does not return before all tasks to be processed (dependent on *immediate*, whether all or only the currently running) are ready. Otherwise the function returns immediately.

After calling this function *pool* must not be used anymore.

pool : a GThreadPool
immediate : should *pool* shut down immediately?
wait : should the function wait for all tasks to be finished?

---

g_thread_pool_set_max_unused_threads ()

```
void        g_thread_pool_set_max_unused_threads
                                        (gint max_threads);
```

Sets the maximal number of unused threads to *max_threads*. If *max_threads* is -1, no limit is imposed on the number of unused threads.

max_threads : maximal number of unused threads

---

g_thread_pool_get_max_unused_threads ()

```
gint        g_thread_pool_get_max_unused_threads
                                        (void);
```

Returns the maximal allowed number of unused threads.

Returns : the maximal number of unused threads

---

g_thread_pool_get_num_unused_threads ()

```
guint       g_thread_pool_get_num_unused_threads
                                        (void);
```

Returns the number of currently unused threads.

Returns : the number of currently unused threads

---

g_thread_pool_stop_unused_threads ()


```
void        g_thread_pool_stop_unused_threads
                                        (void);
```

Stops all currently unused threads. This does not change the maximal number of unused threads. This function can be used to regularly stop all unused threads e.g. from `g_timeout_add()`.

See Also

GThread GLib thread system.

<< Threads **Asynchronous Queues >>**

Asynchronous Queues

Asynchronous Queues — asynchronous communication between threads.

Synopsis

```
#include <glib.h>


            GAsyncQueue;
GAsyncQueue* g_async_queue_new              (void);
GAsyncQueue* g_async_queue_ref              (GAsyncQueue *queue);
void        g_async_queue_unref             (GAsyncQueue *queue);
void        g_async_queue_push              (GAsyncQueue *queue,
                                             gpointer data);
gpointer    g_async_queue_pop               (GAsyncQueue *queue);
gpointer    g_async_queue_try_pop           (GAsyncQueue *queue);
gpointer    g_async_queue_timed_pop         (GAsyncQueue *queue,
                                             GTimeVal *end_time);
gint        g_async_queue_length            (GAsyncQueue *queue);

void        g_async_queue_lock              (GAsyncQueue *queue);
void        g_async_queue_unlock            (GAsyncQueue *queue);
void        g_async_queue_ref_unlocked      (GAsyncQueue *queue);
void        g_async_queue_unref_and_unlock  (GAsyncQueue *queue);
void        g_async_queue_push_unlocked     (GAsyncQueue *queue,
                                             gpointer data);
gpointer    g_async_queue_pop_unlocked      (GAsyncQueue *queue);
gpointer    g_async_queue_try_pop_unlocked  (GAsyncQueue *queue);
gpointer    g_async_queue_timed_pop_unlocked
                                            (GAsyncQueue *queue,
                                             GTimeVal *end_time);
gint        g_async_queue_length_unlocked   (GAsyncQueue *queue);
```

Description

Often you need to communicate between different threads. In general it's safer not to do this by shared memory, but by explicit message passing. These messages only make sense asynchronously for multi-threaded applications though, as a synchronous operation could as well be done in the same thread.

Asynchronous queues are an exception from most other GLib data structures, as they can be used simultaneously from multiple threads without explicit locking and they bring their own builtin reference counting. This is because the nature of an asynchronous queue is that it will always be used by at least 2 concurrent threads.

For using an asynchronous queue you first have to create one with `g_async_queue_new()`. A newly-created queue will get the reference count 1. Whenever another thread is creating a new reference of (that is, pointer to) the queue, it has to increase the reference count (using `g_async_queue_ref()`). Also, before removing this reference, the reference count has to be decreased (using

`g_async_queue_unref()`). After that the queue might no longer exist so you must not access it after that point.

A thread, which wants to send a message to that queue simply calls `g_async_queue_push()` to push the message to the queue.

A thread, which is expecting messages from an asynchronous queue simply calls `g_async_queue_pop()` for that queue. If no message is available in the queue at that point, the thread is now put to sleep until a message arrives. The message will be removed from the queue and returned. The functions `g_async_queue_try_pop()` and `g_async_queue_timed_pop()` can be used to only check for the presence of messages or to only wait a certain time for messages respectively.

For almost every function there exist two variants, one that locks the queue and one that doesn't. That way you can hold the queue lock (acquire it with `g_async_queue_lock()` and release it with `g_async_queue_unlock()`) over multiple queue accessing instructions. This can be necessary to ensure the integrity of the queue, but should only be used when really necessary, as it can make your life harder if used unwisely. Normally you should only use the locking function variants (those without the suffix _unlocked)

Details

GAsyncQueue

```
typedef struct _GAsyncQueue GAsyncQueue;
```

The GAsyncQueue struct is an opaque data structure, which represents an asynchronous queue. It should only be accessed through the `g_async_queue_*` functions.

---

g_async_queue_new ()

```
GAsyncQueue* g_async_queue_new              (void);
```

Creates a new asynchronous queue with the initial reference count of 1.

Returns : the new GAsyncQueue.

---

g_async_queue_ref ()

```
GAsyncQueue* g_async_queue_ref              (GAsyncQueue *queue);
```

Increases the reference count of the asynchronous *queue* by 1. You do not need to hold the lock to call this function.

queue : a GAsyncQueue.
Returns : the *queue* that was passed in (since 2.6)

---

g_async_queue_unref ()

```
void        g_async_queue_unref             (GAsyncQueue *queue);
```

Decreases the reference count of the asynchronous *queue* by 1. If the reference count went to 0, the *queue* will be destroyed and the memory allocated will be freed. So you are not allowed to use the *queue* afterwards, as it might have disappeared. You do not need to hold the lock to call this function.

queue : a GAsyncQueue.

---

g_async_queue_push ()

```
void        g_async_queue_push              (GAsyncQueue *queue,
                                             gpointer data);
```

Pushes the *data* into the *queue*. *data* must not be NULL.

queue : a GAsyncQueue.
data : *data* to push into the *queue*.

---

g_async_queue_pop ()

```
gpointer    g_async_queue_pop               (GAsyncQueue *queue);
```

Pops data from the *queue*. This function blocks until data become available.

queue : a GAsyncQueue.
Returns : data from the queue.

---

g_async_queue_try_pop ()

```
gpointer    g_async_queue_try_pop           (GAsyncQueue *queue);
```

Tries to pop data from the *queue*. If no data is available, NULL is returned.

queue : a GAsyncQueue.
Returns : data from the queue or NULL, when no data is available immediately.

---

g_async_queue_timed_pop ()

```
gpointer    g_async_queue_timed_pop         (GAsyncQueue *queue,
```



```
                                             GTimeVal *end_time);
```

Pops data from the *queue*. If no data is received before *end_time*, NULL is returned.

To easily calculate *end_time* a combination of g_get_current_time() and g_time_val_add() can be used.

queue : a GAsyncQueue.
end_time : a GTimeVal, determining the final time.
Returns : data from the queue or NULL, when no data is received before *end_time*.

---

g_async_queue_length ()

```
gint        g_async_queue_length            (GAsyncQueue *queue);
```

Returns the length of the queue, negative values mean waiting threads, positive values mean available entries in the *queue*. Actually this function returns the number of data items in the queue minus the number of waiting threads. Thus a return value of 0 could mean 'n' entries in the queue and 'n' thread waiting. That can happen due to locking of the queue or due to scheduling.

queue : a GAsyncQueue.
Returns : the length of the *queue*.

---

g_async_queue_lock ()

```
void        g_async_queue_lock              (GAsyncQueue *queue);
```

Acquires the *queue*'s lock. After that you can only call the g_async_queue_*_unlocked() function variants on that *queue*. Otherwise it will deadlock.

queue : a GAsyncQueue.

---

g_async_queue_unlock ()

```
void        g_async_queue_unlock            (GAsyncQueue *queue);
```

Releases the queue's lock.

queue : a GAsyncQueue.

---

g_async_queue_ref_unlocked ()

```
void        g_async_queue_ref_unlocked      (GAsyncQueue *queue);
```

Warning

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

Increases the reference count of the asynchronous *queue* by 1.

queue : a GAsyncQueue.

---

g_async_queue_unref_and_unlock ()

```
void        g_async_queue_unref_and_unlock  (GAsyncQueue *queue);
```

Warning

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

Decreases the reference count of the asynchronous *queue* by 1 and releases the lock. This function must be called while holding the *queue*'s lock. If the reference count went to 0, the *queue* will be destroyed and the memory allocated will be freed.

queue : a GAsyncQueue.

---

g_async_queue_push_unlocked ()

```
void        g_async_queue_push_unlocked     (GAsyncQueue *queue,
                                             gpointer data);
```

Pushes the *data* into the *queue*. *data* must not be `NULL`. This function must be called while holding the *queue*'s lock.

queue : a GAsyncQueue.
data : *data* to push into the *queue*.

---

g_async_queue_pop_unlocked ()

```
gpointer    g_async_queue_pop_unlocked      (GAsyncQueue *queue);
```

Pops data from the *queue*. This function blocks until data become available. This function must be called while holding the *queue*'s lock.

queue : a GAsyncQueue.
Returns : data from the queue.

---

g_async_queue_try_pop_unlocked ()

```
gpointer    g_async_queue_try_pop_unlocked  (GAsyncQueue *queue);
```

Tries to pop data from the *queue*. If no data is available, `NULL` is returned. This function must be called while holding the *queue*'s lock.

queue : a GAsyncQueue.
Returns : data from the queue or `NULL`, when no data is available immediately.

---

g_async_queue_timed_pop_unlocked ()

```
gpointer    g_async_queue_timed_pop_unlocked
                                            (GAsyncQueue *queue,
                                             GTimeVal *end_time);
```

Pops data from the *queue*. If no data is received before *end_time*, `NULL` is returned. This function must be called while holding the *queue*'s lock.

To easily calculate *end_time* a combination of `g_get_current_time()` and `g_time_val_add()` can be used.

queue : a GAsyncQueue.
end_time : a GTimeVal, determining the final time.
Returns : data from the queue or `NULL`, when no data is received before *end_time*.

---

g_async_queue_length_unlocked ()

```
gint        g_async_queue_length_unlocked   (GAsyncQueue *queue);
```

Returns the length of the queue, negative values mean waiting threads, positive values mean available entries in the *queue*. Actually this function returns the number of data items in the queue minus the number of waiting threads. Thus a return value of 0 could mean 'n' entries in the queue and 'n' thread waiting. That can happen due to locking of the queue or due to scheduling. This function must be called while holding the *queue*'s lock.

queue : a GAsyncQueue.
Returns : the length of the *queue*.

<< Thread Pools **Dynamic Loading of Modules >>**

Dynamic Loading of Modules

Dynamic Loading of Modules — portable method for dynamically loading 'plug-ins'.

Synopsis

```
#include <gmodule.h>


            GModule;
gboolean    g_module_supported              (void);
gchar*      g_module_build_path             (const gchar *directory,
                                             const gchar *module_name);
GModule*    g_module_open                   (const gchar *file_name,
                                             GModuleFlags flags);

enum        GModuleFlags;
gboolean    g_module_symbol                 (GModule *module,
                                             const gchar *symbol_name,
                                             gpointer *symbol);
G_CONST_RETURN gchar* g_module_name          (GModule *module);
void        g_module_make_resident          (GModule *module);
gboolean    g_module_close                  (GModule *module);
G_CONST_RETURN gchar* g_module_error         (void);

const gchar* (*GModuleCheckInit)            (GModule *module);
void        (*GModuleUnload)                (GModule *module);
#define     G_MODULE_SUFFIX
#define     G_MODULE_EXPORT
#define     G_MODULE_IMPORT
```

Description

These functions provide a portable way to dynamically load object files (commonly known as 'plug-ins'). The current implementation supports all systems that provide an implementation of `dlopen()` (e.g. Linux/Sun), as well as HP-UX via its `shl_load()` mechanism, and Windows platforms via DLLs.

A program which wants to use these functions must be linked to the libraries output by the command **pkg-config --libs gmodule-2.0**.

To use them you must first determine whether dynamic loading is supported on the platform by calling `g_module_supported()`. If it is, you can open a module with `g_module_open()`, find the module's symbols (e.g. function names) with `g_module_symbol()`, and later close the module with `g_module_close()`. `g_module_name()` will return the file name of a currently opened module.

If any of the above functions fail, the error status can be found with `g_module_error()`.

The GModule implementation features reference counting for opened modules, and supports hook functions within a module which are called when the module is loaded and unloaded (see

GModuleCheckInit and GModuleUnload).

If your module introduces static data to common subsystems in the running program, e.g. through calling `g_quark_from_static_string ("my-module-stuff")`, it must ensure that it is never unloaded, by calling `g_module_make_resident()`.

Example 12. Calling a function defined in a GModule

```
/* the function signature for 'say_hello' */
typedef void (* SayHelloFunc) (const char *message);

gboolean
just_say_hello (const char *filename, GError **error)
{
  SayHelloFunc  say_hello;
  GModule      *module;

  module = g_module_open (filename, G_MODULE_BIND_LAZY);
  if (!module)
    {
      g_set_error (error, FOO_ERROR, FOO_ERROR_BLAH,
                   "%s", g_module_error ());
      return FALSE;
    }

  if (!g_module_symbol (module, "say_hello", (gpointer *)&say_hello))
    {
      g_set_error (error, SAY_ERROR, SAY_ERROR_OPEN,
                   "%s: %s", filename, g_module_error ());
      if (!g_module_close (module))
        g_warning ("%s: %s", filename, g_module_error ());
      return FALSE;
    }

  /* call our function in the module */
  say_hello ("Hello world!");

  if (!g_module_close (module))
    g_warning ("%s: %s", filename, g_module_error ());

  return TRUE;
}
```

Details

GModule

```
typedef struct _GModule GModule;
```

The GModule struct is an opaque data structure to represent a Dynamically-Loaded Module. It should only be accessed via the following functions.

---

g_module_supported ()

```
gboolean    g_module_supported              (void);
```

Checks if modules are supported on the current platform.

Returns : `TRUE` if modules are supported.

---

g_module_build_path ()

```
gchar*      g_module_build_path             (const gchar *directory,
                                             const gchar *module_name);
```

A portable way to build the filename of a module. The platform-specific prefix and suffix are added to the filename, if needed, and the result is added to the directory, using the correct separator character.

The directory should specify the directory where the module can be found. It can be `NULL` or an empty string to indicate that the module is in a standard platform-specific directory, though this is not recommended since the wrong module may be found.

For example, calling `g_module_build_path()` on a Linux system with a *`directory`* of `/lib` and a *`module_name`* of "mylibrary" will return `/lib/libmylibrary.so`. On a Windows system, using `\Windows` as the directory it will return `\Windows\mylibrary.dll`.

`directory` : the directory where the module is. This can be `NULL` or the empty string to indicate that the standard platform-specific directories will be used, though that is not recommended.
`module_name` : the name of the module.
Returns : the complete path of the module, including the standard library prefix and suffix. This should be freed when no longer needed.

---

g_module_open ()

```
GModule*    g_module_open                   (const gchar *file_name,
                                             GModuleFlags flags);
```

Opens a module. If the module has already been opened, its reference count is incremented.

First of all `g_module_open()` tries to open *`file_name`* as a module. If that fails and *`file_name`* has the ".la"-suffix (and is a libtool archive) it tries to open the corresponding module. If that fails and it doesn't have the proper module suffix for the platform (G_MODULE_SUFFIX), this suffix will be appended and the corresponding module will be opended. If that fails and *`file_name`* doesn't have the ".la"-suffix, this suffix is appended and `g_module_open()` tries to open the corresponding module. If eventually that fails as well, `NULL` is returned.

`file_name` : the name of the file containing the module.
`flags` : the flags used for opening the module. This can be the logical OR of any of the GModuleFlags.

Returns : a GModule on success, or `NULL` on failure.

---

enum GModuleFlags

```
typedef enum
{
  G_MODULE_BIND_LAZY    = 1 << 0,
  G_MODULE_BIND_LOCAL   = 1 << 1,
  G_MODULE_BIND_MASK    = 0x03
} GModuleFlags;
```

Flags passed to `g_module_open()`. Note that these flags are not supported on all platforms.

`G_MODULE_BIND_LAZY` specifies that symbols are only resolved when needed. The default action is to bind all symbols when the module is loaded.
`G_MODULE_BIND_LOCAL` specifies that symbols in the module should not be added to the global name space. The default action on most platforms is to place symbols in the module in the global name space, which may cause conflicts with existing symbols.
`G_MODULE_BIND_MASK` mask for all flags.

---

g_module_symbol ()

```
gboolean    g_module_symbol                 (GModule *module,
                                             const gchar *symbol_name,
                                             gpointer *symbol);
```

Gets a symbol pointer from a module.

`module` : a GModule.
`symbol_name` : the name of the symbol to find.
`symbol` : returns the pointer to the symbol value.
Returns : `TRUE` on success.

---

g_module_name ()

```
G_CONST_RETURN gchar* g_module_name         (GModule *module);
```

Gets the filename from a GModule.

`module` : a GModule.
Returns : the filename of the module, or "main" if the module is the main program itself.

---

g_module_make_resident ()

```
void        g_module_make_resident          (GModule *module);
```

Ensures that a module will never be unloaded. Any future `g_module_close()` calls on the module will be ignored.

module : a GModule to make permanently resident.

---

g_module_close ()

```
gboolean    g_module_close                  (GModule *module);
```

Closes a module.

module : a GModule to close.
Returns : `TRUE` on success.

---

g_module_error ()

```
G_CONST_RETURN gchar* g_module_error        (void);
```

Gets a string describing the last module error.

Returns : a string describing the last module error.

GModuleCheckInit ()

```
const gchar* (*GModuleCheckInit)            (GModule *module);
```

Specifies the type of the module initialization function. If a module contains a function named `g_module_check_init()` it is called automatically when the module is loaded. It is passed the GModule structure and should return `NULL` on success or a string describing the initialization error.

module : the GModule corresponding to the module which has just been loaded.
Returns : `NULL` on success, or a string describing the initialization error.

GModuleUnload ()

```
void        (*GModuleUnload)                (GModule *module);
```

Specifies the type of the module function called when it is unloaded. If a module contains a function named `g_module_unload()` it is called automatically when the module is unloaded. It is passed the GModule structure.

module : the GModule about to be unloaded.

G_MODULE_SUFFIX

```
#define G_MODULE_SUFFIX "so"
```

Expands to the proper shared library suffix for the current platform without the leading dot. For the most Unices and Linux this is "so", for some HP-UX versions this is "sl" and for Windows this is "dll".

G_MODULE_EXPORT

```
#define     G_MODULE_EXPORT
```

Used to declare functions exported by modules.

G_MODULE_IMPORT

```
#define G_MODULE_IMPORT         extern
```

Used to declare functions imported from modules.

<< Asynchronous Queues **Memory Allocation >>**

Memory Allocation

Memory Allocation — general memory-handling.

Synopsis

```
#include <glib.h>


#define     g_new                           (struct_type, n_structs)
#define     g_new0                          (struct_type, n_structs)
#define     g_renew                         (struct_type, mem, n_structs)

gpointer    g_malloc                        (gulong n_bytes);
gpointer    g_malloc0                       (gulong n_bytes);
gpointer    g_realloc                       (gpointer mem,
                                             gulong n_bytes);
gpointer    g_try_malloc                    (gulong n_bytes);
gpointer    g_try_realloc                   (gpointer mem,
                                             gulong n_bytes);

void        g_free                          (gpointer mem);

#define     g_alloca                        (size)
#define     g_newa                          (struct_type, n_structs)

#define     g_memmove                       (d,s,n)
gpointer    g_memdup                        (gconstpointer mem,
                                             guint byte_size);


            GMemVTable;
void        g_mem_set_vtable                (GMemVTable *vtable);
gboolean    g_mem_is_system_malloc          (void);

extern      GMemVTable  *glib_mem_profiler_table;
void        g_mem_profile                   (void);
```

Description

These functions provide support for allocating and freeing memory.

Note

If any call to allocate memory fails, the application is terminated. This also means that there is no need to check if the call succeeded.

Details

g_new()


```
#define     g_new(struct_type, n_structs)
```

Allocates *n_structs* elements of type *struct_type*. The returned pointer is cast to a pointer to the given type. If *count* is 0 it returns NULL.

struct_type : the type of the elements to allocate.
n_structs : the number of elements to allocate.
Returns : a pointer to the allocated memory, cast to a pointer to *struct_type*.

---

g_new0()

```
#define     g_new0(struct_type, n_structs)
```

Allocates *n_structs* elements of type *struct_type*, initialized to 0's. The returned pointer is cast to a pointer to the given type. If *count* is 0 it returns NULL.

struct_type : the type of the elements to allocate.
n_structs : the number of elements to allocate.
Returns : a pointer to the allocated memory, cast to a pointer to *struct_type*.

---

g_renew()

```
#define     g_renew(struct_type, mem, n_structs)
```

Reallocates the memory pointed to by *mem*, so that it now has space for *n_struct* elements of type *struct_type*. It returns the new address of the memory, which may have been moved.

struct_type : the type of the elements to allocate.
mem : the currently allocated memory.
n_structs : the number of elements to allocate.
Returns : a pointer to the new allocated memory, cast to a pointer to *struct_type*.

---

g_malloc ()

```
gpointer    g_malloc                        (gulong n_bytes);
```

Allocates *n_bytes* bytes of memory. If *n_bytes* is 0 it returns NULL.

n_bytes : the number of bytes to allocate.
Returns : a pointer to the allocated memory.

---

g_malloc0 ()

```
gpointer    g_malloc0                       (gulong n_bytes);
```

Allocates *n_bytes* bytes of memory, initialized to 0's. If *n_bytes* is 0 it returns NULL.

n_bytes : the number of bytes to allocate.
Returns : a pointer to the allocated memory.

---

g_realloc ()

```
gpointer    g_realloc                       (gpointer mem,
                                             gulong n_bytes);
```

Reallocates the memory pointed to by *mem*, so that it now has space for *n_bytes* bytes of memory. It returns the new address of the memory, which may have been moved. *mem* may be NULL, in which case it's considered to have zero-length. *n_bytes* may be 0, in which case NULL will be returned.

mem : the memory to reallocate.
n_bytes : new size of the memory in bytes.
Returns : the new address of the allocated memory.

---

g_try_malloc ()

```
gpointer    g_try_malloc                    (gulong n_bytes);
```

Attempts to allocate *n_bytes*, and returns NULL on failure. Contrast with g_malloc(), which aborts the program on failure.

n_bytes : number of bytes to allocate.
Returns : the allocated memory, or NULL.

---

g_try_realloc ()

```
gpointer    g_try_realloc                   (gpointer mem,
                                             gulong n_bytes);
```

Attempts to realloc *mem* to a new size, *n_bytes*, and returns NULL on failure. Contrast with g_realloc(), which aborts the program on failure. If *mem* is NULL, behaves the same as g_try_malloc().

mem : previously-allocated memory, or NULL.
n_bytes : number of bytes to allocate.

Returns : the allocated memory, or NULL.

---

g_free ()

```
void        g_free                          (gpointer mem);
```

Frees the memory pointed to by *mem*. If *mem* is NULL it simply returns.

mem : the memory to free.

---

g_alloca()

```
#define     g_alloca(size)
```

Allocates *size* bytes on the stack; these bytes will be freed when the current stack frame is cleaned up. This macro essentially just wraps the alloca() function present on most UNIX variants. Thus it provides the same advantages and pitfalls as alloca():

+ alloca() is very fast, as on most systems it's implemented by just adjusting the stack pointer register.
+ It doesn't cause any memory fragmentation, within its scope, separate alloca() blocks just build up and are released together at function end.
- Allocation sizes have to fit into the current stack frame. For instance in a threaded environment on Linux, the per-thread stack size is limited to 2 Megabytes, so be sparse with alloca() uses.
- Allocation failure due to insufficient stack space is not indicated with a NULL return like e.g. with malloc(). Instead, most systems probably handle it the same way as out of stack space situations from infinite function recursion, i.e. with a segmentation fault.
- Special care has to be taken when mixing alloca() with GNU C variable sized arrays. Stack space allocated with alloca() in the same scope as a variable sized array will be freed together with the variable sized array upon exit of that scope, and not upon exit of the enclosing function scope.

size : number of bytes to allocate.
Returns : space for *size* bytes, allocated on the stack

---

g_newa()

```
#define     g_newa(struct_type, n_structs)
```

Wraps g_alloca() in a more typesafe manner.

struct_type : Type of memory chunks to be allocated

n_structs : Number of chunks to be allocated
Returns : Pointer to stack space for *n_structs* chunks of type *struct_type*

---

g_memmove()

```
#define     g_memmove(d,s,n)
```

Copies a block of memory *n* bytes long, from *s* to *d*. The source and destination areas may overlap.

In order to use this function, you must include `string.h` yourself, because this macro will typically simply resolve to `memmove()` and GLib does not include `string.h` for you.

d : the destination address to copy the bytes to.
s : the source address to copy the bytes from.
n : the number of bytes to copy.

---

g_memdup ()

```
gpointer    g_memdup                        (gconstpointer mem,
                                             guint byte_size);
```

Allocates *byte_size* bytes of memory, and copies *byte_size* bytes into it from *mem*. If *mem* is `NULL` it returns `NULL`.

mem : the memory to copy.
byte_size : the number of bytes to copy.
Returns : a pointer to the newly-allocated copy of the memory, or `NULL` if *mem* is `NULL`.

GMemVTable

```
typedef struct {
  gpointer (*malloc)      (gsize    n_bytes);
  gpointer (*realloc)     (gpointer mem,
                           gsize    n_bytes);
  void     (*free)        (gpointer mem);
  /* optional; set to NULL if not used ! */
  gpointer (*calloc)      (gsize    n_blocks,
                           gsize    n_block_bytes);
  gpointer (*try_malloc)  (gsize    n_bytes);
  gpointer (*try_realloc) (gpointer mem,
                           gsize    n_bytes);
} GMemVTable;
```

A set of functions used to perform memory allocation. The same GMemVTable must be used for all allocations in the same program; a call to `g_mem_set_vtable()`, if it exists, should be prior to any use of GLib.

malloc () function to use for allocating memory.
realloc () function to use for reallocating memory.
free () function to use to free memory.
calloc () function to use for allocating zero-filled memory.
try_malloc () function to use for allocating memory without a default error handler.
try_realloc () function to use for reallocating memory without a default error handler.

---

g_mem_set_vtable ()

```
void        g_mem_set_vtable                (GMemVTable *vtable);
```

Sets the GMemVTable to use for memory allocation. You can use this to provide custom memory allocation routines. *This function must be called before using any other GLib functions.* The *vtable* only needs to provide `malloc()`, `realloc()`, and `free()` functions; GLib can provide default implementations of the others. The `malloc()` and `realloc()` implementations should return `NULL` on failure, GLib will handle error-checking for you. *vtable* is copied, so need not persist after this function has been called.

vtable : table of memory allocation routines.

---

g_mem_is_system_malloc ()

```
gboolean    g_mem_is_system_malloc          (void);
```

Checks whether the allocator used by `g_malloc()` is the system's malloc implementation. If it returns `TRUE` memory allocated with `malloc()` can be used interchangeable with memory allocated using `g_malloc()`. This function is useful for avoiding an extra copy of allocated memory returned by a non-GLib-based API.

A different allocator can be set using `g_mem_set_vtable()`.

Returns : if `TRUE`, `malloc()` and `g_malloc()` can be mixed.

---

glib_mem_profiler_table

```
extern GMemVTable       *glib_mem_profiler_table;
```

A GMemVTable containing profiling variants of the memory allocation functions. Use them together with `g_mem_profile()` in order to get information about the memory allocation pattern of your program.

---

g_mem_profile ()

```
void        g_mem_profile                   (void);
```

Outputs a summary of memory usage.

It outputs the frequency of allocations of different sizes, the total number of bytes which have been allocated, the total number of bytes which have been freed, and the difference between the previous two values, i.e. the number of bytes still in use.

Note that this function will not output anything unless you have previously installed the glib_mem_profiler_table with `g_mem_set_vtable()`.

<< Dynamic Loading of Modules **IO Channels >>**

IO Channels

IO Channels — portable support for using files, pipes and sockets.

Synopsis

```
#include <glib.h>


            GIOChannel;

GIOChannel* g_io_channel_unix_new           (int fd);
gint        g_io_channel_unix_get_fd        (GIOChannel *channel);

void        g_io_channel_init               (GIOChannel *channel);

GIOChannel* g_io_channel_new_file           (const gchar *filename,
                                             const gchar *mode,
                                             GError **error);
GIOStatus   g_io_channel_read_chars         (GIOChannel *channel,
                                             gchar *buf,
                                             gsize count,
                                             gsize *bytes_read,
                                             GError **error);
GIOStatus   g_io_channel_read_unichar       (GIOChannel *channel,
                                             gunichar *thechar,
                                             GError **error);
GIOStatus   g_io_channel_read_line          (GIOChannel *channel,
                                             gchar **str_return,
                                             gsize *length,
                                             gsize *terminator_pos,
                                             GError **error);
GIOStatus   g_io_channel_read_line_string   (GIOChannel *channel,
                                             GString *buffer,
                                             gsize *terminator_pos,
                                             GError **error);
GIOStatus   g_io_channel_read_to_end        (GIOChannel *channel,
                                             gchar **str_return,
                                             gsize *length,
                                             GError **error);
GIOStatus   g_io_channel_write_chars        (GIOChannel *channel,
                                             const gchar *buf,
                                             gssize count,
                                             gsize *bytes_written,
                                             GError **error);
GIOStatus   g_io_channel_write_unichar      (GIOChannel *channel,
                                             gunichar thechar,
                                             GError **error);
GIOStatus   g_io_channel_flush              (GIOChannel *channel,
                                             GError **error);
GIOStatus   g_io_channel_seek_position      (GIOChannel *channel,
                                             gint64 offset,
                                             GSeekType type,
                                             GError **error);
enum        GSeekType;
```



```
GIOStatus   g_io_channel_shutdown           (GIOChannel *channel,
                                             gboolean flush,
                                             GError **err);

enum        GIOStatus;
enum        GIOChannelError;
#define     G_IO_CHANNEL_ERROR
GIOChannelError g_io_channel_error_from_errno
                                            (gint en);

GIOChannel* g_io_channel_ref                (GIOChannel *channel);
void        g_io_channel_unref              (GIOChannel *channel);

GSource*    g_io_create_watch               (GIOChannel *channel,
                                             GIOCondition condition);
guint       g_io_add_watch                  (GIOChannel *channel,
                                             GIOCondition condition,
                                             GIOFunc func,
                                             gpointer user_data);
guint       g_io_add_watch_full             (GIOChannel *channel,
                                             gint priority,
                                             GIOCondition condition,
                                             GIOFunc func,
                                             gpointer user_data,
                                             GDestroyNotify notify);
enum        GIOCondition;
gboolean    (*GIOFunc)                      (GIOChannel *source,
                                             GIOCondition condition,
                                             gpointer data);

            GIOFuncs;

gsize       g_io_channel_get_buffer_size    (GIOChannel *channel);
void        g_io_channel_set_buffer_size    (GIOChannel *channel,
                                             gsize size);
GIOCondition g_io_channel_get_buffer_condition
                                            (GIOChannel *channel);
GIOFlags    g_io_channel_get_flags          (GIOChannel *channel);
GIOStatus   g_io_channel_set_flags          (GIOChannel *channel,
                                             GIOFlags flags,
                                             GError **error);

enum        GIOFlags;
G_CONST_RETURN gchar* g_io_channel_get_line_term
                                            (GIOChannel *channel,
                                             gint *length);
void        g_io_channel_set_line_term      (GIOChannel *channel,
                                             const gchar *line_term,
                                             gint length);
gboolean    g_io_channel_get_buffered       (GIOChannel *channel);
void        g_io_channel_set_buffered       (GIOChannel *channel,
                                             gboolean buffered);
G_CONST_RETURN gchar* g_io_channel_get_encoding
                                            (GIOChannel *channel);
GIOStatus   g_io_channel_set_encoding       (GIOChannel *channel,
                                             const gchar *encoding,
                                             GError **error);
gboolean    g_io_channel_get_close_on_unref (GIOChannel *channel);
void        g_io_channel_set_close_on_unref (GIOChannel *channel,
                                             gboolean do_close);


GIOError    g_io_channel_read               (GIOChannel *channel,
                                             gchar *buf,
                                             gsize count,
```

```
                                        gsize *bytes_read);
enum        GIOError;
GIOError    g_io_channel_write          (GIOChannel *channel,
                                         const gchar *buf,
                                         gsize count,
                                         gsize *bytes_written);
GIOError    g_io_channel_seek           (GIOChannel *channel,
                                         gint64 offset,
                                         GSeekType type);
void        g_io_channel_close          (GIOChannel *channel);
```

Description

The GIOChannel data type aims to provide a portable method for using file descriptors, pipes, and sockets, and integrating them into the main event loop. Currently full support is available on UNIX platforms, support for Windows is only partially complete.

To create a new GIOChannel on UNIX systems use `g_io_channel_unix_new()`. This works for plain file descriptors, pipes and sockets. Alternatively, a channel can be created for a file in a system independent manner using `g_io_channel_new_file()`.

Once a GIOChannel has been created, it can be used in a generic manner with the functions `g_io_channel_read_chars()`, `g_io_channel_write_chars()`, `g_io_channel_seek_position()`, and `g_io_channel_close()`.

To add a GIOChannel to the main event loop use `g_io_add_watch()` or `g_io_add_watch_full()`. Here you specify which events you are interested in on the GIOChannel, and provide a function to be called whenever these events occur.

GIOChannel instances are created with an initial reference count of 1. `g_io_channel_ref()` and `g_io_channel_unref()` can be used to increment or decrement the reference count respectively. When the reference count falls to 0, the GIOChannel is freed. (Though it isn't closed automatically, unless it was created using `g_io_channel_new_from_file()`.) Using `g_io_add_watch()` or `g_io_add_watch_full()` increments a channel's reference count.

The new functions `g_io_channel_read_chars()`, `g_io_channel_read_line()`, `g_io_channel_read_line_string()`, `g_io_channel_read_to_end()`, `g_io_channel_write_chars()`, `g_io_channel_seek_position()`, and `g_io_channel_flush()` should not be mixed with the deprecated functions `g_io_channel_read()`, `g_io_channel_write()`, and `g_io_channel_seek()` on the same channel.

Details

GIOChannel

```
typedef struct {
} GIOChannel;
```

A data structure representing an IO Channel. The fields should be considered private and should only be accessed with the following functions.

---

g_io_channel_unix_new ()

```
GIOChannel* g_io_channel_unix_new           (int fd);
```

Creates a new GIOChannel given a file descriptor. On UNIX systems this works for plain files, pipes, and sockets.

The returned GIOChannel has a reference count of 1.

The default encoding for GIOChannel is UTF-8. If your application is reading output from a command using via pipe, you may need to set the encoding to the encoding of the current locale (see `g_get_charset()`) with the `g_io_channel_set_encoding()` function.

If you want to read raw binary data without interpretation, then call the `g_io_charset_set_encoding()` function with `NULL` for the encoding argument.

fd : a file descriptor.
Returns : a new GIOChannel.

---

g_io_channel_unix_get_fd ()

```
gint        g_io_channel_unix_get_fd        (GIOChannel *channel);
```

Returns the file descriptor of the UNIX GIOChannel.

channel : a GIOChannel, created with `g_io_channel_unix_new()`.
Returns : the file descriptor of the GIOChannel.

---

g_io_channel_init ()

```
void        g_io_channel_init               (GIOChannel *channel);
```

Initializes a GIOChannel struct. This is called by each of the above functions when creating a GIOChannel, and so is not often needed by the application programmer (unless you are creating a new type of GIOChannel).

channel : a GIOChannel.

---

g_io_channel_new_file ()

```
GIOChannel* g_io_channel_new_file           (const gchar *filename,
                                             const gchar *mode,
                                             GError **error);
```

Open a file *filename* as a GIOChannel using mode *mode*. This channel will be closed when the last reference to it is dropped, so there is no need to call g_io_channel_close() (though doing so will not cause problems, as long as no attempt is made to access the channel after it is closed).

filename : A string containing the name of a file.
mode : One of "r", "w", "a", "r+", "w+", "a+". These have the same meaning as in fopen().
error : A location to return an error of type G_FILE_ERROR.
Returns : A GIOChannel on success, NULL on failure.

---

g_io_channel_read_chars ()

```
GIOStatus   g_io_channel_read_chars         (GIOChannel *channel,
                                             gchar *buf,
                                             gsize count,
                                             gsize *bytes_read,
                                             GError **error);
```

Replacement for g_io_channel_read() with the new API.

channel : a GIOChannel
buf : a buffer to read data into
count : the size of the buffer. Note that the buffer may not be complelely filled even if there is data in the buffer if the remaining data is not a complete character.
bytes_read : The number of bytes read. This may be zero even on success if count < 6 and the channel's encoding is non-NULL. This indicates that the next UTF-8 character is too wide for the buffer.
error : A location to return an error of type GConvertError or GIOChannelError.
Returns : the status of the operation.

---

g_io_channel_read_unichar ()

```
GIOStatus   g_io_channel_read_unichar       (GIOChannel *channel,
                                             gunichar *thechar,
                                             GError **error);
```

This function cannot be called on a channel with NULL encoding.

channel : a GIOChannel
thechar : a location to return a character
error : A location to return an error of type GConvertError or GIOChannelError
Returns : a GIOStatus

---

g_io_channel_read_line ()


```
GIOStatus   g_io_channel_read_line          (GIOChannel *channel,
                                             gchar **str_return,
                                             gsize *length,
                                             gsize *terminator_pos,
                                             GError **error);
```

Reads a line, including the terminating character(s), from a GIOChannel into a newly-allocated string. *str_return* will contain allocated memory if the return is G_IO_STATUS_NORMAL.

channel : a GIOChannel
str_return : The line read from the GIOChannel, including the line terminator. This data should be freed with g_free() when no longer needed. This is a nul-terminated string. If a *length* of zero is returned, this will be NULL instead.
length : location to store length of the read data, or NULL
terminator_pos : location to store position of line terminator, or NULL
error : A location to return an error of type GConvertError or GIOChannelError
Returns : the status of the operation.

---

g_io_channel_read_line_string ()

```
GIOStatus   g_io_channel_read_line_string   (GIOChannel *channel,
                                             GString *buffer,
                                             gsize *terminator_pos,
                                             GError **error);
```

Reads a line from a GIOChannel, using a GString as a buffer.

channel : a GIOChannel
buffer : a GString into which the line will be written. If *buffer* already contains data, the old data will be overwritten.
terminator_pos : location to store position of line terminator, or NULL
error : a location to store an error of type GConvertError or GIOChannelError
Returns : the status of the operation.

---

g_io_channel_read_to_end ()

```
GIOStatus   g_io_channel_read_to_end        (GIOChannel *channel,
                                             gchar **str_return,
                                             gsize *length,
                                             GError **error);
```

Reads all the remaining data from the file.

channel : a GIOChannel
str_return : Location to store a pointer to a string holding the remaining data in the

GIOChannel. This data should be freed with g_free() when no longer needed. This data is terminated by an extra nul character, but there may be other nuls in the intervening data.

length : Location to store length of the data

error : A location to return an error of type GConvertError or GIOChannelError

Returns : G_IO_STATUS_NORMAL on success. This function never returns G_IO_STATUS_EOF.

---

g_io_channel_write_chars ()

```
GIOStatus   g_io_channel_write_chars        (GIOChannel *channel,
                                             const gchar *buf,
                                             gssize count,
                                             gsize *bytes_written,
                                             GError **error);
```

Replacement for g_io_channel_write() with the new API.

On seekable channels with encodings other than NULL or UTF-8, generic mixing of reading and writing is not allowed. A call to g_io_channel_write_chars() may only be made on a channel from which data has been read in the cases described in the documentation for g_io_channel_set_encoding().

channel : a GIOChannel

buf : a buffer to write data from

count : the size of the buffer. If -1, the buffer is taken to be a nul-terminated string.

bytes_written : The number of bytes written. This can be nonzero even if the return value is not G_IO_STATUS_NORMAL. If the return value is G_IO_STATUS_NORMAL and the channel is blocking, this will always be equal to *count* if *count* >= 0.

error : A location to return an error of type GConvertError or GIOChannelError

Returns : the status of the operation.

---

g_io_channel_write_unichar ()

```
GIOStatus   g_io_channel_write_unichar      (GIOChannel *channel,
                                             gunichar thechar,
                                             GError **error);
```

This function cannot be called on a channel with NULL encoding.

channel : a GIOChannel

thechar : a character

error : A location to return an error of type GConvertError or GIOChannelError

Returns : a GIOStatus

---

g_io_channel_flush ()

```
GIOStatus   g_io_channel_flush              (GIOChannel *channel,
                                             GError **error);
```

Flushes the write buffer for the GIOChannel.

channel : a GIOChannel

error : location to store an error of type GIOChannelError

Returns : the status of the operation: One of G_IO_CHANNEL_NORMAL, G_IO_CHANNEL_AGAIN, or G_IO_CHANNEL_ERROR.

---

g_io_channel_seek_position ()

```
GIOStatus   g_io_channel_seek_position      (GIOChannel *channel,
                                             gint64 offset,
                                             GSeekType type,
                                             GError **error);
```

Replacement for g_io_channel_seek() with the new API.

channel : a GIOChannel

offset : The offset in bytes from the position specified by *type*

type : a GSeekType. The type G_SEEK_CUR is only allowed in those cases where a call to g_io_channel_set_encoding() is allowed. See the documentation for g_io_channel_set_encoding() for details.

error : A location to return an error of type GIOChannelError

Returns : the status of the operation.

---

enum GSeekType

```
typedef enum
{
  G_SEEK_CUR,
  G_SEEK_SET,
  G_SEEK_END
} GSeekType;
```

An enumeration specifying the base position for a g_io_channel_seek_position() operation.

G_SEEK_CUR the current position in the file.

G_SEEK_SET the start of the file.

G_SEEK_END the end of the file.

---

g_io_channel_shutdown ()

```
GIOStatus   g_io_channel_shutdown           (GIOChannel *channel,
                                             gboolean flush,
                                             GError **err);
```

Close an IO channel. Any pending data to be written will be flushed if *flush* is TRUE. The channel will not be freed until the last reference is dropped using g_io_channel_unref().

channel : a GIOChannel
flush : if TRUE, flush pending
err : location to store a GIOChannelError
Returns : the status of the operation.

---

enum GIOStatus

```
typedef enum
{
  G_IO_STATUS_ERROR,
  G_IO_STATUS_NORMAL,
  G_IO_STATUS_EOF,
  G_IO_STATUS_AGAIN
} GIOStatus;
```

Stati returned by most of the GIOFuncs functions.

G_IO_STATUS_ERROR	An error occurred.
G_IO_STATUS_NORMAL	Success.
G_IO_STATUS_EOF	End of file.
G_IO_STATUS_AGAIN	Resource temporarily unavailable.

---

enum GIOChannelError

```
typedef enum
{
  /* Derived from errno */
  G_IO_CHANNEL_ERROR_FBIG,
  G_IO_CHANNEL_ERROR_INVAL,
  G_IO_CHANNEL_ERROR_IO,
  G_IO_CHANNEL_ERROR_ISDIR,
  G_IO_CHANNEL_ERROR_NOSPC,
  G_IO_CHANNEL_ERROR_NXIO,
  G_IO_CHANNEL_ERROR_OVERFLOW,
  G_IO_CHANNEL_ERROR_PIPE,
  /* Other */
  G_IO_CHANNEL_ERROR_FAILED
} GIOChannelError;
```

Error codes returned by GIOChannel operations.

G_IO_CHANNEL_ERROR_FBIG	File too large.
G_IO_CHANNEL_ERROR_INVAL	Invalid argument.
G_IO_CHANNEL_ERROR_IO	IO error.
G_IO_CHANNEL_ERROR_ISDIR	File is a directory.
G_IO_CHANNEL_ERROR_NOSPC	No space left on device.
G_IO_CHANNEL_ERROR_NXIO	No such device or address.
G_IO_CHANNEL_ERROR_OVERFLOW	Value too large for defined datatype.
G_IO_CHANNEL_ERROR_PIPE	Broken pipe.
G_IO_CHANNEL_ERROR_FAILED	Some other error.

G_IO_CHANNEL_ERROR

```
#define G_IO_CHANNEL_ERROR g_io_channel_error_quark()
```

Error domain for GIOChannel operations. Errors in this domain will be from the GIOChannelError enumeration. See GError for information on error domains.

---

g_io_channel_error_from_errno ()

```
GIOChannelError g_io_channel_error_from_errno
                                            (gint en);
```

Converts an errno error number to a GIOChannelError.

en : an errno error number, e.g. EINVAL.
Returns : a GIOChannelError error number, e.g. G_IO_CHANNEL_ERROR_INVAL.

---

g_io_channel_ref ()

```
GIOChannel* g_io_channel_ref                (GIOChannel *channel);
```

Increments the reference count of a GIOChannel.

channel : a GIOChannel.
Returns : the *channel* that was passed in (since 2.6)

---

g_io_channel_unref ()

```
void        g_io_channel_unref              (GIOChannel *channel);
```

Decrements the reference count of a GIOChannel.

channel : a GIOChannel.

---

g_io_create_watch ()

```
GSource*    g_io_create_watch               (GIOChannel *channel,
                                             GIOCondition condition);
```

Creates a GSource that's dispatched when *condition* is met for the given *channel*. For example, if condition is G_IO_IN, the source will be dispatched when there's data available for reading. g_io_add_watch() is a simpler interface to this same functionality, for the case where you want to add the source to the default main loop at the default priority.

channel : a GIOChannel to watch
condition : conditions to watch for
Returns : a new GSource

---

g_io_add_watch ()

```
guint       g_io_add_watch                  (GIOChannel *channel,
                                             GIOCondition condition,
                                             GIOFunc func,
                                             gpointer user_data);
```

Adds the GIOChannel into the main event loop with the default priority.

channel : a GIOChannel.
condition : the condition to watch for.
func : the function to call when the condition is satisfied.
user_data : user data to pass to *func*.
Returns : the event source id.

---

g_io_add_watch_full ()

```
guint       g_io_add_watch_full             (GIOChannel *channel,
                                             gint priority,
                                             GIOCondition condition,
                                             GIOFunc func,
                                             gpointer user_data,
                                             GDestroyNotify notify);
```

Adds the GIOChannel into the main event loop with the given priority.

channel : a GIOChannel.
priority : the priority of the GIOChannel source.
condition : the condition to watch for.

func : the function to call when the condition is satisfied.
user_data : user data to pass to *func*.
notify : the function to call when the source is removed.
Returns : the event source id.

---

enum GIOCondition

```
typedef enum
{
  G_IO_IN       GLIB_SYSDEF_POLLIN,
  G_IO_OUT      GLIB_SYSDEF_POLLOUT,
  G_IO_PRI      GLIB_SYSDEF_POLLPRI,
  G_IO_ERR      GLIB_SYSDEF_POLLERR,
  G_IO_HUP      GLIB_SYSDEF_POLLHUP,
  G_IO_NVAL     GLIB_SYSDEF_POLLNVAL
} GIOCondition;
```

A bitwise combination representing a condition to watch for on an event source.

G_IO_IN	There is data to read.
G_IO_OUT	Data can be written (without blocking).
G_IO_PRI	There is urgent data to read.
G_IO_ERR	Error condition.
G_IO_HUP	Hung up (the connection has been broken, usually for pipes and sockets).
G_IO_NVAL	Invalid request. The file descriptor is not open.

GIOFunc ()

```
gboolean    (*GIOFunc)                      (GIOChannel *source,
                                             GIOCondition condition,
                                             gpointer data);
```

Specifies the type of function passed to g_io_add_watch() or g_io_add_watch_full(), which is called when the requested condition on a GIOChannel is satisfied.

source : the GIOChannel event source.
condition : the condition which has been satisfied.
data : user data set in g_io_add_watch() or g_io_add_watch_full().
Returns : the function should return FALSE if the event source should be removed.

GIOFuncs

```
typedef struct {
  GIOStatus (*io_read)          (GIOChannel   *channel,
                                 gchar        *buf,
                                 gsize         count,
```

```
                                gsize        *bytes_read,
                                GError      **err);
  GIOStatus (*io_write)        (GIOChannel   *channel,
                                const gchar  *buf,
                                gsize         count,
                                gsize        *bytes_written,
                                GError      **err);
  GIOStatus (*io_seek)         (GIOChannel   *channel,
                                gint64        offset,
                                GSeekType     type,
                                GError      **err);
  GIOStatus  (*io_close)       (GIOChannel   *channel,
                                GError      **err);
  GSource*   (*io_create_watch) (GIOChannel   *channel,
                                 GIOCondition  condition);
  void       (*io_free)        (GIOChannel   *channel);
  GIOStatus  (*io_set_flags)   (GIOChannel   *channel,
                                GIOFlags      flags,
                                GError      **err);
  GIOFlags   (*io_get_flags)   (GIOChannel   *channel);
} GIOFuncs;
```

A table of functions used to handle different types of GIOChannel in a generic way.

---

g_io_channel_get_buffer_size ()

```
gsize       g_io_channel_get_buffer_size    (GIOChannel *channel);
```

Gets the buffer size.

channel : a GIOChannel
Returns : the size of the buffer.

---

g_io_channel_set_buffer_size ()

```
void        g_io_channel_set_buffer_size    (GIOChannel *channel,
                                             gsize size);
```

Sets the buffer size.

channel : a GIOChannel
size : the size of the buffer. 0 == pick a good size

---

g_io_channel_get_buffer_condition ()

```
GIOCondition g_io_channel_get_buffer_condition
                                            (GIOChannel *channel);
```


This function returns a GIOCondition depending on whether there is data to be read/space to write data in the internal buffers in the GIOChannel. Only the flags G_IO_IN and G_IO_OUT may be set.

channel : A GIOChannel
Returns : A GIOCondition

---

g_io_channel_get_flags ()

```
GIOFlags    g_io_channel_get_flags          (GIOChannel *channel);
```

Gets the current flags for a GIOChannel, including read-only flags such as G_IO_FLAG_IS_READABLE.

The values of the flags G_IO_FLAG_IS_READABLE and G_IO_FLAG_IS_WRITEABLE are cached for internal use by the channel when it is created. If they should change at some later point (e.g. partial shutdown of a socket with the UNIX shutdown() function), the user should immediately call g_io_channel_get_flags() to update the internal values of these flags.

channel : a GIOChannel
Returns : the flags which are set on the channel

---

g_io_channel_set_flags ()

```
GIOStatus   g_io_channel_set_flags          (GIOChannel *channel,
                                             GIOFlags flags,
                                             GError **error);
```

Sets the (writeable) flags in *channel* to (*flags* & G_IO_CHANNEL_SET_MASK).

channel : a GIOChannel.
flags : the flags to set on the IO channel.
error : A location to return an error of type GIOChannelError.
Returns : the status of the operation.

---

enum GIOFlags

```
typedef enum
{
  G_IO_FLAG_APPEND = 1 << 0,
  G_IO_FLAG_NONBLOCK = 1 << 1,
  G_IO_FLAG_IS_READABLE = 1 << 2,       /* Read only flag */
  G_IO_FLAG_IS_WRITEABLE = 1 << 3,      /* Read only flag */
  G_IO_FLAG_IS_SEEKABLE = 1 << 4,       /* Read only flag */
  G_IO_FLAG_MASK = (1 << 5) - 1,
  G_IO_FLAG_GET_MASK = G_IO_FLAG_MASK,
  G_IO_FLAG_SET_MASK = G_IO_FLAG_APPEND | G_IO_FLAG_NONBLOCK
```

```
} GIOFlags;
```

Specifies properties of a GIOChannel. Some of the flags can only be read with `g_io_channel_get_flags()`, but not changed with `g_io_channel_set_flags()`.

`G_IO_FLAG_APPEND`	turns on append mode, corresponds to `O_APPEND` (see the documentation of the UNIX `open()` syscall).
`G_IO_FLAG_NONBLOCK`	turns on nonblocking mode, corresponds to `O_NONBLOCK`/`O_NDELAY` (see the documentation of the UNIX `open()` syscall).
`G_IO_FLAG_IS_READABLE`	indicates that the io channel is readable. This flag can not be changed.
`G_IO_FLAG_IS_WRITEABLE`	indicates that the io channel is writable. This flag can not be changed.
`G_IO_FLAG_IS_SEEKABLE`	indicates that the io channel is seekable, i.e. that `g_io_channel_seek_position()` can be used on it. This flag can not be changed.
`G_IO_FLAG_MASK`	
`G_IO_FLAG_GET_MASK`	
`G_IO_FLAG_SET_MASK`	

---

g_io_channel_get_line_term ()

```
G_CONST_RETURN gchar* g_io_channel_get_line_term
                                        (GIOChannel *channel,
                                         gint *length);
```

This returns the string that GIOChannel uses to determine where in the file a line break occurs. A value of `NULL` indicates auto detection.

channel : a GIOChannel

length : a location to return the length of the line terminator

Returns : The line termination string. This value is owned by GLib and must not be freed.

---

g_io_channel_set_line_term ()

```
void        g_io_channel_set_line_term      (GIOChannel *channel,
                                             const gchar *line_term,
                                             gint length);
```

This sets the string that GIOChannel uses to determine where in the file a line break occurs.

channel : a GIOChannel

line_term : The line termination string. Use `NULL` for auto detect. Auto detection breaks on "\n", "\r\n", "\r", "\0", and the Unicode paragraph separator. Auto detection should not be used for anything other than file-based channels.

length : The length of the termination string. If -1 is passed, the string is assumed to be nul-terminated. This option allows termination strings with embeded nuls.

---

g_io_channel_get_buffered ()

```
gboolean    g_io_channel_get_buffered       (GIOChannel *channel);
```

Returns whether *channel* is buffered.

channel : a GIOChannel.

Returns : `TRUE` if the *channel* is buffered.

---

g_io_channel_set_buffered ()

```
void        g_io_channel_set_buffered       (GIOChannel *channel,
                                             gboolean buffered);
```

The buffering state can only be set if the channel's encoding is `NULL`. For any other encoding, the channel must be buffered.

A buffered channel can only be set unbuffered if the channel's internal buffers have been flushed. Newly created channels or channels which have returned `G_IO_STATUS_EOF` not require such a flush. For write-only channels, a call to `g_io_channel_flush()` is sufficient. For all other channels, the buffers may be flushed by a call to `g_io_channel_seek_position()`. This includes the possibility of seeking with seek type `G_SEEK_CUR` and an offset of zero. Note that this means that socket-based channels cannot be set unbuffered once they have had data read from them.

On unbuffered channels, it is safe to mix read and write calls from the new and old APIs, if this is necessary for maintaining old code.

The default state of the channel is buffered.

channel : a GIOChannel

buffered : whether to set the channel buffered or unbuffered

---

g_io_channel_get_encoding ()

```
G_CONST_RETURN gchar* g_io_channel_get_encoding
                                            (GIOChannel *channel);
```

Gets the encoding for the input/output of the channel. The internal encoding is always UTF-8. The encoding `NULL` makes the channel safe for binary data.

channel : a GIOChannel

Returns : A string containing the encoding, this string is owned by GLib and must not be

freed.

---

g_io_channel_set_encoding ()

```
GIOStatus   g_io_channel_set_encoding       (GIOChannel *channel,
                                             const gchar *encoding,
                                             GError **error);
```

Sets the encoding for the input/output of the channel. The internal encoding is always UTF-8. The default encoding for the external file is UTF-8.

The encoding NULL is safe to use with binary data.

The encoding can only be set if one of the following conditions is true:

1. The channel was just created, and has not been written to or read from yet.

2. The channel is write-only.

3. The channel is a file, and the file pointer was just repositioned by a call to g_io_channel_seek_position(). (This flushes all the internal buffers.)

4. The current encoding is NULL or UTF-8.

5. One of the (new API) read functions has just returned G_IO_STATUS_EOF (or, in the case of g_io_channel_read_to_end(), G_IO_STATUS_NORMAL).

6. One of the functions g_io_channel_read_chars() or g_io_channel_read_unichar() has returned G_IO_STATUS_AGAIN or G_IO_STATUS_ERROR. This may be useful in the case of G_CONVERT_ERROR_ILLEGAL_SEQUENCE. Returning one of these statuses from g_io_channel_read_line(), g_io_channel_read_line_string(), or g_io_channel_read_to_end() does *not* guarantee that the encoding can be changed.

Channels which do not meet one of the above conditions cannot call g_io_channel_seek_position() with an offset of G_SEEK_CUR, and, if they are "seekable", cannot call g_io_channel_write_chars() after calling one of the API "read" functions.

channel : a GIOChannel
encoding : the encoding type
error : location to store an error of type GConvertError.
Returns : G_IO_STATUS_NORMAL if the encoding was successfully set.

---

g_io_channel_get_close_on_unref ()

```
gboolean    g_io_channel_get_close_on_unref (GIOChannel *channel);
```

Returns whether the file/socket/whatever associated with *channel* will be closed when *channel*

receives its final unref and is destroyed. The default value of this is TRUE for channels created by g_io_channel_new_file(), and FALSE for all other channels.

channel : a GIOChannel.
Returns : Whether the channel will be closed on the final unref of the GIOChannel data structure.

---

g_io_channel_set_close_on_unref ()

```
void        g_io_channel_set_close_on_unref (GIOChannel *channel,
                                             gboolean do_close);
```

Setting this flag to TRUE for a channel you have already closed can cause problems.

channel : a GIOChannel
do_close : Whether to close the channel on the final unref of the GIOChannel data structure. The default value of this is TRUE for channels created by g_io_channel_new_file(), and FALSE for all other channels.

---

g_io_channel_read ()

```
GIOError    g_io_channel_read               (GIOChannel *channel,
                                             gchar *buf,
                                             gsize count,
                                             gsize *bytes_read);
```

Warning

g_io_channel_read is deprecated and should not be used in newly-written code. Use g_io_channel_read_chars() instead.

Reads data from a GIOChannel.

channel : a GIOChannel.
buf : a buffer to read the data into (which should be at least count bytes long).
count : the number of bytes to read from the GIOChannel.
bytes_read : returns the number of bytes actually read.
Returns : G_IO_ERROR_NONE if the operation was successful.

---

enum GIOError

```
typedef enum
{
  G_IO_ERROR_NONE,
  G_IO_ERROR_AGAIN,
  G_IO_ERROR_INVAL,
```

```
  G_IO_ERROR_UNKNOWN
} GIOError;
```

GIOError is only used by the deprecated functions g_io_channel_read(), g_io_channel_write(), and g_io_channel_seek().

---

g_io_channel_write ()

```
GIOError    g_io_channel_write              (GIOChannel *channel,
                                             const gchar *buf,
                                             gsize count,
                                             gsize *bytes_written);
```

Warning

g_io_channel_write is deprecated and should not be used in newly-written code. Use g_io_channel_write_chars() instead.

Writes data to a GIOChannel.

channel : a GIOChannel.
buf : the buffer containing the data to write.
count : the number of bytes to write.
bytes_written : the number of bytes actually written.
Returns : G_IO_ERROR_NONE if the operation was successful.

---

g_io_channel_seek ()

```
GIOError    g_io_channel_seek               (GIOChannel *channel,
                                             gint64 offset,
                                             GSeekType type);
```

Warning

g_io_channel_seek is deprecated and should not be used in newly-written code. Use g_io_channel_seek_position() instead.

Sets the current position in the GIOChannel, similar to the standard library function fseek().

channel : a GIOChannel.
offset : an offset, in bytes, which is added to the position specified by *type*
type : the position in the file, which can be G_SEEK_CUR (the current position), G_SEEK_SET (the start of the file), or G_SEEK_END (the end of the file).
Returns : G_IO_ERROR_NONE if the operation was successful.

---

g_io_channel_close ()

```
void        g_io_channel_close              (GIOChannel *channel);
```

Warning

g_io_channel_close is deprecated and should not be used in newly-written code. Use g_io_channel_shutdown() instead.

Close an IO channel. Any pending data to be written will be flushed, ignoring errors. The channel will not be freed until the last reference is dropped using g_io_channel_unref().

channel : A GIOChannel

See Also

gtk_input_add_full(), gtk_input_remove(), gdk_input_add(), gdk_input_add_full(), gdk_input_remove()	Convenience functions for creating GIOChannel instances and adding them to the main event loop.

<< Memory Allocation **Error Reporting >>**

Error Reporting

Error Reporting — a system for reporting errors.

Synopsis

```
#include <glib.h>


            GError;
GError*     g_error_new                     (GQuark domain,
                                             gint code,
                                             const gchar *format,
                                             ...);
GError*     g_error_new_literal             (GQuark domain,
                                             gint code,
                                             const gchar *message);
void        g_error_free                    (GError *error);
GError*     g_error_copy                    (const GError *error);
gboolean    g_error_matches                 (const GError *error,
                                             GQuark domain,
                                             gint code);
void        g_set_error                     (GError **err,
                                             GQuark domain,
                                             gint code,
                                             const gchar *format,
                                             ...);
void        g_propagate_error               (GError **dest,
                                             GError *src);
void        g_clear_error                   (GError **err);
```

Description

GLib provides a standard method of reporting errors from a called function to the calling code. (This is the same problem solved by exceptions in other languages.) It's important to understand that this method is both a *data type* (the GError object) and a *set of rules*. If you use GError incorrectly, then your code will not properly interoperate with other code that uses GError, and users of your API will probably get confused.

First and foremost: *GError should only be used to report recoverable runtime errors, never to report programming errors.* If the programmer has screwed up, then you should use `g_warning()`, `g_return_if_fail()`, `g_assert()`, `g_error()`, or some similar facility. (Incidentally, remember that the `g_error()` function should *only* be used for programming errors, it should not be used to print any error reportable via GError.)

Examples of recoverable runtime errors are "file not found" or "failed to parse input." Examples of programming errors are "NULL passed to `strcmp()`" or "attempted to free the same pointer twice." These two kinds of errors are fundamentally different: runtime errors should be handled or reported to the user, programming errors should be eliminated by fixing the bug in the program. This is why most functions in GLib and GTK+ do not use the GError facility.

Functions that can fail take a return location for a GError as their last argument. For example:

```
gboolean g_file_get_contents (const gchar *filename,
                              gchar      **contents,
                              gsize       *length,
                              GError     **error);
```

If you pass a non-`NULL` value for the `error` argument, it should point to a location where an error can be placed. For example:

```
gchar *contents;
GError *err = NULL;
g_file_get_contents ("foo.txt", &contents, NULL, &err);
g_assert ((contents == NULL && err != NULL) || (contents != NULL && err == NULL)
if (err != NULL)
  {
    /* Report error to user, and free error */
    g_assert (contents == NULL);
    fprintf (stderr, "Unable to read file: %s\n", err->message);
    g_error_free (err);
  }
else
  {
    /* Use file contents */
    g_assert (contents != NULL);
  }
```

Note that `err != NULL` in this example is a *reliable* indicator of whether `g_file_get_contents()` failed. Additionally, `g_file_get_contents()` returns a boolean which indicates whether it was successful.

Because `g_file_get_contents()` returns `FALSE` on failure, if you are only interested in whether it failed and don't need to display an error message, you can pass `NULL` for the `error` argument:

```
if (g_file_get_contents ("foo.txt", &contents, NULL, NULL)) /* ignore errors */
  /* no error occurred */ ;
else
  /* error */ ;
```

The GError object contains three fields: `domain` indicates the module the error-reporting function is located in, `code` indicates the specific error that occurred, and `message` is a user-readable error message with as many details as possible. Several functions are provided to deal with an error received from a called function: `g_error_matches()` returns `TRUE` if the error matches a given domain and code, `g_propagate_error()` copies an error into an error location (so the calling function will receive it), and `g_clear_error()` clears an error location by freeing the error and resetting the location to `NULL`. To display an error to the user, simply display `error->message`, perhaps along with additional context known only to the calling function (the file being opened, or whatever -- though in the `g_file_get_contents()` case, `error->message` already contains a filename).

When implementing a function that can report errors, the basic tool is `g_set_error()`. Typically, if a fatal error occurs you want to `g_set_error()`, then return immediately. `g_set_error()` does nothing if the error location passed to it is `NULL`. Here's an example:

```
gint
foo_open_file (GError **error)
{
  gint fd;

  fd = open ("file.txt", O_RDONLY);

  if (fd < 0)
    {
      g_set_error (error,
                   FOO_ERROR,                /* error domain */
                   FOO_ERROR_BLAH,           /* error code */
                   "Failed to open file: %s", /* error message format string */
                   g_strerror (errno));
      return -1;
    }
  else
    return fd;
}
```

Things are somewhat more complicated if you yourself call another function that can report a GError. If the sub-function indicates fatal errors in some way other than reporting a GError, such as by returning `TRUE` on success, you can simply do the following:

```
gboolean
my_function_that_can_fail (GError **err)
{
  g_return_val_if_fail (err == NULL || *err == NULL, FALSE);

  if (!sub_function_that_can_fail (err))
    {
       /* assert that error was set by the sub-function */
       g_assert (err == NULL || *err != NULL);
       return FALSE;
    }

  /* otherwise continue, no error occurred */
  g_assert (err == NULL || *err == NULL);
}
```

If the sub-function does not indicate errors other than by reporting a GError, you need to create a temporary GError since the passed-in one may be `NULL`. `g_propagate_error()` is intended for use in this case.

```
gboolean
my_function_that_can_fail (GError **err)
{
  GError *tmp_error;

  g_return_val_if_fail (err == NULL || *err == NULL, FALSE);

  tmp_error = NULL;
  sub_function_that_can_fail (&tmp_error);

  if (tmp_error != NULL)
    {
       /* store tmp_error in err, if err != NULL,
        * otherwise call g_error_free() on tmp_error
        */
       g_propagate_error (err, tmp_error);
       return FALSE;
    }

  /* otherwise continue, no error occurred */
}
```


Error pileups are always a bug. For example, this code is incorrect:

```
gboolean
my_function_that_can_fail (GError **err)
{
  GError *tmp_error;

  g_return_val_if_fail (err == NULL || *err == NULL, FALSE);

  tmp_error = NULL;
  sub_function_that_can_fail (&tmp_error);
  other_function_that_can_fail (&tmp_error);

  if (tmp_error != NULL)
    {
       g_propagate_error (err, tmp_error);
       return FALSE;
    }
}
```

`tmp_error` should be checked immediately after `sub_function_that_can_fail()`, and either cleared or propagated upward. The rule is: *after each error, you must either handle the error, or return it to the calling function*. Note that passing `NULL` for the error location is the equivalent of handling an error by always doing nothing about it. So the following code is fine, assuming errors in `sub_function_that_can_fail()` are not fatal to `my_function_that_can_fail()`:

```
gboolean
my_function_that_can_fail (GError **err)
{
  GError *tmp_error;

  g_return_val_if_fail (err == NULL || *err == NULL, FALSE);

  sub_function_that_can_fail (NULL); /* ignore errors */

  tmp_error = NULL;
  other_function_that_can_fail (&tmp_error);

  if (tmp_error != NULL)
    {
       g_propagate_error (err, tmp_error);
       return FALSE;
    }
}
```

Note that passing `NULL` for the error location *ignores* errors; it's equivalent to `try { sub_function_that_can_fail(); } catch (...) {}` in C++. It does *not* mean to leave errors unhandled; it means to handle them by doing nothing.

Error domains and codes are conventionally named as follows:

- The error domain is called `<NAMESPACE>_<MODULE>_ERROR`, for example `G_EXEC_ERROR` or `G_THREAD_ERROR`.
- The error codes are in an enumeration called `<Namespace>_<Module>_Error`; for example, GThreadError or GSpawnError.
- Members of the error code enumeration are called `<NAMESPACE>_<MODULE>_ERROR_<CODE>`, for example `G_SPAWN_ERROR_FORK` or `G_THREAD_ERROR_AGAIN`.
- If there's a "generic" or "unknown" error code for unrecoverable errors it doesn't make sense to distinguish with specific codes, it should be called `<NAMESPACE>_<MODULE>_ERROR_FAILED`, for example `G_SPAWN_ERROR_FAILED` or `G_THREAD_ERROR_FAILED`.

Summary of rules for use of GError:

- Do not report programming errors via GError.
- The last argument of a function that returns an error should be a location where a GError can be placed (i.e. "GError** error"). If GError is used with varargs, the GError** should be the last argument before the "...".
- The caller may pass `NULL` for the GError** if they are not interested in details of the exact error that occurred.
- If `NULL` is passed for the GError** argument, then errors should not be returned to the caller, but your function should still abort and return if an error occurs. That is, control flow should not be affected by whether the caller wants to get a GError.
- If a GError is reported, then your function by definition *had a fatal failure and did not complete whatever it was supposed to do*. If the failure was not fatal, then you handled it and you should not report it. If it was fatal, then you must report it and discontinue whatever you were doing immediately.
- A GError* must be initialized to `NULL` before passing its address to a function that can report errors.
- "Piling up" errors is always a bug. That is, if you assign a new GError to a GError* that is non-`NULL`, thus overwriting the previous error, it indicates that you should have aborted the operation instead of continuing. If you were able to continue, you should have cleared the previous error with `g_clear_error()`. `g_set_error()` will complain if you pile up errors.
- By convention, if you return a boolean value indicating success then `TRUE` means success and `FALSE` means failure. If `FALSE` is returned, the error *must* be set to a non-`NULL` value.
- A `NULL` return value is also frequently used to mean that an error occurred. You should make clear in your documentation whether `NULL` is a valid return value in non-error cases; if `NULL` is a valid value, then users must check whether an error was returned to see if the function succeeded.
- When implementing a function that can report errors, you may want to add a check at the top of your function that the error return location is either `NULL` or contains a `NULL` error (e.g. `g_return_if_fail (error == NULL || *error == NULL);`).

Details

GError

```
typedef struct {
  GQuark       domain;
  gint         code;
  gchar       *message;
} GError;
```

The GError structure contains information about an error that has occurred.

GQuark *domain*; error domain, e.g. G_FILE_ERROR.
gint *code*; error code, e.g. `G_FILE_ERROR_NOENT`.
gchar **message*; human-readable informative error message.

---

g_error_new ()

```
GError*     g_error_new                     (GQuark domain,
                                             gint code,
                                             const gchar *format,
                                             ...);
```

Creates a new GError with the given *domain* and *code*, and a message formatted with *format*.

domain : error domain
code : error code
format : `printf()`-style format for error message
... : parameters for message format
Returns : a new GError

---

g_error_new_literal ()

```
GError*     g_error_new_literal             (GQuark domain,
                                             gint code,
                                             const gchar *message);
```

Creates a new GError; unlike `g_error_new()`, *message* is not a `printf()`-style format string. Use this function if *message* contains text you don't have control over, that could include `printf()` escape sequences.

domain : error domain
code : error code
message : error message
Returns : a new GError

g_error_free ()

```
void        g_error_free                    (GError *error);
```

Frees a GError and associated resources.

error : a GError

g_error_copy ()

```
GError*     g_error_copy                    (const GError *error);
```

Makes a copy of *error*.

error : a GError
Returns : a new GError

g_error_matches ()

```
gboolean    g_error_matches                 (const GError *error,
                                             GQuark domain,
                                             gint code);
```

Returns TRUE if *error* matches *domain* and *code*, FALSE otherwise.

error : a GError
domain : an error domain
code : an error code
Returns : whether *error* has *domain* and *code*

g_set_error ()

```
void        g_set_error                     (GError **err,
                                             GQuark domain,
                                             gint code,
                                             const gchar *format,
                                             ...);
```

Does nothing if *err* is NULL; if *err* is non-NULL, then **err* must be NULL. A new GError is created and assigned to **err*.

err : a return location for a GError, or NULL

domain : error domain
code : error code
format : printf()-style format
... : args for *format*

g_propagate_error ()

```
void        g_propagate_error               (GError **dest,
                                             GError *src);
```

If *dest* is NULL, free *src*; otherwise, moves *src* into **dest*. **dest* must be NULL.

dest : error return location
src : error to move into the return location

g_clear_error ()

```
void        g_clear_error                   (GError **err);
```

If *err* is NULL, does nothing. If *err* is non-NULL, calls g_error_free() on **err* and sets **err* to NULL.

err : a GError return location

<< IO Channels **Message Output and Debugging Functions >>**

Message Output and Debugging Functions

Message Output and Debugging Functions — functions to output messages and help debug applications.

Synopsis

```
#include <glib.h>


void        g_print                         (const gchar *format,
                                             ...);
GPrintFunc  g_set_print_handler             (GPrintFunc func);
void        (*GPrintFunc)                   (const gchar *string);

void        g_printerr                      (const gchar *format,
                                             ...);
GPrintFunc  g_set_printerr_handler          (GPrintFunc func);

#define     g_return_if_fail                (expr)
#define     g_return_val_if_fail            (expr,val)
#define     g_return_if_reached             ()
#define     g_return_val_if_reached         (val)

#define     g_assert                        (expr)
#define     g_assert_not_reached            ()

void        g_on_error_query                (const gchar *prg_name);
void        g_on_error_stack_trace          (const gchar *prg_name);

#define     G_BREAKPOINT                    ()
```

Description

These functions provide support for outputting messages.

Details

g_print ()

```
void        g_print                         (const gchar *format,
                                             ...);
```

Outputs a formatted message via the print handler. The default print handler simply outputs the message to stdout.

g_print() should not be used from within libraries for debugging messages, since it may be redirected by applications to special purpose message windows or even files. Instead, libraries

should use g_log(), or the convenience functions g_message(), g_warning() and g_error().

format : the message format. See the printf() documentation.
... : the parameters to insert into the format string.

---

g_set_print_handler ()

```
GPrintFunc  g_set_print_handler             (GPrintFunc func);
```

Sets the print handler. Any messages passed to g_print() will be output via the new handler. The default handler simply outputs the message to stdout. By providing your own handler you can redirect the output, to a GTK+ widget or a log file for example.

func : the new print handler.
Returns : the old print handler.

GPrintFunc ()

```
void        (*GPrintFunc)                   (const gchar *string);
```

Specifies the type of the print handler functions. These are called with the complete formatted string to output.

string : the message to be output.

---

g_printerr ()

```
void        g_printerr                      (const gchar *format,
                                             ...);
```

Outputs a formatted message via the error message handler. The default handler simply outputs the message to stderr.

g_printerr() should not be used from within libraries. Instead g_log() should be used, or the convenience functions g_message(), g_warning() and g_error().

format : the message format. See the printf() documentation.
... : the parameters to insert into the format string.

---

g_set_printerr_handler ()

```
GPrintFunc  g_set_printerr_handler          (GPrintFunc func);
```

Sets the handler for printing error messages. Any messages passed to `g_printerr()` will be output via the new handler. The default handler simply outputs the message to stderr. By providing your own handler you can redirect the output, to a GTK+ widget or a log file for example.

func : the new error message handler.
Returns : the old error message handler.

---

g_return_if_fail()

```
#define     g_return_if_fail(expr)
```

Returns from the current function if the expression is not true. If the expression evaluates to FALSE, a critical message is logged and the function returns. This can only be used in functions which do not return a value.

expr : the expression to check.

---

g_return_val_if_fail()

```
#define     g_return_val_if_fail(expr,val)
```

Returns from the current function, returning the value *val*, if the expression is not true. If the expression evaluates to FALSE, a critical message is logged and *val* is returned.

expr : the expression to check.
val : the value to return from the current function if the expression is not true.

---

g_return_if_reached()

```
#define     g_return_if_reached()
```

Logs a critical message and returns from the current function. This can only be used in functions which do not return a value.

---

g_return_val_if_reached()

```
#define     g_return_val_if_reached(val)
```

Logs a critical message and returns *val*.

val : the value to return from the current function.

---

g_assert()

```
#define     g_assert(expr)
```

Debugging macro to terminate the application if the assertion fails. If the assertion fails (i.e. the expression is not true), an error message is logged and the application is terminated.

The macro can be turned off in final releases of code by defining G_DISABLE_ASSERT when compiling the application.

expr : the expression to check.

---

g_assert_not_reached()

```
#define     g_assert_not_reached()
```

Debugging macro to terminate the application if it is ever reached. If it is reached, an error message is logged and the application is terminated.

The macro can be turned off in final releases of code by defining G_DISABLE_ASSERT when compiling the application.

---

g_on_error_query ()

```
void        g_on_error_query                (const gchar *prg_name);
```

Prompts the user with `[E]xit, [H]alt, show [S]tack trace or [P]roceed`. This function is intended to be used for debugging use only. The following example shows how it can be used together with the `g_log()` functions.

```
#include <glib.h>

static void
log_handler (const gchar   *log_domain,
             GLogLevelFlags log_level,
             const gchar   *message,
             gpointer       user_data)
{
  g_log_default_handler (log_domain, log_level, message, user_data);

  g_on_error_query (MY_PROGRAM_NAME);
}

int main (int argc, char *argv[])
{
  g_log_set_handler (MY_LOG_DOMAIN,
                     G_LOG_LEVEL_WARNING |
                     G_LOG_LEVEL_ERROR |
```

```
                    G_LOG_LEVEL_CRITICAL,
                    log_handler,
                    NULL);

  /* ... */
```

If [E]xit is selected, the application terminates with a call to `_exit(0)`.

If [H]alt is selected, the application enters an infinite loop. The infinite loop can only be stopped by killing the application, or by setting glib_on_error_halt to `FALSE` (possibly via a debugger).

If [S]tack trace is selected, `g_on_error_stack_trace()` is called. This invokes **gdb**, which attaches to the current process and shows a stack trace. The prompt is then shown again.

If [P]roceed is selected, the function returns.

This function may cause different actions on non-UNIX platforms.

prg_name : the program name, needed by **gdb** for the [S]tack trace option. If *prg_name* is `NULL`, `g_get_prgname()` is called to get the program name (which will work correctly if `gdk_init()` or `gtk_init()` has been called).

---

g_on_error_stack_trace ()

```
void        g_on_error_stack_trace          (const gchar *prg_name);
```

Invokes **gdb**, which attaches to the current process and shows a stack trace. Called by `g_on_error_query()` when the [S]tack trace option is selected.

This function may cause different actions on non-UNIX platforms.

prg_name : the program name, needed by **gdb** for the [S]tack trace option. If *prg_name* is `NULL`, `g_get_prgname()` is called to get the program name (which will work correctly if `gdk_init()` or `gtk_init()` has been called).

G_BREAKPOINT()

```
#define     G_BREAKPOINT()
```

Inserts a breakpoint instruction into the code (on x86 machines only).

<< Error Reporting **Message Logging >>**

Message Logging

Message Logging — versatile support for logging messages with different levels of importance.

Synopsis

```
#include <glib.h>


#define     G_LOG_DOMAIN
#define     G_LOG_FATAL_MASK
#define     G_LOG_LEVEL_USER_SHIFT
void        (*GLogFunc)                     (const gchar *log_domain,
                                             GLogLevelFlags log_level,
                                             const gchar *message,
                                             gpointer user_data);
enum        GLogLevelFlags;

void        g_log                           (const gchar *log_domain,
                                             GLogLevelFlags log_level,
                                             const gchar *format,
                                             ...);
void        g_logv                          (const gchar *log_domain,
                                             GLogLevelFlags log_level,
                                             const gchar *format,
                                             va_list args);
#define     g_message                       (...)
#define     g_warning                       (...)
#define     g_critical                      (...)
#define     g_error                         (...)
#define     g_debug                         (...)

guint       g_log_set_handler               (const gchar *log_domain,
                                             GLogLevelFlags log_levels,
                                             GLogFunc log_func,
                                             gpointer user_data);
void        g_log_remove_handler            (const gchar *log_domain,
                                             guint handler_id);
GLogLevelFlags g_log_set_always_fatal       (GLogLevelFlags fatal_mask);
GLogLevelFlags g_log_set_fatal_mask         (const gchar *log_domain,
                                             GLogLevelFlags fatal_mask);
void        g_log_default_handler           (const gchar *log_domain,
                                             GLogLevelFlags log_level,
                                             const gchar *message,
                                             gpointer unused_data);
GLogFunc    g_log_set_default_handler       (GLogFunc log_func,
                                             gpointer user_data);
```

Description

These functions provide support for logging error messages or messages used for debugging.

There are several built-in levels of messages, defined in GLogLevelFlags. These can be extended

with user-defined levels.

Details

G_LOG_DOMAIN

```
#define G_LOG_DOMAIN    ((gchar*) 0)
```

Defines the log domain. For applications, this is typically left as the default `NULL` (or "") domain. Libraries should define this so that any messages which they log can be differentiated from messages from other libraries and application code. But be careful not to define it in any public header files.

For example, GTK+ uses this in its Makefile.am:

```
INCLUDES = -DG_LOG_DOMAIN=\"Gtk\"
```

G_LOG_FATAL_MASK

```
#define G_LOG_FATAL_MASK        (G_LOG_FLAG_RECURSION | G_LOG_LEVEL_ERROR)
```

GLib log levels that are considered fatal by default.

G_LOG_LEVEL_USER_SHIFT

```
#define G_LOG_LEVEL_USER_SHIFT  (8)
```

Log level shift offset for user defined log levels (0-7 are used by GLib).

GLogFunc ()

```
void        (*GLogFunc)                     (const gchar *log_domain,
                                             GLogLevelFlags log_level,
                                             const gchar *message,
                                             gpointer user_data);
```

Specifies the prototype of log handler functions.

log_domain : the log domain of the message.
log_level : the log level of the message (including the fatal and recursion flags).
message : the message to process.
user_data : user data, set in g_log_set_handler().

---

enum GLogLevelFlags

```
typedef enum
{
  /* log flags */
  G_LOG_FLAG_RECURSION          = 1 << 0,
  G_LOG_FLAG_FATAL              = 1 << 1,

  /* GLib log levels */
  G_LOG_LEVEL_ERROR             = 1 << 2,       /* always fatal */
  G_LOG_LEVEL_CRITICAL          = 1 << 3,
  G_LOG_LEVEL_WARNING           = 1 << 4,
  G_LOG_LEVEL_MESSAGE           = 1 << 5,
  G_LOG_LEVEL_INFO              = 1 << 6,
  G_LOG_LEVEL_DEBUG             = 1 << 7,

  G_LOG_LEVEL_MASK              = ~(G_LOG_FLAG_RECURSION | G_LOG_FLAG_FATAL)
} GLogLevelFlags;
```

Flags specifying the level of log messages.

---

g_log ()

```
void        g_log                           (const gchar *log_domain,
                                             GLogLevelFlags log_level,
                                             const gchar *format,
                                             ...);
```

Logs an error or debugging message. If the log level has been set as fatal, the `abort()` function is called to terminate the program.

log_domain : the log domain, usually G_LOG_DOMAIN.
log_level : the log level, either from GLogLevelFlags or a user-defined level.
format : the message format. See the `printf()` documentation.
... : the parameters to insert into the format string.

---

g_logv ()

```
void        g_logv                          (const gchar *log_domain,
                                             GLogLevelFlags log_level,
                                             const gchar *format,
                                             va_list args);
```

Logs an error or debugging message. If the log level has been set as fatal, the `abort()` function is called to terminate the program.

log_domain : the log domain.
log_level : the log level.
format : the message format. See the `printf()` documentation.

args : the parameters to insert into the format string.

---

g_message()

```
#define     g_message(...)
```

A convenience function/macro to log a normal message.

... : format string, followed by parameters to insert into the format string (as with `printf ()`)

---

g_warning()

```
#define     g_warning(...)
```

A convenience function/macro to log a warning message.

... : format string, followed by parameters to insert into the format string (as with `printf ()`)

---

g_critical()

```
#define     g_critical(...)
```

Logs a "critical warning" (G_LOG_LEVEL_CRITICAL). It's more or less application-defined what constitutes a critical vs. a regular warning. You could call `g_log_set_always_fatal()` to make critical warnings exit the program, then use `g_critical()` for fatal errors, for example.

... : format string, followed by parameters to insert into the format string (as with `printf ()`)

---

g_error()

```
#define     g_error(...)
```

A convenience function/macro to log an error message. Error messages are always fatal, resulting in a call to `abort()` to terminate the application. This function will result in a core dump; don't use it for errors you expect. Using this function indicates a bug in your program, i.e. an assertion failure.

... : the parameters to insert into the format string.

---

g_debug()

```
#define     g_debug(...)
```

A convenience function/macro to log a debug message.

... : format string, followed by parameters to insert into the format string (as with `printf ()`)

Since 2.6

---

g_log_set_handler ()

```
guint       g_log_set_handler               (const gchar *log_domain,
                                             GLogLevelFlags log_levels,
                                             GLogFunc log_func,
                                             gpointer user_data);
```

Sets the log handler for a domain and a set of log levels. To handle fatal and recursive messages the *log_levels* parameter must be combined with the G_LOG_FLAG_FATAL and G_LOG_FLAG_RECURSION bit flags.

Note that since the G_LOG_LEVEL_ERROR log level is always fatal, if you want to set a handler for this log level you must combine it with G_LOG_FLAG_FATAL.

Example 13. Adding a log handler for all warning messages in the default (application) domain

```
g_log_set_handler (NULL, G_LOG_LEVEL_WARNING | G_LOG_FLAG_FATAL
                   | G_LOG_FLAG_RECURSION, my_log_handler, NULL);
```

Example 14. Adding a log handler for all critical messages from GTK+

```
g_log_set_handler ("Gtk", G_LOG_LEVEL_CRITICAL | G_LOG_FLAG_FATAL
                   | G_LOG_FLAG_RECURSION, my_log_handler, NULL);
```

Example 15. Adding a log handler for *all* messages from GLib

```
g_log_set_handler ("GLib", G_LOG_LEVEL_MASK | G_LOG_FLAG_FATAL
                   | G_LOG_FLAG_RECURSION, my_log_handler, NULL);
```

log_domain : the log domain, or `NULL` for the default "" application domain.
log_levels : the log levels to apply the log handler for. To handle fatal and recursive messages as well, combine the log levels with the G_LOG_FLAG_FATAL and G_LOG_FLAG_RECURSION bit flags.
log_func : the log handler function.
user_data : data passed to the log handler.

Returns : the id of the new handler.

---

g_log_remove_handler ()

```
void        g_log_remove_handler            (const gchar *log_domain,
                                             guint handler_id);
```

Removes the log handler.

log_domain : the log domain.
handler_id : the id of the handler, which was returned in `g_log_set_handler()`.

---

g_log_set_always_fatal ()

```
GLogLevelFlags g_log_set_always_fatal       (GLogLevelFlags fatal_mask);
```

Sets the message levels which are always fatal, in any log domain. When a message with any of these levels is logged the program terminates. You can only set the levels defined by GLib to be fatal. `G_LOG_LEVEL_ERROR` is always fatal.

fatal_mask : the mask containing bits set for each level of error which is to be fatal.
Returns : the old fatal mask.

---

g_log_set_fatal_mask ()

```
GLogLevelFlags g_log_set_fatal_mask         (const gchar *log_domain,
                                             GLogLevelFlags fatal_mask);
```

Sets the log levels which are fatal in the given domain. `G_LOG_LEVEL_ERROR` is always fatal.

log_domain : the log domain.
fatal_mask : the new fatal mask.
Returns : the old fatal mask for the log domain.

---

g_log_default_handler ()

```
void        g_log_default_handler           (const gchar *log_domain,
                                             GLogLevelFlags log_level,
                                             const gchar *message,
                                             gpointer unused_data);
```

The default log handler set up by GLib; `g_log_set_default_handler()` allows to install an alternate default log handler. This is used if no log handler has been set for the particular log domain

and log level combination. It outputs the message to stderr or stdout and if the log level is fatal it calls `abort()`.

stderr is used for levels `G_LOG_LEVEL_ERROR`, `G_LOG_LEVEL_CRITICAL`, `G_LOG_LEVEL_WARNING` and `G_LOG_LEVEL_MESSAGE`. stdout is used for the rest.

log_domain : the log domain of the message.
log_level : the level of the message.
message : the message.
unused_data : data passed from `g_log()` which is unused.

---

g_log_set_default_handler ()

```
GLogFunc    g_log_set_default_handler       (GLogFunc log_func,
                                             gpointer user_data);
```

Installs a default log handler which is used is used if no log handler has been set for the particular log domain and log level combination. By default, GLib uses `g_log_default_handler()` as default log handler.

log_func : the log handler function.
user_data : data passed to the log handler.
Returns : the previous default log handler

Since 2.6

<< Message Output and Debugging Functions **GLib Utilities >>**

GLib Utilities

String Utility Functions - various string-related functions.
Character Set Conversion - convert strings between different character sets using iconv().
Unicode Manipulation - functions operating on Unicode characters and UTF-8 strings.
Internationalization - gettext support macros.
Date and Time Functions - calendrical calculations and miscellaneous time stuff.
Random Numbers - pseudo-random number generator.
Hook Functions - support for manipulating lists of hook functions.
Miscellaneous Utility Functions - a selection of portable utility functions.
Lexical Scanner - a general purpose lexical scanner.
Automatic String Completion - support for automatic completion using a group of target strings.
Timers - keep track of elapsed time.
Spawning Processes - process launching with fork()/exec().
File Utilities - various file-related functions.
Shell-related Utilities - shell-like commandline handling.
Commandline option parser - parses commandline options
Glob-style pattern matching - matches strings against patterns containing '*' (wildcard) and '?' (joker).
Simple XML Subset Parser - parses a subset of XML.
Key-value file parser - parses .ini-like config files
Windows Compatibility Functions - UNIX emulation on Windows.

<< Message Logging **String Utility Functions >>**

String Utility Functions

String Utility Functions — various string-related functions.

Synopsis

```
#include <glib.h>
#include <glib/gprintf.h>


gchar*      g_strdup                    (const gchar *str);
gchar*      g_strndup                   (const gchar *str,
                                         gsize n);
gchar**     g_strdupv                   (gchar **str_array);
gchar*      g_strnfill                  (gsize length,
                                         gchar fill_char);
gchar*      g_stpcpy                    (gchar *dest,
                                         const char *src);
gchar*      g_strstr_len                (const gchar *haystack,
                                         gssize haystack_len,
                                         const gchar *needle);
gchar*      g_strrstr                   (const gchar *haystack,
                                         const gchar *needle);
gchar*      g_strrstr_len               (const gchar *haystack,
                                         gssize haystack_len,
                                         const gchar *needle);
gboolean    g_str_has_prefix            (const gchar *str,
                                         const gchar *prefix);
gboolean    g_str_has_suffix            (const gchar *str,
                                         const gchar *suffix);

gsize       g_strlcpy                   (gchar *dest,
                                         const gchar *src,
                                         gsize dest_size);
gsize       g_strlcat                   (gchar *dest,
                                         const gchar *src,
                                         gsize dest_size);

gchar*      g_strdup_printf             (const gchar *format,
                                         ...);
gchar*      g_strdup_vprintf            (const gchar *format,
                                         va_list args);
gint        g_printf                    (gchar const *format,
                                         ...);
gint        g_vprintf                   (gchar const *format,
                                         va_list args);
gint        g_fprintf                   (FILE *file,
                                         gchar const *format,
                                         ...);
gint        g_vfprintf                  (FILE *file,
                                         gchar const *format,
                                         va_list args);
gint        g_sprintf                   (gchar *string,
                                         gchar const *format,
                                         ...);
```



```
gint        g_vsprintf                  (gchar *string,
                                         gchar const *format,
                                         va_list args);
gint        g_snprintf                  (gchar *string,
                                         gulong n,
                                         gchar const *format,
                                         ...);
gint        g_vsnprintf                 (gchar *string,
                                         gulong n,
                                         gchar const *format,
                                         va_list args);
gint        g_vasprintf                 (gchar **string,
                                         gchar const *format,
                                         va_list args);
gsize       g_printf_string_upper_bound (const gchar *format,
                                         va_list args);

gboolean    g_ascii_isalnum             (gchar c);
gboolean    g_ascii_isalpha             (gchar c);
gboolean    g_ascii_iscntrl             (gchar c);
gboolean    g_ascii_isdigit             (gchar c);
gboolean    g_ascii_isgraph             (gchar c);
gboolean    g_ascii_islower             (gchar c);
gboolean    g_ascii_isprint             (gchar c);
gboolean    g_ascii_ispunct             (gchar c);
gboolean    g_ascii_isspace             (gchar c);
gboolean    g_ascii_isupper             (gchar c);
gboolean    g_ascii_isxdigit            (gchar c);

gint        g_ascii_digit_value         (gchar c);
gint        g_ascii_xdigit_value        (gchar c);

gint        g_ascii_strcasecmp          (const gchar *s1,
                                         const gchar *s2);
gint        g_ascii_strncasecmp         (const gchar *s1,
                                         const gchar *s2,
                                         gsize n);

gchar*      g_ascii_strup               (const gchar *str,
                                         gssize len);
gchar*      g_ascii_strdown             (const gchar *str,
                                         gssize len);

gchar       g_ascii_tolower             (gchar c);
gchar       g_ascii_toupper             (gchar c);

GString*    g_string_ascii_up           (GString *string);
GString*    g_string_ascii_down         (GString *string);

gchar*      g_strup                     (gchar *string);
gchar*      g_strdown                   (gchar *string);

gint        g_strcasecmp                (const gchar *s1,
                                         const gchar *s2);
gint        g_strncasecmp               (const gchar *s1,
                                         const gchar *s2,
                                         guint n);

gchar*      g_strreverse                (gchar *string);

guint64     g_ascii_strtoull            (const gchar *nptr,
                                         gchar **endptr,
                                         guint base);
#define     G_ASCII_DTOSTR_BUF_SIZE
```

```
gdouble     g_ascii_strtod                  (const gchar *nptr,
                                             gchar **endptr);
gchar*      g_ascii_dtostr                  (gchar *buffer,
                                             gint buf_len,
                                             gdouble d);
gchar*      g_ascii_formatd                 (gchar *buffer,
                                             gint buf_len,
                                             const gchar *format,
                                             gdouble d);
gdouble     g_strtod                        (const gchar *nptr,
                                             gchar **endptr);

gchar*      g_strchug                       (gchar *string);
gchar*      g_strchomp                      (gchar *string);
#define     g_strstrip                      ( string )

gchar*      g_strdelimit                    (gchar *string,
                                             const gchar *delimiters,
                                             gchar new_delimiter);
#define     G_STR_DELIMITERS
gchar*      g_strescape                     (const gchar *source,
                                             const gchar *exceptions);
gchar*      g_strcompress                   (const gchar *source);
gchar*      g_strcanon                      (gchar *string,
                                             const gchar *valid_chars,
                                             gchar substitutor);
gchar**     g_strsplit                      (const gchar *string,
                                             const gchar *delimiter,
                                             gint max_tokens);
gchar**     g_strsplit_set                  (const gchar *string,
                                             const gchar *delimiters,
                                             gint max_tokens);
void        g_strfreev                      (gchar **str_array);
gchar*      g_strconcat                     (const gchar *string1,
                                             ...);
gchar*      g_strjoin                       (const gchar *separator,
                                             ...);
gchar*      g_strjoinv                      (const gchar *separator,
                                             gchar **str_array);
guint       g_strv_length                   (gchar **str_array);

G_CONST_RETURN gchar* g_strerror            (gint errnum);
G_CONST_RETURN gchar* g_strsignal           (gint signum);
```

Description

This section describes a number of utility functions for creating, duplicating, and manipulating strings.

Note that the functions g_printf(), g_fprintf(), g_sprintf(), g_snprintf(), g_vprintf(), g_vfprintf(), g_vsprintf() and g_vsnprintf() are declared in the header gprintf.h which is *not* included in glib.h (otherwise using glib.h would drag in stdio.h), so you'll have to explicitly include <glib/gprintf.h> in order to use the GLib printf() functions.

While you may use the printf() functions to format UTF-8 strings, notice that the precision of a %Ns parameter is interpreted as the number of *bytes*, not *characters* to print. On top of that, the GNU libc implementation of the printf() functions has the "feature" that it checks that the string given for the %Ns parameter consists of a whole number of characters in the current encoding. So, unless you are sure you are always going to be in an UTF-8 locale or your know your text is restricted to

ASCII, avoid using %Ns. If your intention is to format strings for a certain number of columns, then %Ns is not a correct solution anyway, since it fails to take wide characters (see g_unichar_iswide()) into account.

Details

g_strdup ()

```
gchar*      g_strdup                        (const gchar *str);
```

Duplicates a string. If *str* is NULL it returns NULL. The returned string should be freed when no longer needed.

str : the string to duplicate.
Returns : a newly-allocated copy of *str*.

---

g_strndup ()

```
gchar*      g_strndup                       (const gchar *str,
                                             gsize n);
```

Duplicates the first *n* characters of a string, returning a newly-allocated buffer *n* + 1 characters long which will always be nul-terminated. If *str* is less than *n* characters long the buffer is padded with nuls. If *str* is NULL it returns NULL. The returned value should be freed when no longer needed.

str : the string to duplicate part of.
n : the maximum number of characters to copy from *str*.
Returns : a newly-allocated buffer containing the first *n* characters of *str*, nul-terminated.

---

g_strdupv ()

```
gchar**     g_strdupv                       (gchar **str_array);
```

Copies NULL-terminated array of strings. The copy is a deep copy; the new array should be freed by first freeing each string, then the array itself. g_strfreev() does this for you. If called on a NULL value, g_strdupv() simply returns NULL.

str_array : NULL-terminated array of strings.
Returns : a new NULL-terminated array of strings.

---

g_strnfill ()

```
gchar*      g_strnfill                      (gsize length,
```

```
                                        gchar fill_char);
```

Creates a new string *length* characters long filled with *fill_char*. The returned string should be freed when no longer needed.

length : the length of the new string.
fill_char : the character to fill the string with.
Returns : a newly-allocated string filled the *fill_char*.

---

g_stpcpy ()

```
gchar*      g_stpcpy                    (gchar *dest,
                                         const char *src);
```

Copies a nul-terminated string into the dest buffer, include the trailing nul, and return a pointer to the trailing nul byte. This is useful for concatenating multiple strings together without having to repeatedly scan for the end.

dest : destination buffer.
src : source string.
Returns : a pointer to trailing nul byte.

---

g_strstr_len ()

```
gchar*      g_strstr_len                (const gchar *haystack,
                                         gssize haystack_len,
                                         const gchar *needle);
```

Searches the string *haystack* for the first occurrence of the string *needle*, limiting the length of the search to *haystack_len*.

haystack : a string.
haystack_len : the maximum length of *haystack*.
needle : the string to search for.
Returns : a pointer to the found occurrence, or `NULL` if not found.

---

g_strrstr ()

```
gchar*      g_strrstr                   (const gchar *haystack,
                                         const gchar *needle);
```

Searches the string *haystack* for the last occurrence of the string *needle*.

haystack : a nul-terminated string.

needle : the nul-terminated string to search for.
Returns : a pointer to the found occurrence, or `NULL` if not found.

---

g_strrstr_len ()

```
gchar*      g_strrstr_len               (const gchar *haystack,
                                         gssize haystack_len,
                                         const gchar *needle);
```

Searches the string *haystack* for the last occurrence of the string *needle*, limiting the length of the search to *haystack_len*.

haystack : a nul-terminated string.
haystack_len : the maximum length of *haystack*.
needle : the nul-terminated string to search for.
Returns : a pointer to the found occurrence, or `NULL` if not found.

---

g_str_has_prefix ()

```
gboolean    g_str_has_prefix            (const gchar *str,
                                         const gchar *prefix);
```

Looks whether the string *str* begins with *prefix*.

str : a nul-terminated string.
prefix : the nul-terminated prefix to look for.
Returns : `TRUE` if *str* begins with *prefix*, `FALSE` otherwise.

Since 2.2

---

g_str_has_suffix ()

```
gboolean    g_str_has_suffix            (const gchar *str,
                                         const gchar *suffix);
```

Looks whether the string *str* ends with *suffix*.

str : a nul-terminated string.
suffix : the nul-terminated suffix to look for.
Returns : `TRUE` if *str* end with *suffix*, `FALSE` otherwise.

Since 2.2

g_strlcpy ()

```
gsize       g_strlcpy                       (gchar *dest,
                                             const gchar *src,
                                             gsize dest_size);
```

Portability wrapper that calls `strlcpy()` on systems which have it, and emulates `strlcpy()` otherwise. Copies *`src`* to *`dest`*; *`dest`* is guaranteed to be nul-terminated; *`src`* must be nul-terminated; *`dest_size`* is the buffer size, not the number of chars to copy. Caveat: `strlcpy()` is supposedly more secure than `strcpy()` or `strncpy()`, but if you really want to avoid screwups, `g_strdup()` is an even better idea.

`dest` : destination buffer
`src` : source buffer
`dest_size` : length of *`dest`* in bytes
Returns : length of *`src`*

---

g_strlcat ()

```
gsize       g_strlcat                       (gchar *dest,
                                             const gchar *src,
                                             gsize dest_size);
```

Portability wrapper that calls `strlcat()` on systems which have it, and emulates it otherwise. Appends nul-terminated *`src`* string to *`dest`*, guaranteeing nul-termination for *`dest`*. The total size of *`dest`* won't exceed *`dest_size`*. Caveat: this is supposedly a more secure alternative to `strcat()` or `strncat()`, but for real security `g_strconcat()` is harder to mess up.

`dest` : destination buffer, already containing one nul-terminated string
`src` : source buffer
`dest_size` : length of *`dest`* buffer in bytes (not length of existing string inside *`dest`*)
Returns : length of *`src`* plus initial length of string in *`dest`*

---

g_strdup_printf ()

```
gchar*      g_strdup_printf                 (const gchar *format,
                                             ...);
```

Similar to the standard C `sprintf()` function but safer, since it calculates the maximum space required and allocates memory to hold the result. The returned string should be freed when no longer needed.

`format` : a standard `printf()` format string, but notice string precision pitfalls.
`...` : the parameters to insert into the format string.

Returns : a newly-allocated string holding the result.

---

g_strdup_vprintf ()

```
gchar*      g_strdup_vprintf                (const gchar *format,
                                             va_list args);
```

Similar to the standard C `vsprintf()` function but safer, since it calculates the maximum space required and allocates memory to hold the result. The returned string should be freed when no longer needed.

See also `g_vasprintf()`, which offers the same functionality, but additionally returns the length of the allocated string.

`format` : a standard `printf()` format string, but notice string precision pitfalls.
`args` : the list of parameters to insert into the format string.
Returns : a newly-allocated string holding the result.

---

g_printf ()

```
gint        g_printf                        (gchar const *format,
                                             ...);
```

An implementation of the standard `printf()` function which supports positional parameters, as specified in the Single Unix Specification.

`format` : a standard `printf()` format string, but notice string precision pitfalls.
`...` : the arguments to insert in the output.
Returns : the number of characters printed.

Since 2.2

---

g_vprintf ()

```
gint        g_vprintf                       (gchar const *format,
                                             va_list args);
```

An implementation of the standard `vprintf()` function which supports positional parameters, as specified in the Single Unix Specification.

`format` : a standard `printf()` format string, but notice string precision pitfalls.
`args` : the list of arguments to insert in the output.
Returns : the number of characters printed.

Since 2.2

---

g_fprintf ()

```
gint        g_fprintf                       (FILE *file,
                                             gchar const *format,
                                             ...);
```

An implementation of the standard `fprintf()` function which supports positional parameters, as specified in the Single Unix Specification.

file : the stream to write to.
format : a standard `printf()` format string, but notice string precision pitfalls.
`...` : the arguments to insert in the output.
Returns : the number of characters printed.

Since 2.2

---

g_vfprintf ()

```
gint        g_vfprintf                      (FILE *file,
                                             gchar const *format,
                                             va_list args);
```

An implementation of the standard `fprintf()` function which supports positional parameters, as specified in the Single Unix Specification.

file : the stream to write to.
format : a standard `printf()` format string, but notice string precision pitfalls.
args : the list of arguments to insert in the output.
Returns : the number of characters printed.

Since 2.2

---

g_sprintf ()

```
gint        g_sprintf                       (gchar *string,
                                             gchar const *format,
                                             ...);
```

An implementation of the standard `sprintf()` function which supports positional parameters, as specified in the Single Unix Specification.

string : the buffer to hold the output.
format : a standard `printf()` format string, but notice string precision pitfalls.
`...` : the arguments to insert in the output.
Returns : the number of characters printed.

Since 2.2

---

g_vsprintf ()

```
gint        g_vsprintf                      (gchar *string,
                                             gchar const *format,
                                             va_list args);
```

An implementation of the standard `vsprintf()` function which supports positional parameters, as specified in the Single Unix Specification.

string : the buffer to hold the output.
format : a standard `printf()` format string, but notice string precision pitfalls.
args : the list of arguments to insert in the output.
Returns : the number of characters printed.

Since 2.2

---

g_snprintf ()

```
gint        g_snprintf                      (gchar *string,
                                             gulong n,
                                             gchar const *format,
                                             ...);
```

A safer form of the standard `sprintf()` function. The output is guaranteed to not exceed *n* characters (including the terminating nul character), so it is easy to ensure that a buffer overflow cannot occur.

See also `g_strdup_printf()`.

In versions of GLib prior to 1.2.3, this function may return -1 if the output was truncated, and the truncated string may not be nul-terminated. In versions prior to 1.3.12, this function returns the length of the output string.

The return value of `g_snprintf()` conforms to the `snprintf()` function as standardized in ISO C99. Note that this is different from traditional `snprintf()`, which returns the length of the output string.

The format string may contain positional parameters, as specified in the Single Unix Specification.

string : the buffer to hold the output.
n : the maximum number of characters to produce (including the terminating nul character).
format : a standard `printf()` format string, but notice string precision pitfalls.
... : the arguments to insert in the output.
Returns : the number of characters which would be produced if the buffer was large enough.

---

g_vsnprintf ()

```
gint        g_vsnprintf                     (gchar *string,
                                             gulong n,
                                             gchar const *format,
                                             va_list args);
```

A safer form of the standard `vsprintf()` function. The output is guaranteed to not exceed *n* characters (including the terminating nul character), so it is easy to ensure that a buffer overflow cannot occur.

See also `g_strdup_vprintf()`.

In versions of GLib prior to 1.2.3, this function may return -1 if the output was truncated, and the truncated string may not be nul-terminated. In versions prior to 1.3.12, this function returns the length of the output string.

The return value of `g_vsnprintf()` conforms to the `vsnprintf()` function as standardized in ISO C99. Note that this is different from traditional `vsnprintf()`, which returns the length of the output string.

The format string may contain positional parameters, as specified in the Single Unix Specification.

string : the buffer to hold the output.
n : the maximum number of characters to produce (including the terminating nul character).
format : a standard `printf()` format string, but notice string precision pitfalls.
args : the list of arguments to insert in the output.
Returns : the number of characters which would be produced if the buffer was large enough.

---

g_vasprintf ()

```
gint        g_vasprintf                     (gchar **string,
                                             gchar const *format,
                                             va_list args);
```

An implementation of the GNU `vasprintf()` function which supports positional parameters, as specified in the Single Unix Specification. This function is similar to `g_vsprintf()`, except that it

allocates a string to hold the output, instead of putting the output in a buffer you allocate in advance.

string : the return location for the newly-allocated string.
format : a standard `printf()` format string, but notice string precision pitfalls.
args : the list of arguments to insert in the output.
Returns : the number of characters printed.

Since 2.4

---

g_printf_string_upper_bound ()

```
gsize       g_printf_string_upper_bound     (const gchar *format,
                                             va_list args);
```

Calculates the maximum space needed to store the output of the `sprintf()` function.

format : the format string. See the `printf()` documentation.
args : the parameters to be inserted into the format string.
Returns : the maximum space needed to store the formatted string.

---

g_ascii_isalnum ()

```
gboolean    g_ascii_isalnum                 (gchar c);
```

Determines whether a character is alphanumeric.

Unlike the standard C library `isalnum()` function, this only recognizes standard ASCII letters and ignores the locale, returning `FALSE` for all non-ASCII characters. Also unlike the standard library function, this takes a char, not an int, so don't call it on `EOF` but no need to cast to guchar before passing a possibly non-ASCII character in.

c : any character
Returns : `TRUE` if *c* is an ASCII alphanumeric character

---

g_ascii_isalpha ()

```
gboolean    g_ascii_isalpha                 (gchar c);
```

Determines whether a character is alphabetic (i.e. a letter).

Unlike the standard C library `isalpha()` function, this only recognizes standard ASCII letters and ignores the locale, returning `FALSE` for all non-ASCII characters. Also unlike the standard library function, this takes a char, not an int, so don't call it on `EOF` but no need to cast to guchar before

passing a possibly non-ASCII character in.

c : any character
Returns : TRUE if *c* is an ASCII alphabetic character

---

g_ascii_iscntrl ()

```
gboolean    g_ascii_iscntrl                 (gchar c);
```

Determines whether a character is a control character.

Unlike the standard C library `iscntrl()` function, this only recognizes standard ASCII control characters and ignores the locale, returning FALSE for all non-ASCII characters. Also unlike the standard library function, this takes a char, not an int, so don't call it on EOF but no need to cast to guchar before passing a possibly non-ASCII character in.

c : any character
Returns : TRUE if *c* is an ASCII control character.

---

g_ascii_isdigit ()

```
gboolean    g_ascii_isdigit                 (gchar c);
```

Determines whether a character is digit (0-9).

Unlike the standard C library `isdigit()` function, this takes a char, not an int, so don't call it on EOF but no need to cast to guchar before passing a possibly non-ASCII character in.

c : any character
Returns : TRUE if *c* is an ASCII digit.

---

g_ascii_isgraph ()

```
gboolean    g_ascii_isgraph                 (gchar c);
```

Determines whether a character is a printing character and not a space.

Unlike the standard C library `isgraph()` function, this only recognizes standard ASCII characters and ignores the locale, returning FALSE for all non-ASCII characters. Also unlike the standard library function, this takes a char, not an int, so don't call it on EOF but no need to cast to guchar before passing a possibly non-ASCII character in.

c : any character
Returns : TRUE if *c* is an ASCII printing character other than space.

---

g_ascii_islower ()

```
gboolean    g_ascii_islower                 (gchar c);
```

Determines whether a character is an ASCII lower case letter.

Unlike the standard C library `islower()` function, this only recognizes standard ASCII letters and ignores the locale, returning FALSE for all non-ASCII characters. Also unlike the standard library function, this takes a char, not an int, so don't call it on EOF but no need to worry about casting to guchar before passing a possibly non-ASCII character in.

c : any character
Returns : TRUE if *c* is an ASCII lower case letter

---

g_ascii_isprint ()

```
gboolean    g_ascii_isprint                 (gchar c);
```

Determines whether a character is a printing character.

Unlike the standard C library `isprint()` function, this only recognizes standard ASCII characters and ignores the locale, returning FALSE for all non-ASCII characters. Also unlike the standard library function, this takes a char, not an int, so don't call it on EOF but no need to cast to guchar before passing a possibly non-ASCII character in.

c : any character
Returns : TRUE if *c* is an ASCII printing character.

---

g_ascii_ispunct ()

```
gboolean    g_ascii_ispunct                 (gchar c);
```

Determines whether a character is a punctuation character.

Unlike the standard C library `ispunct()` function, this only recognizes standard ASCII letters and ignores the locale, returning FALSE for all non-ASCII characters. Also unlike the standard library function, this takes a char, not an int, so don't call it on EOF but no need to cast to guchar before passing a possibly non-ASCII character in.

c : any character
Returns : TRUE if *c* is an ASCII punctuation character.

---

g_ascii_isspace ()

```
gboolean    g_ascii_isspace                 (gchar c);
```

Determines whether a character is a white-space character.

Unlike the standard C library `isspace()` function, this only recognizes standard ASCII white-space and ignores the locale, returning `FALSE` for all non-ASCII characters. Also unlike the standard library function, this takes a char, not an int, so don't call it on `EOF` but no need to cast to guchar before passing a possibly non-ASCII character in.

c : any character
Returns : `TRUE` if *c* is an ASCII white-space character

---

g_ascii_isupper ()

```
gboolean    g_ascii_isupper                 (gchar c);
```

Determines whether a character is an ASCII upper case letter.

Unlike the standard C library `isupper()` function, this only recognizes standard ASCII letters and ignores the locale, returning `FALSE` for all non-ASCII characters. Also unlike the standard library function, this takes a char, not an int, so don't call it on `EOF` but no need to worry about casting to guchar before passing a possibly non-ASCII character in.

c : any character
Returns : `TRUE` if *c* is an ASCII upper case letter

---

g_ascii_isxdigit ()

```
gboolean    g_ascii_isxdigit                (gchar c);
```

Determines whether a character is a hexadecimal-digit character.

Unlike the standard C library `isxdigit()` function, this takes a char, not an int, so don't call it on `EOF` but no need to cast to guchar before passing a possibly non-ASCII character in.

c : any character
Returns : `TRUE` if *c* is an ASCII hexadecimal-digit character.

---

g_ascii_digit_value ()

```
gint        g_ascii_digit_value             (gchar c);
```

Determines the numeric value of a character as a decimal digit. Differs from `g_unichar_digit_value()` because it takes a char, so there's no worry about sign extension if

characters are signed.

c : an ASCII character.
Returns : If *c* is a decimal digit (according to `g_ascii_isdigit()`), its numeric value. Otherwise, -1.

---

g_ascii_xdigit_value ()

```
gint        g_ascii_xdigit_value            (gchar c);
```

Determines the numeric value of a character as a hexidecimal digit. Differs from `g_unichar_xdigit_value()` because it takes a char, so there's no worry about sign extension if characters are signed.

c : an ASCII character.
Returns : If *c* is a hex digit (according to `g_ascii_isxdigit()`), its numeric value. Otherwise, -1.

---

g_ascii_strcasecmp ()

```
gint        g_ascii_strcasecmp              (const gchar *s1,
                                             const gchar *s2);
```

Compare two strings, ignoring the case of ASCII characters.

Unlike the BSD `strcasecmp()` function, this only recognizes standard ASCII letters and ignores the locale, treating all non-ASCII characters as if they are not letters.

s1 : string to compare with *s2*.
s2 : string to compare with *s1*.
Returns : an integer less than, equal to, or greater than zero if *s1* is found, respectively, to be less than, to match, or to be greater than *s2*.

---

g_ascii_strncasecmp ()

```
gint        g_ascii_strncasecmp             (const gchar *s1,
                                             const gchar *s2,
                                             gsize n);
```

Compare *s1* and *s2*, ignoring the case of ASCII characters and any characters after the first *n* in each string.

Unlike the BSD `strcasecmp()` function, this only recognizes standard ASCII letters and ignores the locale, treating all non-ASCII characters as if they are not letters.

s1 : string to compare with *s2*.

s2 : string to compare with *s1*.

n : number of characters to compare.

Returns : an integer less than, equal to, or greater than zero if the first *n* bytes of *s1* is found, respectively, to be less than, to match, or to be greater than the first *n* bytes of *s2*.

---

g_ascii_strup ()

```
gchar*      g_ascii_strup                   (const gchar *str,
                                             gssize len);
```

Converts all lower case ASCII letters to upper case ASCII letters.

str : a string.

len : length of *str* in bytes, or -1 if *str* is nul-terminated.

Returns : a newly allocated string, with all the lower case characters in *str* converted to upper case, with semantics that exactly match g_ascii_toupper(). (Note that this is unlike the old g_strup(), which modified the string in place.)

---

g_ascii_strdown ()

```
gchar*      g_ascii_strdown                 (const gchar *str,
                                             gssize len);
```

Converts all upper case ASCII letters to lower case ASCII letters.

str : a string.

len : length of *str* in bytes, or -1 if *str* is nul-terminated.

Returns : a newly-allocated string, with all the upper case characters in *str* converted to lower case, with semantics that exactly match g_ascii_tolower(). (Note that this is unlike the old g_strdown(), which modified the string in place.)

---

g_ascii_tolower ()

```
gchar       g_ascii_tolower                 (gchar c);
```

Convert a character to ASCII lower case.

Unlike the standard C library tolower() function, this only recognizes standard ASCII letters and ignores the locale, returning all non-ASCII characters unchanged, even if they are lower case letters in a particular character set. Also unlike the standard library function, this takes and returns a char, not an int, so don't call it on EOF but no need to worry about casting to guchar before passing a possibly non-ASCII character in.

c : any character.

Returns : the result of converting *c* to lower case. If *c* is not an ASCII upper case letter, *c* is returned unchanged.

---

g_ascii_toupper ()

```
gchar       g_ascii_toupper                 (gchar c);
```

Convert a character to ASCII upper case.

Unlike the standard C library toupper() function, this only recognizes standard ASCII letters and ignores the locale, returning all non-ASCII characters unchanged, even if they are upper case letters in a particular character set. Also unlike the standard library function, this takes and returns a char, not an int, so don't call it on EOF but no need to worry about casting to guchar before passing a possibly non-ASCII character in.

c : any character.

Returns : the result of converting *c* to upper case. If *c* is not an ASCII lower case letter, *c* is returned unchanged.

---

g_string_ascii_up ()

```
GString*    g_string_ascii_up               (GString *string);
```

Converts all lower case ASCII letters to upper case ASCII letters.

string : a GString

Returns : passed-in *string* pointer, with all the lower case characters converted to upper case in place, with semantics that exactly match g_ascii_toupper.

---

g_string_ascii_down ()

```
GString*    g_string_ascii_down             (GString *string);
```

Converts all upper case ASCII letters to lower case ASCII letters.

string : a GString

Returns : passed-in *string* pointer, with all the upper case characters converted to lower case in place, with semantics that exactly match g_ascii_tolower.

---

g_strup ()

```
gchar*      g_strup                         (gchar *string);
```

Warning

g_strup is deprecated and should not be used in newly-written code. This function is totally broken for the reasons discussed in the g_strncasecmp() docs - use g_ascii_strup() or g_utf8_strup() instead.

Converts a string to upper case.

string : the string to convert.
Returns : the string

---

g_strdown ()

```
gchar*      g_strdown                       (gchar *string);
```

Warning

g_strdown is deprecated and should not be used in newly-written code. This function is totally broken for the reasons discussed in the g_strncasecmp() docs - use g_ascii_strdown() or g_utf8_strdown() instead.

Converts a string to lower case.

string : the string to convert.
Returns : the string

---

g_strcasecmp ()

```
gint        g_strcasecmp                    (const gchar *s1,
                                             const gchar *s2);
```

Warning

g_strcasecmp is deprecated and should not be used in newly-written code. See g_strncasecmp() for a discussion of why this function is deprecated and how to replace it.

A case-insensitive string comparison, corresponding to the standard strcasecmp() function on platforms which support it.

s1 : a string.
s2 : a string to compare with *s1*.
Returns : 0 if the strings match, a negative value if *s1* < *s2*, or a positive value if *s1* > *s2*.

---

g_strncasecmp ()

```
gint        g_strncasecmp                   (const gchar *s1,
                                             const gchar *s2,
                                             guint n);
```

Warning

g_strncasecmp is deprecated and should not be used in newly-written code. The problem with g_strncasecmp() is that it does the comparison by calling toupper()/tolower(). These functions are locale-specific and operate on single bytes. However, it is impossible to handle things correctly from an I18N standpoint by operating on bytes, since characters may be multibyte. Thus g_strncasecmp() is broken if your string is guaranteed to be ASCII, since it's locale-sensitive, and it's broken if your string is localized, since it doesn't work on many encodings at all, including UTF-8, EUC-JP, etc.

There are therefore two replacement functions: g_ascii_strncasecmp(), which only works on ASCII and is not locale-sensitive, and g_utf8_casefold(), which is good for case-insensitive sorting of UTF-8.

A case-insensitive string comparison, corresponding to the standard strncasecmp() function on platforms which support it. It is similar to g_strcasecmp() except it only compares the first *n* characters of the strings.

s1 : a string.
s2 : a string to compare with *s1*.
n : the maximum number of characters to compare.
Returns : 0 if the strings match, a negative value if *s1* < *s2*, or a positive value if *s1* > *s2*.

---

g_strreverse ()

```
gchar*      g_strreverse                    (gchar *string);
```

Reverses all of the bytes in a string. For example, g_strreverse ("abcdef") will result in "fedcba".

Note that g_strreverse() doesn't work on UTF-8 strings containing multibyte characters. For that purpose, use g_utf8_strreverse().

string : the string to reverse.
Returns : the same pointer passed in as *string*.

---

g_ascii_strtoull ()

```
guint64     g_ascii_strtoull                (const gchar *nptr,
                                             gchar **endptr,
                                             guint base);
```

Converts a string to a guint64 value. This function behaves like the standard `strtoull()` function does in the C locale. It does this without actually changing the current locale, since that would not be thread-safe.

This function is typically used when reading configuration files or other non-user input that should be locale independent. To handle input from the user you should normally use the locale-sensitive system `strtoull()` function.

If the correct value would cause overflow, `G_MAXUINT64` is returned, and `ERANGE` is stored in `errno`.

nptr : the string to convert to a numeric value.
endptr : if non-`NULL`, it returns the character after the last character used in the conversion.
base : to be used for the conversion, 2..36 or 0
Returns : the guint64 value.

Since 2.2

G_ASCII_DTOSTR_BUF_SIZE

```
#define G_ASCII_DTOSTR_BUF_SIZE (29 + 10)
```

A good size for a buffer to be passed into `g_ascii_dtostr()`. It is guaranteed to be enough for all output of that function on systems with 64bit IEEE-compatible doubles.

The typical usage would be something like:

```
  char buf[G_ASCII_DTOSTR_BUF_SIZE];

  fprintf (out, "value=%s\n", g_ascii_dtostr (buf, sizeof (buf), value));
```

---

g_ascii_strtod ()

```
gdouble     g_ascii_strtod                  (const gchar *nptr,
                                             gchar **endptr);
```

Converts a string to a gdouble value. This function behaves like the standard `strtod()` function does in the C locale. It does this without actually changing the current locale, since that would not be thread-safe.

This function is typically used when reading configuration files or other non-user input that should be locale independent. To handle input from the user you should normally use the locale-sensitive

system `strtod()` function.

To convert from a gdouble to a string in a locale-insensitive way, use `g_ascii_dtostr()`.

If the correct value would cause overflow, plus or minus `HUGE_VAL` is returned (according to the sign of the value), and `ERANGE` is stored in `errno`. If the correct value would cause underflow, zero is returned and `ERANGE` is stored in `errno`.

This function resets `errno` before calling `strtod()` so that you can reliably detect overflow and underflow.

nptr : the string to convert to a numeric value.
endptr : if non-`NULL`, it returns the character after the last character used in the conversion.
Returns : the gdouble value.

---

g_ascii_dtostr ()

```
gchar*      g_ascii_dtostr                  (gchar *buffer,
                                             gint buf_len,
                                             gdouble d);
```

Converts a gdouble to a string, using the '.' as decimal point.

This functions generates enough precision that converting the string back using `g_ascii_strtod()` gives the same machine-number (on machines with IEEE compatible 64bit doubles). It is guaranteed that the size of the resulting string will never be larger than *`G_ASCII_DTOSTR_BUF_SIZE`* bytes.

buffer : A buffer to place the resulting string in
buf_len : The length of the buffer.
d : The gdouble to convert
Returns : The pointer to the buffer with the converted string.

---

g_ascii_formatd ()

```
gchar*      g_ascii_formatd                 (gchar *buffer,
                                             gint buf_len,
                                             const gchar *format,
                                             gdouble d);
```

Converts a gdouble to a string, using the '.' as decimal point. To format the number you pass in a `printf()`-style format string. Allowed conversion specifiers are 'e', 'E', 'f', 'F', 'g' and 'G'.

If you just want to want to serialize the value into a string, use `g_ascii_dtostr()`.

buffer : A buffer to place the resulting string in
buf_len : The length of the buffer.
format : The `printf()`-style format to use for the code to use for converting.

d : The gdouble to convert
Returns : The pointer to the buffer with the converted string.

---

g_strtod ()

```
gdouble     g_strtod                        (const gchar *nptr,
                                             gchar **endptr);
```

Converts a string to a gdouble value. It calls the standard `strtod()` function to handle the conversion, but if the string is not completely converted it attempts the conversion again with `g_ascii_strtod()`, and returns the best match.

This function should seldomly be used. The normal situation when reading numbers not for human consumption is to use `g_ascii_strtod()`. Only when you know that you must expect both locale formatted and C formatted numbers should you use this. Make sure that you don't pass strings such as comma separated lists of values, since the commas may be interpreted as a decimal point in some locales, causing unexpected results.

nptr : the string to convert to a numeric value.
endptr : if non-`NULL`, it returns the character after the last character used in the conversion.
Returns : the gdouble value.

---

g_strchug ()

```
gchar*      g_strchug                       (gchar *string);
```

Removes leading whitespace from a string, by moving the rest of the characters forward.

string : a string to remove the leading whitespace from.
Returns : *string*.

---

g_strchomp ()

```
gchar*      g_strchomp                      (gchar *string);
```

Removes trailing whitespace from a string.

string : a string to remove the trailing whitespace from.
Returns : *string*.

---

g_strstrip()


```
#define     g_strstrip( string )
```

Removes leading and trailing whitespace from a string.

string : a string to remove the leading and trailing whitespace from.

---

g_strdelimit ()

```
gchar*      g_strdelimit                    (gchar *string,
                                             const gchar *delimiters,
                                             gchar new_delimiter);
```

Converts any delimiter characters in *string* to *new_delimiter*. Any characters in *string* which are found in *delimiters* are changed to the *new_delimiter* character. Modifies *string* in place, and returns *string* itself, not a copy. The return value is to allow nesting such as `g_ascii_strup (g_strdelimit (str, "abc", '?'))`.

string : the string to convert.
delimiters : a string containing the current delimiters, or `NULL` to use the standard delimiters defined in G_STR_DELIMITERS.
new_delimiter : the new delimiter character.
Returns : *string*.

G_STR_DELIMITERS

```
#define  G_STR_DELIMITERS       "_-|> <."
```

The standard delimiters, used in `g_strdelimit()`.

---

g_strescape ()

```
gchar*      g_strescape                     (const gchar *source,
                                             const gchar *exceptions);
```

Escapes the special characters '\b', '\f', '\n', '\r', '\t', '\' and '"' in the string *source* by inserting a '\' before them. Additionally all characters in the range 0x01-0x1F (everything below SPACE) and in the range 0x7F-0xFF (all non-ASCII chars) are replaced with a '\' followed by their octal representation. Characters supplied in *exceptions* are not escaped.

`g_strcompress()` does the reverse conversion.

source : a string to escape.
exceptions : a string of characters not to escape in *source*.

Returns : a newly-allocated copy of *source* with certain characters escaped. See above.

---

g_strcompress ()

```
gchar*      g_strcompress                   (const gchar *source);
```

Replaces all escaped characters with their one byte equivalent. It does the reverse conversion of g_strescape().

source : a string to compress.

Returns : a newly-allocated copy of *source* with all escaped character compressed.

---

g_strcanon ()

```
gchar*      g_strcanon                      (gchar *string,
                                             const gchar *valid_chars,
                                             gchar substitutor);
```

For each character in *string*, if the character is not in *valid_chars*, replaces the character with *substitutor*. Modifies *string* in place, and return *string* itself, not a copy. The return value is to allow nesting such as `g_ascii_strup (g_strcanon (str, "abc", '?'))`.

string : a nul-terminated array of bytes.

valid_chars : bytes permitted in *string*.

substitutor : replacement character for disallowed bytes.

Returns : *string*.

---

g_strsplit ()

```
gchar**     g_strsplit                      (const gchar *string,
                                             const gchar *delimiter,
                                             gint max_tokens);
```

Splits a string into a maximum of *max_tokens* pieces, using the given *delimiter*. If *max_tokens* is reached, the remainder of *string* is appended to the last token.

As a special case, the result of splitting the empty string "" is an empty vector, not a vector containing a single string. The reason for this special case is that being able to represent a empty vector is typically more useful than consistent handling of empty elements. If you do need to represent empty elements, you'll need to check for the empty string before calling g_strsplit().

string : a string to split.

delimiter : a string which specifies the places at which to split the string. The delimiter is not included in any of the resulting strings, unless *max_tokens* is reached.

max_tokens : the maximum number of pieces to split *string* into. If this is less than 1, the string is split completely.

Returns : a newly-allocated NULL-terminated array of strings. Use g_strfreev() to free it.

---

g_strsplit_set ()

```
gchar**     g_strsplit_set                  (const gchar *string,
                                             const gchar *delimiters,
                                             gint max_tokens);
```

Splits *string* into a number of tokens not containing any of the characters in *delimiter*. A token is the (possibly empty) longest string that does not contain any of the characters in *delimiters*. If *max_tokens* is reached, the remainder is appended to the last token.

For example the result of g_strsplit_set ("abc:def/ghi", ":/", -1) is a NULL-terminated vector containing the three strings "abc", "def", and "ghi".

The result if g_strsplit_set (":def/ghi:", ":/", -1) is a NULL-terminated vector containing the four strings "", "def", "ghi", and "".

As a special case, the result of splitting the empty string "" is an empty vector, not a vector containing a single string. The reason for this special case is that being able to represent a empty vector is typically more useful than consistent handling of empty elements. If you do need to represent empty elements, you'll need to check for the empty string before calling g_strsplit_set ().

Note that this function works on bytes not characters, so it can't be used to delimit UTF-8 strings for anything but ASCII characters.

string : The string to be tokenized

delimiters : A nul-terminated string containing bytes that are used to split the string.

max_tokens : The maximum number of tokens to split *string* into. If this is less than 1, the string is split completely

Returns : a newly-allocated NULL-terminated array of strings. Use g_strfreev() to free it.

Since 2.4

---

g_strfreev ()

```
void        g_strfreev                      (gchar **str_array);
```

Frees a NULL-terminated array of strings, and the array itself. If called on a NULL value, g_strfreev () simply returns.

str_array : a NULL-terminated array of strings to free.

---

g_strconcat ()

```
gchar*      g_strconcat                     (const gchar *string1,
                                             ...);
```

Concatenates all of the given strings into one long string. The returned string should be freed when no longer needed.

Warning

The variable argument list *must* end with NULL. If you forget the NULL, g_strconcat () will start appending random memory junk to your string.

string1 : The first string to add, which must not be NULL.
... : a NULL-terminated list of strings to append to the string.
Returns : a newly-allocated string containing all the string arguments.

---

g_strjoin ()

```
gchar*      g_strjoin                       (const gchar *separator,
                                             ...);
```

Joins a number of strings together to form one long string, with the optional *separator* inserted between each of them.

separator : a string to insert between each of the strings, or NULL.
... : a NULL-terminated list of strings to join.
Returns : a newly-allocated string containing all of the strings joined together, with *separator* between them.

---

g_strjoinv ()

```
gchar*      g_strjoinv                      (const gchar *separator,
                                             gchar **str_array);
```

Joins a number of strings together to form one long string, with the optional *separator* inserted between each of them.

separator : a string to insert between each of the strings, or NULL.
str_array : a NULL-terminated array of strings to join.
Returns : a newly-allocated string containing all of the strings joined together, with *separator* between them.

---

g_strv_length ()

```
guint       g_strv_length                   (gchar **str_array);
```

Returns the length of the given NULL-terminated string array *str_array*.

str_array : a NULL-terminated array of strings.
Returns : length of *str_array*.

Since 2.6

---

g_strerror ()

```
G_CONST_RETURN gchar* g_strerror            (gint errnum);
```

Returns a string corresponding to the given error code, e.g. "no such process". This function is included since not all platforms support the strerror() function.

errnum : the system error number. See the standard C errno documentation.
Returns : a string describing the error code. If the error code is unknown, it returns "unknown error (<code>)". The string can only be used until the next call to g_strerror().

---

g_strsignal ()

```
G_CONST_RETURN gchar* g_strsignal           (gint signum);
```

Returns a string describing the given signal, e.g. "Segmentation fault". This function is included since not all platforms support the strsignal() function.

signum : the signal number. See the signal documentation.
Returns : a string describing the signal. If the signal is unknown, it returns "unknown signal (<signum>)". The string can only be used until the next call to g_strsignal().

<< GLib Utilities **Character Set Conversion >>**

Character Set Conversion

Character Set Conversion — convert strings between different character sets using `iconv()`.

Synopsis

```
#include <glib.h>


gchar*      g_convert                       (const gchar *str,
                                             gssize len,
                                             const gchar *to_codeset,
                                             const gchar *from_codeset,
                                             gsize *bytes_read,
                                             gsize *bytes_written,
                                             GError **error);
gchar*      g_convert_with_fallback         (const gchar *str,
                                             gssize len,
                                             const gchar *to_codeset,
                                             const gchar *from_codeset,
                                             gchar *fallback,
                                             gsize *bytes_read,
                                             gsize *bytes_written,
                                             GError **error);
            GIConv;
gchar*      g_convert_with_iconv            (const gchar *str,
                                             gssize len,
                                             GIConv converter,
                                             gsize *bytes_read,
                                             gsize *bytes_written,
                                             GError **error);
#define     G_CONVERT_ERROR
GIConv      g_iconv_open                    (const gchar *to_codeset,
                                             const gchar *from_codeset);
size_t      g_iconv                         (GIConv converter,
                                             gchar **inbuf,
                                             gsize *inbytes_left,
                                             gchar **outbuf,
                                             gsize *outbytes_left);
gint        g_iconv_close                   (GIConv converter);
gchar*      g_locale_to_utf8                (const gchar *opsysstring,
                                             gssize len,
                                             gsize *bytes_read,
                                             gsize *bytes_written,
                                             GError **error);
gchar*      g_filename_to_utf8              (const gchar *opsysstring,
                                             gssize len,
                                             gsize *bytes_read,
                                             gsize *bytes_written,
                                             GError **error);
gchar*      g_filename_from_utf8            (const gchar *utf8string,
                                             gssize len,
                                             gsize *bytes_read,
                                             gsize *bytes_written,
                                             GError **error);
```



```
gchar*      g_filename_from_uri             (const gchar *uri,
                                             gchar **hostname,
                                             GError **error);
gchar*      g_filename_to_uri               (const gchar *filename,
                                             const gchar *hostname,
                                             GError **error);
gboolean    g_get_filename_charsets         (G_CONST_RETURN gchar ***charsets);
gchar*      g_filename_display_name         (const gchar *filename);
gchar**     g_uri_list_extract_uris         (const gchar *uri_list);
gchar*      g_locale_from_utf8              (const gchar *utf8string,
                                             gssize len,
                                             gsize *bytes_read,
                                             gsize *bytes_written,
                                             GError **error);
enum        GConvertError;

gboolean    g_get_charset                   (G_CONST_RETURN char **charset);
```

Description

File Name Encodings

Historically, Unix has not had a defined encoding for file names: a file name is valid as long as it does not have path separators in it ("/"). However, displaying file names may require conversion: from the character set in which they were created, to the character set in which the application operates. Consider the Spanish file name "`Presentación.sxi`". If the application which created it uses ISO-8859-1 for its encoding, then the actual file name on disk would look like this:

```
Character:  P  r  e  s  e  n  t  a  c  i  ó  n  .  s  x  i
Hex code:   50 72 65 73 65 6e 74 61 63 69 f3 6e 2e 73 78 69
```

However, if the application use UTF-8, the actual file name on disk would look like this:

```
Character:  P  r  e  s  e  n  t  a  c  i  ó     n  .  s  x  i
Hex code:   50 72 65 73 65 6e 74 61 63 69 c3 b3 6e 2e 73 78 69
```

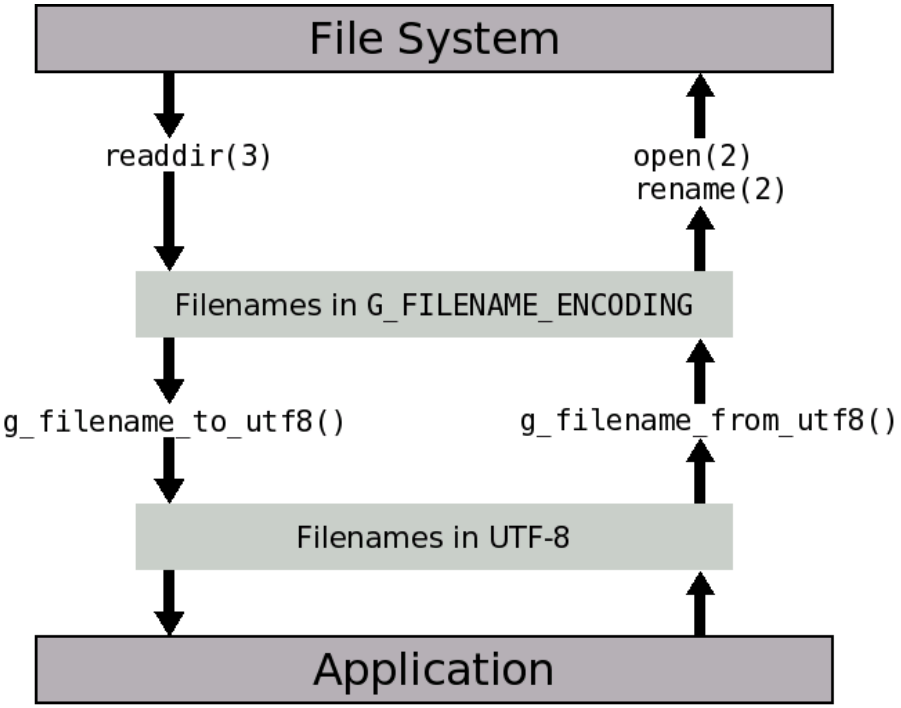
Glib uses UTF-8 for its strings, and GUI toolkits like GTK+ that use Glib do the same thing. If you get a file name from the file system, for example, from `readdir(3)` or from `g_dir_read_name()`, and you wish to display the file name to the user, you *will* need to convert it into UTF-8. The opposite case is when the user types the name of a file he wishes to save: the toolkit will give you that string in UTF-8 encoding, and you will need to convert it to the character set used for file names before you can create the file with `open(2)` or `fopen(3)`.

By default, Glib assumes that file names on disk are in UTF-8 encoding. This is a valid assumption for file systems which were created relatively recently: most applications use UTF-8 encoding for their strings, and that is also what they use for the file names they create. However, older file systems may still contain file names created in "older" encodings, such as ISO-8859-1. In this case, for compatibility reasons, you may want to instruct Glib to use that particular encoding for file names rather than UTF-8. You can do this by specifying the encoding for file names in the `G_FILENAME_ENCODING` environment variable. For example, if your installation uses ISO-8859-1 for file names, you can put this in your `~/.profile`:

```
export G_FILENAME_ENCODING=ISO-8859-1
```

Glib provides the functions `g_filename_to_utf8()` and `g_filename_from_utf8()` to perform the necessary conversions. These functions convert file names from the encoding specified in `G_FILENAME_ENCODING` to UTF-8 and vice-versa. Figure 1, “Conversion between File Name Encodings” illustrates how these functions are used to convert between UTF-8 and the encoding for file names in the file system.

Figure 1. Conversion between File Name Encodings



Checklist for Application Writers

This section is a practical summary of the detailed description above. You can use this as a checklist of things to do to make sure your applications process file name encodings correctly.

1. If you get a file name from the file system from a function such as `readdir(3)` or `gtk_file_chooser_get_filename()`, you do not need to do any conversion to pass that file name to functions like `open(2)`, `rename(2)`, or `fopen(3)` — those are "raw" file names which the file system understands.

2. If you need to display a file name, convert it to UTF-8 first by using `g_filename_to_utf8()`. If conversion fails, display a string like "`Unknown file name`". *Do not* convert this string back into the encoding used for file names if you wish to pass it to the file system; use the original file name instead. For example, the document window of a word processor could display "Unknown file name" in its title bar but still let the user save the file, as it would keep the raw file name internally. This can happen if the user has not set the `G_FILENAME_ENCODING` environment variable even though he has files whose names are not encoded in UTF-8.

3. If your user interface lets the user type a file name for saving or renaming, convert it to the encoding used for file names in the file system by using `g_filename_from_utf8()`. Pass the converted file name to functions like `fopen(3)`. If conversion fails, ask the user to enter a different file name. This can happen if the user types Japanese characters when `G_FILENAME_ENCODING` is set to `ISO-8859-1`, for example.

Details

g_convert ()

```
gchar*      g_convert                       (const gchar *str,
                                             gssize len,
                                             const gchar *to_codeset,
                                             const gchar *from_codeset,
                                             gsize *bytes_read,
                                             gsize *bytes_written,
                                             GError **error);
```

Converts a string from one character set to another.

str :	the string to convert
len :	the length of the string
to_codeset :	name of character set into which to convert *str*
from_codeset :	character set of *str*.
bytes_read :	location to store the number of bytes in the input string that were successfully converted, or `NULL`. Even if the conversion was successful, this may be less than *len* if there were partial characters at the end of the input. If the error G_CONVERT_ERROR_ILLEGAL_SEQUENCE occurs, the value stored will the byte offset after the last valid input sequence.
bytes_written :	the number of bytes stored in the output buffer (not including the terminating nul).
error :	location to store the error occuring, or `NULL` to ignore errors. Any of the errors in GConvertError may occur.
Returns :	If the conversion was successful, a newly allocated nul-terminated string, which must be freed with `g_free()`. Otherwise `NULL` and *error* will be set.

---

g_convert_with_fallback ()

```
gchar*      g_convert_with_fallback         (const gchar *str,
                                             gssize len,
                                             const gchar *to_codeset,
                                             const gchar *from_codeset,
                                             gchar *fallback,
```

```
                                        gsize *bytes_read,
                                        gsize *bytes_written,
                                        GError **error);
```

Converts a string from one character set to another, possibly including fallback sequences for characters not representable in the output. Note that it is not guaranteed that the specification for the fallback sequences in *fallback* will be honored. Some systems may do a approximate conversion from *from_codeset* to *to_codeset* in their iconv() functions, in which case GLib will simply return that approximate conversion.

str :	the string to convert
len :	the length of the string
to_codeset :	name of character set into which to convert *str*
from_codeset :	character set of *str*.
fallback :	UTF-8 string to use in place of character not present in the target encoding. (This must be in the target encoding), if NULL, characters not in the target encoding will be represented as Unicode escapes \uxxxx or \Uxxxxyyyy.
bytes_read :	location to store the number of bytes in the input string that were successfully converted, or NULL. Even if the conversion was successful, this may be less than *len* if there were partial characters at the end of the input.
bytes_written :	the number of bytes stored in the output buffer (not including the terminating nul).
error :	location to store the error occuring, or NULL to ignore errors. Any of the errors in GConvertError may occur.
Returns :	If the conversion was successful, a newly allocated nul-terminated string, which must be freed with g_free(). Otherwise NULL and *error* will be set.

GIConv

```
typedef struct _GIConv GIConv;
```

The GIConv struct wraps an iconv() conversion descriptor. It contains private data and should only be accessed using the following functions.

---

g_convert_with_iconv ()

```
gchar*      g_convert_with_iconv            (const gchar *str,
                                             gssize len,
                                             GIConv converter,
                                             gsize *bytes_read,
                                             gsize *bytes_written,
                                             GError **error);
```

Converts a string from one character set to another.

str :	the string to convert
len :	the length of the string
converter :	conversion descriptor from g_iconv_open()
bytes_read :	location to store the number of bytes in the input string that were successfully converted, or NULL. Even if the conversion was successful, this may be less than *len* if there were partial characters at the end of the input. If the error G_CONVERT_ERROR_ILLEGAL_SEQUENCE occurs, the value stored will the byte offset after the last valid input sequence.
bytes_written :	the number of bytes stored in the output buffer (not including the terminating nul).
error :	location to store the error occuring, or NULL to ignore errors. Any of the errors in GConvertError may occur.
Returns :	If the conversion was successful, a newly allocated nul-terminated string, which must be freed with g_free(). Otherwise NULL and *error* will be set.

G_CONVERT_ERROR

```
#define G_CONVERT_ERROR g_convert_error_quark()
```

Error domain for character set conversions. Errors in this domain will be from the GConvertError enumeration. See GError for information on error domains.

---

g_iconv_open ()

```
GIConv      g_iconv_open                    (const gchar *to_codeset,
                                             const gchar *from_codeset);
```

Same as the standard UNIX routine iconv_open(), but may be implemented via libiconv on UNIX flavors that lack a native implementation.

GLib provides g_convert() and g_locale_to_utf8() which are likely more convenient than the raw iconv wrappers.

to_codeset :	destination codeset
from_codeset :	source codeset
Returns :	a "conversion descriptor", or (GIConv)-1 if opening the converter failed.

---

g_iconv ()

```
size_t      g_iconv                         (GIConv converter,
                                             gchar **inbuf,
                                             gsize *inbytes_left,
                                             gchar **outbuf,
```

```
                                        gsize *outbytes_left);
```

Same as the standard UNIX routine `iconv()`, but may be implemented via libiconv on UNIX flavors that lack a native implementation.

GLib provides `g_convert()` and `g_locale_to_utf8()` which are likely more convenient than the raw iconv wrappers.

converter :	conversion descriptor from `g_iconv_open()`
inbuf :	bytes to convert
inbytes_left :	inout parameter, bytes remaining to convert in *inbuf*
outbuf :	converted output bytes
outbytes_left :	inout parameter, bytes available to fill in *outbuf*
Returns :	count of non-reversible conversions, or -1 on error

---

g_iconv_close ()

```
gint        g_iconv_close                   (GIConv converter);
```

Same as the standard UNIX routine `iconv_close()`, but may be implemented via libiconv on UNIX flavors that lack a native implementation. Should be called to clean up the conversion descriptor from `g_iconv_open()` when you are done converting things.

GLib provides `g_convert()` and `g_locale_to_utf8()` which are likely more convenient than the raw iconv wrappers.

converter :	a conversion descriptor from `g_iconv_open()`
Returns :	-1 on error, 0 on success

---

g_locale_to_utf8 ()

```
gchar*      g_locale_to_utf8                (const gchar *opsysstring,
                                             gssize len,
                                             gsize *bytes_read,
                                             gsize *bytes_written,
                                             GError **error);
```

Converts a string which is in the encoding used for strings by the C runtime (usually the same as that used by the operating system) in the current locale into a UTF-8 string.

opsysstring :	a string in the encoding of the current locale
len :	the length of the string, or -1 if the string is nul-terminated.
bytes_read :	location to store the number of bytes in the input string that were successfully converted, or `NULL`. Even if the conversion was successful, this may be less than *len* if there were partial characters at the end of the input. If the error G_CONVERT_ERROR_ILLEGAL_SEQUENCE

	occurs, the value stored will the byte offset after the last valid input sequence.
bytes_written :	the number of bytes stored in the output buffer (not including the terminating nul).
error :	location to store the error occuring, or `NULL` to ignore errors. Any of the errors in GConvertError may occur.
Returns :	The converted string, or `NULL` on an error.

---

g_filename_to_utf8 ()

```
gchar*      g_filename_to_utf8              (const gchar *opsysstring,
                                             gssize len,
                                             gsize *bytes_read,
                                             gsize *bytes_written,
                                             GError **error);
```

Converts a string which is in the encoding used for filenames into a UTF-8 string.

opsysstring :	a string in the encoding for filenames
len :	the length of the string, or -1 if the string is nul-terminated.
bytes_read :	location to store the number of bytes in the input string that were successfully converted, or `NULL`. Even if the conversion was successful, this may be less than *len* if there were partial characters at the end of the input. If the error G_CONVERT_ERROR_ILLEGAL_SEQUENCE occurs, the value stored will the byte offset after the last valid input sequence.
bytes_written :	the number of bytes stored in the output buffer (not including the terminating nul).
error :	location to store the error occuring, or `NULL` to ignore errors. Any of the errors in GConvertError may occur.
Returns :	The converted string, or `NULL` on an error.

---

g_filename_from_utf8 ()

```
gchar*      g_filename_from_utf8            (const gchar *utf8string,
                                             gssize len,
                                             gsize *bytes_read,
                                             gsize *bytes_written,
                                             GError **error);
```

Converts a string from UTF-8 to the encoding used for filenames.

utf8string :	a UTF-8 encoded string.
len :	the length of the string, or -1 if the string is nul-terminated.
bytes_read :	location to store the number of bytes in the input string that were successfully converted, or `NULL`. Even if the conversion was successful, this may be less than *len* if there were partial characters at the end of the

input. If the error G_CONVERT_ERROR_ILLEGAL_SEQUENCE occurs, the value stored will the byte offset after the last valid input sequence.

bytes_written : the number of bytes stored in the output buffer (not including the terminating nul).

error : location to store the error occuring, or NULL to ignore errors. Any of the errors in GConvertError may occur.

Returns : The converted string, or NULL on an error.

---

g_filename_from_uri ()

```
gchar*      g_filename_from_uri             (const gchar *uri,
                                             gchar **hostname,
                                             GError **error);
```

Converts an escaped ASCII-encoded URI to a local filename in the encoding used for filenames.

uri : a uri describing a filename (escaped, encoded in ASCII).

hostname : Location to store hostname for the URI, or NULL. If there is no hostname in the URI, NULL will be stored in this location.

error : location to store the error occuring, or NULL to ignore errors. Any of the errors in GConvertError may occur.

Returns : a newly-allocated string holding the resulting filename, or NULL on an error.

---

g_filename_to_uri ()

```
gchar*      g_filename_to_uri               (const gchar *filename,
                                             const gchar *hostname,
                                             GError **error);
```

Converts an absolute filename to an escaped ASCII-encoded URI.

filename : an absolute filename specified in the encoding used for filenames by the operating system.

hostname : A UTF-8 encoded hostname, or NULL for none.

error : location to store the error occuring, or NULL to ignore errors. Any of the errors in GConvertError may occur.

Returns : a newly-allocated string holding the resulting URI, or NULL on an error.

---

g_get_filename_charsets ()

```
gboolean    g_get_filename_charsets         (G_CONST_RETURN gchar ***charsets);
```

Determines the preferred character sets used for filenames. The first character set from the *charsets*

is the filename encoding, the subsequent character sets are used when trying to generate a displayable representation of a filename, see g_filename_display_name().

The character sets are determined by consulting the environment variables G_FILENAME_ENCODING and G_BROKEN_FILENAMES.

G_FILENAME_ENCODING may be set to a comma-separated list of character set names. The special token "*locale*" is taken to mean the character set for the current locale. If G_FILENAME_ENCODING is not set, but G_BROKEN_FILENAMES is, the character set of the current locale is taken as the filename encoding. If neither environment variable is set, UTF-8 is taken as the filename encoding, but the character set of the current locale is also put in the list of encodings.

The returned *charsets* belong to GLib and must not be freed.

Note that on Unix, regardless of the locale character set or G_FILENAME_ENCODING value, the actual file names present on a system might be in any random encoding or just gibberish.

charsets : return location for the NULL-terminated list of encoding names

Returns : TRUE if the filename encoding is UTF-8.

Since 2.6

---

g_filename_display_name ()

```
gchar*      g_filename_display_name         (const gchar *filename);
```

Converts a filename into a valid UTF-8 string. The conversion is not necessarily reversible, so you should keep the original around and use the return value of this function only for display purposes. Unlike g_filename_to_utf8(), the result is guaranteed to be non-NULL even if the filename actually isn't in the GLib file name encoding.

filename : a pathname hopefully in the GLib file name encoding

Returns : a newly allocated string containing a rendition of the filename in valid UTF-8

Since 2.6

---

g_uri_list_extract_uris ()

```
gchar**     g_uri_list_extract_uris         (const gchar *uri_list);
```

Splits an URI list conforming to the text/uri-list mime type defined in RFC 2483 into individual URIs, discarding any comments. The URIs are not validated.

uri_list : an URI list

Returns : a newly allocated NULL-terminated list of strings holding the individual URIs.

The array should be freed with g_strfreev().

Since 2.6

---

g_locale_from_utf8 ()

```
gchar*      g_locale_from_utf8              (const gchar *utf8string,
                                             gssize len,
                                             gsize *bytes_read,
                                             gsize *bytes_written,
                                             GError **error);
```

Converts a string from UTF-8 to the encoding used for strings by the C runtime (usually the same as that used by the operating system) in the current locale.

utf8string : a UTF-8 encoded string

len : the length of the string, or -1 if the string is nul-terminated.

bytes_read : location to store the number of bytes in the input string that were successfully converted, or NULL. Even if the conversion was successful, this may be less than *len* if there were partial characters at the end of the input. If the error G_CONVERT_ERROR_ILLEGAL_SEQUENCE occurs, the value stored will the byte offset after the last valid input sequence.

bytes_written : the number of bytes stored in the output buffer (not including the terminating nul).

error : location to store the error occuring, or NULL to ignore errors. Any of the errors in GConvertError may occur.

Returns : The converted string, or NULL on an error.

---

enum GConvertError

```
typedef enum
{
  G_CONVERT_ERROR_NO_CONVERSION,
  G_CONVERT_ERROR_ILLEGAL_SEQUENCE,
  G_CONVERT_ERROR_FAILED,
  G_CONVERT_ERROR_PARTIAL_INPUT,
  G_CONVERT_ERROR_BAD_URI,
  G_CONVERT_ERROR_NOT_ABSOLUTE_PATH
} GConvertError;
```

Error codes returned by character set conversion routines.

G_CONVERT_ERROR_NO_CONVERSION	Conversion between the requested character sets is not supported.
G_CONVERT_ERROR_ILLEGAL_SEQUENCE	Invalid byte sequence in conversion input.
G_CONVERT_ERROR_FAILED	Conversion failed for some reason.

G_CONVERT_ERROR_PARTIAL_INPUT	Partial character sequence at end of input.
G_CONVERT_ERROR_BAD_URI	URI is invalid.
G_CONVERT_ERROR_NOT_ABSOLUTE_PATH	Pathname is not an absolute path.

---

g_get_charset ()

```
gboolean    g_get_charset                   (G_CONST_RETURN char **charset);
```

Obtains the character set for the current locale; you might use this character set as an argument to g_convert(), to convert from the current locale's encoding to some other encoding. (Frequently g_locale_to_utf8() and g_locale_from_utf8() are nice shortcuts, though.)

The return value is TRUE if the locale's encoding is UTF-8, in that case you can perhaps avoid calling g_convert().

The string returned in *charset* is not allocated, and should not be freed.

charset : return location for character set name

Returns : TRUE if the returned charset is UTF-8

<< String Utility Functions **Unicode Manipulation >>**

Unicode Manipulation

Unicode Manipulation — functions operating on Unicode characters and UTF-8 strings.

Synopsis

```
#include <glib.h>


typedef     gunichar;
typedef     gunichar2;

gboolean    g_unichar_validate              (gunichar ch);
gboolean    g_unichar_isalnum               (gunichar c);
gboolean    g_unichar_isalpha               (gunichar c);
gboolean    g_unichar_iscntrl               (gunichar c);
gboolean    g_unichar_isdigit               (gunichar c);
gboolean    g_unichar_isgraph               (gunichar c);
gboolean    g_unichar_islower               (gunichar c);
gboolean    g_unichar_isprint               (gunichar c);
gboolean    g_unichar_ispunct               (gunichar c);
gboolean    g_unichar_isspace               (gunichar c);
gboolean    g_unichar_isupper               (gunichar c);
gboolean    g_unichar_isxdigit              (gunichar c);
gboolean    g_unichar_istitle               (gunichar c);
gboolean    g_unichar_isdefined             (gunichar c);
gboolean    g_unichar_iswide                (gunichar c);
gunichar    g_unichar_toupper               (gunichar c);
gunichar    g_unichar_tolower               (gunichar c);
gunichar    g_unichar_totitle               (gunichar c);
gint        g_unichar_digit_value           (gunichar c);
gint        g_unichar_xdigit_value          (gunichar c);
enum        GUnicodeType;
GUnicodeType g_unichar_type                 (gunichar c);
enum        GUnicodeBreakType;
GUnicodeBreakType g_unichar_break_type      (gunichar c);
void        g_unicode_canonical_ordering    (gunichar *string,
                                             gsize len);
gunichar*   g_unicode_canonical_decomposition
                                            (gunichar ch,
                                             gsize *result_len);
gboolean    g_unichar_get_mirror_char       (gunichar ch,
                                             gunichar *mirrored_ch);

#define     g_utf8_next_char                (p)
gunichar    g_utf8_get_char                 (const gchar *p);
gunichar    g_utf8_get_char_validated       (const  gchar *p,
                                             gssize max_len);
gchar*      g_utf8_offset_to_pointer        (const gchar *str,
                                             glong offset);
glong       g_utf8_pointer_to_offset        (const gchar *str,
                                             const gchar *pos);
gchar*      g_utf8_prev_char                (const gchar *p);
gchar*      g_utf8_find_next_char           (const gchar *p,
                                             const gchar *end);
```



```
gchar*      g_utf8_find_prev_char           (const gchar *str,
                                             const gchar *p);
glong       g_utf8_strlen                   (const gchar *p,
                                             gssize max);
gchar*      g_utf8_strncpy                  (gchar *dest,
                                             const gchar *src,
                                             gsize n);
gchar*      g_utf8_strchr                   (const gchar *p,
                                             gssize len,
                                             gunichar c);
gchar*      g_utf8_strrchr                  (const gchar *p,
                                             gssize len,
                                             gunichar c);
gchar*      g_utf8_strreverse               (const gchar *str,
                                             gssize len);
gboolean    g_utf8_validate                 (const gchar *str,
                                             gssize max_len,
                                             const gchar **end);

gchar*      g_utf8_strup                    (const gchar *str,
                                             gssize len);
gchar*      g_utf8_strdown                  (const gchar *str,
                                             gssize len);
gchar*      g_utf8_casefold                 (const gchar *str,
                                             gssize len);
gchar*      g_utf8_normalize                (const gchar *str,
                                             gssize len,
                                             GNormalizeMode mode);
enum        GNormalizeMode;
gint        g_utf8_collate                  (const gchar *str1,
                                             const gchar *str2);
gchar*      g_utf8_collate_key              (const gchar *str,
                                             gssize len);

gunichar2*  g_utf8_to_utf16                 (const gchar *str,
                                             glong len,
                                             glong *items_read,
                                             glong *items_written,
                                             GError **error);
gunichar*   g_utf8_to_ucs4                  (const gchar *str,
                                             glong len,
                                             glong *items_read,
                                             glong *items_written,
                                             GError **error);
gunichar*   g_utf8_to_ucs4_fast             (const gchar *str,
                                             glong len,
                                             glong *items_written);
gunichar*   g_utf16_to_ucs4                 (const gunichar2 *str,
                                             glong len,
                                             glong *items_read,
                                             glong *items_written,
                                             GError **error);
gchar*      g_utf16_to_utf8                 (const gunichar2 *str,
                                             glong len,
                                             glong *items_read,
                                             glong *items_written,
                                             GError **error);
gunichar2*  g_ucs4_to_utf16                 (const gunichar *str,
                                             glong len,
                                             glong *items_read,
                                             glong *items_written,
                                             GError **error);
gchar*      g_ucs4_to_utf8                  (const gunichar *str,
                                             glong len,
```

```
                                        glong *items_read,
                                        glong *items_written,
                                        GError **error);
gint        g_unichar_to_utf8           (gunichar c,
                                        gchar *outbuf);
```

Description

This section describes a number of functions for dealing with Unicode characters and strings. There are analogues of the traditional `ctype.h` character classification and case conversion functions, UTF-8 analogues of some string utility functions, functions to perform normalization, case conversion and collation on UTF-8 strings and finally functions to convert between the UTF-8, UTF-16 and UCS-4 encodings of Unicode.

Details

gunichar

```
typedef guint32 gunichar;
```

A type which can hold any UCS-4 character code.

---

gunichar2

```
typedef guint16 gunichar2;
```

A type which can hold any UTF-16 character code.

---

g_unichar_validate ()

```
gboolean    g_unichar_validate              (gunichar ch);
```

Checks whether *ch* is a valid Unicode character. Some possible integer values of *ch* will not be valid. 0 is considered a valid character, though it's normally a string terminator.

ch : a Unicode character
Returns : `TRUE` if *ch* is a valid Unicode character

---

g_unichar_isalnum ()

```
gboolean    g_unichar_isalnum               (gunichar c);
```

Determines whether a character is alphanumeric. Given some UTF-8 text, obtain a character value

with `g_utf8_get_char()`.

c : a Unicode character
Returns : `TRUE` if *c* is an alphanumeric character

---

g_unichar_isalpha ()

```
gboolean    g_unichar_isalpha               (gunichar c);
```

Determines whether a character is alphabetic (i.e. a letter). Given some UTF-8 text, obtain a character value with `g_utf8_get_char()`.

c : a Unicode character
Returns : `TRUE` if *c* is an alphabetic character

---

g_unichar_iscntrl ()

```
gboolean    g_unichar_iscntrl               (gunichar c);
```

Determines whether a character is a control character. Given some UTF-8 text, obtain a character value with `g_utf8_get_char()`.

c : a Unicode character
Returns : `TRUE` if *c* is a control character

---

g_unichar_isdigit ()

```
gboolean    g_unichar_isdigit               (gunichar c);
```

Determines whether a character is numeric (i.e. a digit). This covers ASCII 0-9 and also digits in other languages/scripts. Given some UTF-8 text, obtain a character value with `g_utf8_get_char()`.

c : a Unicode character
Returns : `TRUE` if *c* is a digit

---

g_unichar_isgraph ()

```
gboolean    g_unichar_isgraph               (gunichar c);
```

Determines whether a character is printable and not a space (returns `FALSE` for control characters, format characters, and spaces). `g_unichar_isprint()` is similar, but returns `TRUE` for spaces. Given some UTF-8 text, obtain a character value with `g_utf8_get_char()`.

c : a Unicode character
Returns : TRUE if *c* is printable unless it's a space

---

g_unichar_islower ()

```
gboolean    g_unichar_islower               (gunichar c);
```

Determines whether a character is a lowercase letter. Given some UTF-8 text, obtain a character value with g_utf8_get_char().

c : a Unicode character
Returns : TRUE if *c* is a lowercase letter

---

g_unichar_isprint ()

```
gboolean    g_unichar_isprint               (gunichar c);
```

Determines whether a character is printable. Unlike g_unichar_isgraph(), returns TRUE for spaces. Given some UTF-8 text, obtain a character value with g_utf8_get_char().

c : a Unicode character
Returns : TRUE if *c* is printable

---

g_unichar_ispunct ()

```
gboolean    g_unichar_ispunct               (gunichar c);
```

Determines whether a character is punctuation or a symbol. Given some UTF-8 text, obtain a character value with g_utf8_get_char().

c : a Unicode character
Returns : TRUE if *c* is a punctuation or symbol character

---

g_unichar_isspace ()

```
gboolean    g_unichar_isspace               (gunichar c);
```

Determines whether a character is a space, tab, or line separator (newline, carriage return, etc.). Given some UTF-8 text, obtain a character value with g_utf8_get_char().

(Note: don't use this to do word breaking; you have to use Pango or equivalent to get word breaking right, the algorithm is fairly complex.)

c : a Unicode character
Returns : TRUE if *c* is a punctuation character

---

g_unichar_isupper ()

```
gboolean    g_unichar_isupper               (gunichar c);
```

Determines if a character is uppercase.

c : a Unicode character
Returns : TRUE if *c* is an uppercase character

---

g_unichar_isxdigit ()

```
gboolean    g_unichar_isxdigit              (gunichar c);
```

Determines if a character is a hexidecimal digit.

c : a Unicode character.
Returns : TRUE if the character is a hexadecimal digit

---

g_unichar_istitle ()

```
gboolean    g_unichar_istitle               (gunichar c);
```

Determines if a character is titlecase. Some characters in Unicode which are composites, such as the DZ digraph have three case variants instead of just two. The titlecase form is used at the beginning of a word where only the first letter is capitalized. The titlecase form of the DZ digraph is U+01F2 LATIN CAPITAL LETTTER D WITH SMALL LETTER Z.

c : a Unicode character
Returns : TRUE if the character is titlecase

---

g_unichar_isdefined ()

```
gboolean    g_unichar_isdefined             (gunichar c);
```

Determines if a given character is assigned in the Unicode standard.

c : a Unicode character
Returns : TRUE if the character has an assigned value

g_unichar_iswide ()

```
gboolean    g_unichar_iswide                (gunichar c);
```

Determines if a character is typically rendered in a double-width cell.

c : a Unicode character
Returns : TRUE if the character is wide

g_unichar_toupper ()

```
gunichar    g_unichar_toupper               (gunichar c);
```

Converts a character to uppercase.

c : a Unicode character
Returns : the result of converting *c* to uppercase. If *c* is not an lowercase or titlecase character, or has no upper case equivalent *c* is returned unchanged.

g_unichar_tolower ()

```
gunichar    g_unichar_tolower               (gunichar c);
```

Converts a character to lower case.

c : a Unicode character.
Returns : the result of converting *c* to lower case. If *c* is not an upperlower or titlecase character, or has no lowercase equivalent *c* is returned unchanged.

g_unichar_totitle ()

```
gunichar    g_unichar_totitle               (gunichar c);
```

Converts a character to the titlecase.

c : a Unicode character
Returns : the result of converting *c* to titlecase. If *c* is not an uppercase or lowercase character, *c* is returned unchanged.

g_unichar_digit_value ()


```
gint        g_unichar_digit_value           (gunichar c);
```

Determines the numeric value of a character as a decimal digit.

c : a Unicode character
Returns : If *c* is a decimal digit (according to g_unichar_isdigit()), its numeric value. Otherwise, -1.

g_unichar_xdigit_value ()

```
gint        g_unichar_xdigit_value          (gunichar c);
```

Determines the numeric value of a character as a hexidecimal digit.

c : a Unicode character
Returns : If *c* is a hex digit (according to g_unichar_isxdigit()), its numeric value. Otherwise, -1.

enum GUnicodeType

```
typedef enum
{
  G_UNICODE_CONTROL,
  G_UNICODE_FORMAT,
  G_UNICODE_UNASSIGNED,
  G_UNICODE_PRIVATE_USE,
  G_UNICODE_SURROGATE,
  G_UNICODE_LOWERCASE_LETTER,
  G_UNICODE_MODIFIER_LETTER,
  G_UNICODE_OTHER_LETTER,
  G_UNICODE_TITLECASE_LETTER,
  G_UNICODE_UPPERCASE_LETTER,
  G_UNICODE_COMBINING_MARK,
  G_UNICODE_ENCLOSING_MARK,
  G_UNICODE_NON_SPACING_MARK,
  G_UNICODE_DECIMAL_NUMBER,
  G_UNICODE_LETTER_NUMBER,
  G_UNICODE_OTHER_NUMBER,
  G_UNICODE_CONNECT_PUNCTUATION,
  G_UNICODE_DASH_PUNCTUATION,
  G_UNICODE_CLOSE_PUNCTUATION,
  G_UNICODE_FINAL_PUNCTUATION,
  G_UNICODE_INITIAL_PUNCTUATION,
  G_UNICODE_OTHER_PUNCTUATION,
  G_UNICODE_OPEN_PUNCTUATION,
  G_UNICODE_CURRENCY_SYMBOL,
  G_UNICODE_MODIFIER_SYMBOL,
  G_UNICODE_MATH_SYMBOL,
  G_UNICODE_OTHER_SYMBOL,
  G_UNICODE_LINE_SEPARATOR,
  G_UNICODE_PARAGRAPH_SEPARATOR,
  G_UNICODE_SPACE_SEPARATOR
```

```
} GUnicodeType;
```

These are the possible character classifications. See http://www.unicode.org/Public/UNIDATA/UnicodeData.html.

---

g_unichar_type ()

```
GUnicodeType g_unichar_type                (gunichar c);
```

Classifies a Unicode character by type.

c : a Unicode character
Returns : the type of the character.

---

enum GUnicodeBreakType

```
typedef enum
{
  G_UNICODE_BREAK_MANDATORY,
  G_UNICODE_BREAK_CARRIAGE_RETURN,
  G_UNICODE_BREAK_LINE_FEED,
  G_UNICODE_BREAK_COMBINING_MARK,
  G_UNICODE_BREAK_SURROGATE,
  G_UNICODE_BREAK_ZERO_WIDTH_SPACE,
  G_UNICODE_BREAK_INSEPARABLE,
  G_UNICODE_BREAK_NON_BREAKING_GLUE,
  G_UNICODE_BREAK_CONTINGENT,
  G_UNICODE_BREAK_SPACE,
  G_UNICODE_BREAK_AFTER,
  G_UNICODE_BREAK_BEFORE,
  G_UNICODE_BREAK_BEFORE_AND_AFTER,
  G_UNICODE_BREAK_HYPHEN,
  G_UNICODE_BREAK_NON_STARTER,
  G_UNICODE_BREAK_OPEN_PUNCTUATION,
  G_UNICODE_BREAK_CLOSE_PUNCTUATION,
  G_UNICODE_BREAK_QUOTATION,
  G_UNICODE_BREAK_EXCLAMATION,
  G_UNICODE_BREAK_IDEOGRAPHIC,
  G_UNICODE_BREAK_NUMERIC,
  G_UNICODE_BREAK_INFIX_SEPARATOR,
  G_UNICODE_BREAK_SYMBOL,
  G_UNICODE_BREAK_ALPHABETIC,
  G_UNICODE_BREAK_PREFIX,
  G_UNICODE_BREAK_POSTFIX,
  G_UNICODE_BREAK_COMPLEX_CONTEXT,
  G_UNICODE_BREAK_AMBIGUOUS,
  G_UNICODE_BREAK_UNKNOWN,
  G_UNICODE_BREAK_NEXT_LINE,
  G_UNICODE_BREAK_WORD_JOINER
} GUnicodeBreakType;
```

These are the possible line break classifications. See http://www.unicode.org/unicode/reports/tr14/.

---

g_unichar_break_type ()

```
GUnicodeBreakType g_unichar_break_type      (gunichar c);
```

Determines the break type of *c*. *c* should be a Unicode character (to derive a character from UTF-8 encoded text, use `g_utf8_get_char()`). The break type is used to find word and line breaks ("text boundaries"), Pango implements the Unicode boundary resolution algorithms and normally you would use a function such as `pango_break()` instead of caring about break types yourself.

c : a Unicode character
Returns : the break type of *c*

---

g_unicode_canonical_ordering ()

```
void        g_unicode_canonical_ordering    (gunichar *string,
                                             gsize len);
```

Computes the canonical ordering of a string in-place. This rearranges decomposed characters in the string according to their combining classes. See the Unicode manual for more information.

string : a UCS-4 encoded string.
len : the maximum length of *string* to use.

---

g_unicode_canonical_decomposition ()

```
gunichar*   g_unicode_canonical_decomposition
                                            (gunichar ch,
                                             gsize *result_len);
```

Computes the canonical decomposition of a Unicode character.

ch : a Unicode character.
result_len : location to store the length of the return value.
Returns : a newly allocated string of Unicode characters. *result_len* is set to the resulting length of the string.

---

g_unichar_get_mirror_char ()

```
gboolean    g_unichar_get_mirror_char       (gunichar ch,
                                             gunichar *mirrored_ch);
```

In Unicode, some characters are *mirrored*. This means that their images are mirrored horizontally in

text that is laid out from right to left. For instance, "(" would become its mirror image, ")", in right-to-left text.

If *ch* has the Unicode mirrored property and there is another unicode character that typically has a glyph that is the mirror image of *ch*'s glyph, puts that character in the address pointed to by *mirrored_ch*.

ch : a unicode character
mirrored_ch : location to store the mirrored character
Returns : TRUE if *ch* has a mirrored character and *mirrored_ch* is filled in, FALSE otherwise

Since 2.4

---

g_utf8_next_char()

```
#define     g_utf8_next_char(p)
```

Skips to the next character in a UTF-8 string. The string must be valid; this macro is as fast as possible, and has no error-checking. You would use this macro to iterate over a string character by character. The macro returns the start of the next UTF-8 character. Before using this macro, use g_utf8_validate() to validate strings that may contain invalid UTF-8.

p : Pointer to the start of a valid UTF-8 character.

---

g_utf8_get_char ()

```
gunichar    g_utf8_get_char                 (const gchar *p);
```

Converts a sequence of bytes encoded as UTF-8 to a Unicode character. If *p* does not point to a valid UTF-8 encoded character, results are undefined. If you are not sure that the bytes are complete valid Unicode characters, you should use g_utf8_get_char_validated() instead.

p : a pointer to Unicode character encoded as UTF-8
Returns : the resulting character

---

g_utf8_get_char_validated ()

```
gunichar    g_utf8_get_char_validated       (const  gchar *p,
                                             gssize max_len);
```

Convert a sequence of bytes encoded as UTF-8 to a Unicode character. This function checks for incomplete characters, for invalid characters such as characters that are out of the range of Unicode, and for overlong encodings of valid characters.

p : a pointer to Unicode character encoded as UTF-8
max_len : the maximum number of bytes to read, or -1, for no maximum.
Returns : the resulting character. If *p* points to a partial sequence at the end of a string that could begin a valid character, returns (gunichar)-2; otherwise, if *p* does not point to a valid UTF-8 encoded Unicode character, returns (gunichar)-1.

---

g_utf8_offset_to_pointer ()

```
gchar*      g_utf8_offset_to_pointer        (const gchar *str,
                                             glong offset);
```

Converts from an integer character offset to a pointer to a position within the string.

str : a UTF-8 encoded string
offset : a character offset within *str*
Returns : the resulting pointer

---

g_utf8_pointer_to_offset ()

```
glong       g_utf8_pointer_to_offset        (const gchar *str,
                                             const gchar *pos);
```

Converts from a pointer to position within a string to a integer character offset.

str : a UTF-8 encoded string
pos : a pointer to a position within *str*
Returns : the resulting character offset

---

g_utf8_prev_char ()

```
gchar*      g_utf8_prev_char                (const gchar *p);
```

Finds the previous UTF-8 character in the string before *p*.

p does not have to be at the beginning of a UTF-8 character. No check is made to see if the character found is actually valid other than it starts with an appropriate byte. If *p* might be the first character of the string, you must use g_utf8_find_prev_char() instead.

p : a pointer to a position within a UTF-8 encoded string
Returns : a pointer to the found character.

---

g_utf8_find_next_char ()

```
gchar*      g_utf8_find_next_char           (const gchar *p,
                                             const gchar *end);
```

Finds the start of the next UTF-8 character in the string after *p*.

p does not have to be at the beginning of a UTF-8 character. No check is made to see if the character found is actually valid other than it starts with an appropriate byte.

p : a pointer to a position within a UTF-8 encoded string
end : a pointer to the end of the string, or NULL to indicate that the string is nul-terminated, in which case the returned value will be
Returns : a pointer to the found character or NULL

---

g_utf8_find_prev_char ()

```
gchar*      g_utf8_find_prev_char           (const gchar *str,
                                             const gchar *p);
```

Given a position *p* with a UTF-8 encoded string *str*, find the start of the previous UTF-8 character starting before *p*. Returns NULL if no UTF-8 characters are present in *p* before *str*.

p does not have to be at the beginning of a UTF-8 character. No check is made to see if the character found is actually valid other than it starts with an appropriate byte.

str : pointer to the beginning of a UTF-8 encoded string
p : pointer to some position within *str*
Returns : a pointer to the found character or NULL.

---

g_utf8_strlen ()

```
glong       g_utf8_strlen                   (const gchar *p,
                                             gssize max);
```

Returns the length of the string in characters.

p : pointer to the start of a UTF-8 encoded string.
max : the maximum number of bytes to examine. If *max* is less than 0, then the string is assumed to be nul-terminated. If *max* is 0, *p* will not be examined and may be NULL.
Returns : the length of the string in characters

---

g_utf8_strncpy ()

```
gchar*      g_utf8_strncpy                  (gchar *dest,
```



```
                                             const gchar *src,
                                             gsize n);
```

Like the standard C strncpy() function, but copies a given number of characters instead of a given number of bytes. The *src* string must be valid UTF-8 encoded text. (Use g_utf8_validate() on all text before trying to use UTF-8 utility functions with it.)

dest : buffer to fill with characters from *src*
src : UTF-8 encoded string
n : character count
Returns : *dest*

---

g_utf8_strchr ()

```
gchar*      g_utf8_strchr                   (const gchar *p,
                                             gssize len,
                                             gunichar c);
```

Finds the leftmost occurrence of the given ISO10646 character in a UTF-8 encoded string, while limiting the search to *len* bytes. If *len* is -1, allow unbounded search.

p : a nul-terminated UTF-8 encoded string
len : the maximum length of *p*
c : a ISO10646 character
Returns : NULL if the string does not contain the character, otherwise, a pointer to the start of the leftmost occurrence of the character in the string.

---

g_utf8_strrchr ()

```
gchar*      g_utf8_strrchr                  (const gchar *p,
                                             gssize len,
                                             gunichar c);
```

Find the rightmost occurrence of the given ISO10646 character in a UTF-8 encoded string, while limiting the search to *len* bytes. If *len* is -1, allow unbounded search.

p : a nul-terminated UTF-8 encoded string
len : the maximum length of *p*
c : a ISO10646 character
Returns : NULL if the string does not contain the character, otherwise, a pointer to the start of the rightmost occurrence of the character in the string.

---

g_utf8_strreverse ()

```
gchar*      g_utf8_strreverse               (const gchar *str,
                                             gssize len);
```

Reverses a UTF-8 string. *str* must be valid UTF-8 encoded text. (Use g_utf8_validate() on all text before trying to use UTF-8 utility functions with it.)

Note that unlike g_strreverse(), this function returns newly-allocated memory, which should be freed with g_free() when no longer needed.

str : a UTF-8 encoded string
len : the maximum length of *str* to use. If *len* < 0, then the string is nul-terminated.
Returns : a newly-allocated string which is the reverse of *str*.

Since 2.2

---

g_utf8_validate ()

```
gboolean    g_utf8_validate                 (const gchar *str,
                                             gssize max_len,
                                             const gchar **end);
```

Validates UTF-8 encoded text. *str* is the text to validate; if *str* is nul-terminated, then *max_len* can be -1, otherwise *max_len* should be the number of bytes to validate. If *end* is non-NULL, then the end of the valid range will be stored there (i.e. the address of the first invalid byte if some bytes were invalid, or the end of the text being validated otherwise).

Returns TRUE if all of *str* was valid. Many GLib and GTK+ routines *require* valid UTF-8 as input; so data read from a file or the network should be checked with g_utf8_validate() before doing anything else with it.

str : a pointer to character data
max_len : max bytes to validate, or -1 to go until nul
end : return location for end of valid data
Returns : TRUE if the text was valid UTF-8

---

g_utf8_strup ()

```
gchar*      g_utf8_strup                    (const gchar *str,
                                             gssize len);
```

Converts all Unicode characters in the string that have a case to uppercase. The exact manner that this is done depends on the current locale, and may result in the number of characters in the string increasing. (For instance, the German ess-zet will be changed to SS.)

str : a UTF-8 encoded string

len : length of *str*, in bytes, or -1 if *str* is nul-terminated.
Returns : a newly allocated string, with all characters converted to uppercase.

---

g_utf8_strdown ()

```
gchar*      g_utf8_strdown                  (const gchar *str,
                                             gssize len);
```

Converts all Unicode characters in the string that have a case to lowercase. The exact manner that this is done depends on the current locale, and may result in the number of characters in the string changing.

str : a UTF-8 encoded string
len : length of *str*, in bytes, or -1 if *str* is nul-terminated.
Returns : a newly allocated string, with all characters converted to lowercase.

---

g_utf8_casefold ()

```
gchar*      g_utf8_casefold                 (const gchar *str,
                                             gssize len);
```

Converts a string into a form that is independent of case. The result will not correspond to any particular case, but can be compared for equality or ordered with the results of calling g_utf8_casefold() on other strings.

Note that calling g_utf8_casefold() followed by g_utf8_collate() is only an approximation to the correct linguistic case insensitive ordering, though it is a fairly good one. Getting this exactly right would require a more sophisticated collation function that takes case sensitivity into account. GLib does not currently provide such a function.

str : a UTF-8 encoded string
len : length of *str*, in bytes, or -1 if *str* is nul-terminated.
Returns : a newly allocated string, that is a case independent form of *str*.

---

g_utf8_normalize ()

```
gchar*      g_utf8_normalize                (const gchar *str,
                                             gssize len,
                                             GNormalizeMode mode);
```

Converts a string into canonical form, standardizing such issues as whether a character with an accent is represented as a base character and combining accent or as a single precomposed character. You should generally call g_utf8_normalize() before comparing two Unicode strings.

The normalization mode G_NORMALIZE_DEFAULT only standardizes differences that do not affect the

text content, such as the above-mentioned accent representation. G_NORMALIZE_ALL also standardizes the "compatibility" characters in Unicode, such as SUPERSCRIPT THREE to the standard forms (in this case DIGIT THREE). Formatting information may be lost but for most text operations such characters should be considered the same. For example, g_utf8_collate() normalizes with G_NORMALIZE_ALL as its first step.

G_NORMALIZE_DEFAULT_COMPOSE and G_NORMALIZE_ALL_COMPOSE are like G_NORMALIZE_DEFAULT and G_NORMALIZE_ALL, but returned a result with composed forms rather than a maximally decomposed form. This is often useful if you intend to convert the string to a legacy encoding or pass it to a system with less capable Unicode handling.

str : a UTF-8 encoded string.
len : length of *str*, in bytes, or -1 if *str* is nul-terminated.
mode : the type of normalization to perform.
Returns : a newly allocated string, that is the normalized form of *str*.

---

enum GNormalizeMode

```
typedef enum {
  G_NORMALIZE_DEFAULT,
  G_NORMALIZE_NFD = G_NORMALIZE_DEFAULT,
  G_NORMALIZE_DEFAULT_COMPOSE,
  G_NORMALIZE_NFC = G_NORMALIZE_DEFAULT_COMPOSE,
  G_NORMALIZE_ALL,
  G_NORMALIZE_NFKD = G_NORMALIZE_ALL,
  G_NORMALIZE_ALL_COMPOSE,
  G_NORMALIZE_NFKC = G_NORMALIZE_ALL_COMPOSE
} GNormalizeMode;
```

Defines how a Unicode string is transformed in a canonical form, standardizing such issues as whether a character with an accent is represented as a base character and combining accent or as a single precomposed character. Unicode strings should generally be normalized before comparing them.

G_NORMALIZE_DEFAULT	standardize differences that do not affect the text content, such as the above-mentioned accent representation.
G_NORMALIZE_NFD	another name for G_NORMALIZE_DEFAULT.
G_NORMALIZE_DEFAULT_COMPOSE	like G_NORMALIZE_DEFAULT, but with composed forms rather than a maximally decomposed form.
G_NORMALIZE_NFC	another name for G_NORMALIZE_DEFAULT_COMPOSE.
G_NORMALIZE_ALL	beyond G_NORMALIZE_DEFAULT also standardize the "compatibility" characters in Unicode, such as SUPERSCRIPT THREE to the standard forms (in this case DIGIT THREE). Formatting information may be lost but for most text operations such characters should be considered the same.
G_NORMALIZE_NFKD	another name for G_NORMALIZE_ALL.
G_NORMALIZE_ALL_COMPOSE	like G_NORMALIZE_ALL, but with composed forms rather than a maximally decomposed form.

G_NORMALIZE_NFKC	another name for G_NORMALIZE_ALL_COMPOSE.

---

g_utf8_collate ()

```
gint        g_utf8_collate                  (const gchar *str1,
                                             const gchar *str2);
```

Compares two strings for ordering using the linguistically correct rules for the current locale. When sorting a large number of strings, it will be significantly faster to obtain collation keys with g_utf8_collate_key() and compare the keys with strcmp() when sorting instead of sorting the original strings.

str1 : a UTF-8 encoded string
str2 : a UTF-8 encoded string
Returns : -1 if *str1* compares before *str2*, 0 if they compare equal, 1 if *str1* compares after *str2*.

---

g_utf8_collate_key ()

```
gchar*      g_utf8_collate_key              (const gchar *str,
                                             gssize len);
```

Converts a string into a collation key that can be compared with other collation keys using strcmp (). The results of comparing the collation keys of two strings with strcmp() will always be the same as comparing the two original keys with g_utf8_collate().

str : a UTF-8 encoded string.
len : length of *str*, in bytes, or -1 if *str* is nul-terminated.
Returns : a newly allocated string. This string should be freed with g_free() when you are done with it.

---

g_utf8_to_utf16 ()

```
gunichar2*  g_utf8_to_utf16                 (const gchar *str,
                                             glong len,
                                             glong *items_read,
                                             glong *items_written,
                                             GError **error);
```

Convert a string from UTF-8 to UTF-16. A 0 word will be added to the result after the converted text.

str : a UTF-8 encoded string
len : the maximum length of *str* to use. If *len* < 0, then the string is nul-terminated.

items_read : location to store number of bytes read, or NULL. If NULL, then G_CONVERT_ERROR_PARTIAL_INPUT will be returned in case *str* contains a trailing partial character. If an error occurs then the index of the invalid input is stored here.

items_written : location to store number of words written, or NULL. The value stored here does not include the trailing 0 word.

error : location to store the error occuring, or NULL to ignore errors. Any of the errors in GConvertError other than G_CONVERT_ERROR_NO_CONVERSION may occur.

Returns : a pointer to a newly allocated UTF-16 string. This value must be freed with g_free(). If an error occurs, NULL will be returned and *error* set.

---

g_utf8_to_ucs4 ()

```
gunichar*   g_utf8_to_ucs4                  (const gchar *str,
                                             glong len,
                                             glong *items_read,
                                             glong *items_written,
                                             GError **error);
```

Convert a string from UTF-8 to a 32-bit fixed width representation as UCS-4. A trailing 0 will be added to the string after the converted text.

str : a UTF-8 encoded string

len : the maximum length of *str* to use. If *len* < 0, then the string is nul-terminated.

items_read : location to store number of bytes read, or NULL. If NULL, then G_CONVERT_ERROR_PARTIAL_INPUT will be returned in case *str* contains a trailing partial character. If an error occurs then the index of the invalid input is stored here.

items_written : location to store number of characters written or NULL. The value here stored does not include the trailing 0 character.

error : location to store the error occuring, or NULL to ignore errors. Any of the errors in GConvertError other than G_CONVERT_ERROR_NO_CONVERSION may occur.

Returns : a pointer to a newly allocated UCS-4 string. This value must be freed with g_free(). If an error occurs, NULL will be returned and *error* set.

---

g_utf8_to_ucs4_fast ()

```
gunichar*   g_utf8_to_ucs4_fast             (const gchar *str,
                                             glong len,
                                             glong *items_written);
```

Convert a string from UTF-8 to a 32-bit fixed width representation as UCS-4, assuming valid UTF-8 input. This function is roughly twice as fast as g_utf8_to_ucs4() but does no error checking on the input.

str : a UTF-8 encoded string

len : the maximum length of *str* to use. If *len* < 0, then the string is nul-terminated.

items_written : location to store the number of characters in the result, or NULL.

Returns : a pointer to a newly allocated UCS-4 string. This value must be freed with g_free().

---

g_utf16_to_ucs4 ()

```
gunichar*   g_utf16_to_ucs4                 (const gunichar2 *str,
                                             glong len,
                                             glong *items_read,
                                             glong *items_written,
                                             GError **error);
```

Convert a string from UTF-16 to UCS-4. The result will be terminated with a 0 character.

str : a UTF-16 encoded string

len : the maximum length of *str* to use. If *len* < 0, then the string is terminated with a 0 character.

items_read : location to store number of words read, or NULL. If NULL, then G_CONVERT_ERROR_PARTIAL_INPUT will be returned in case *str* contains a trailing partial character. If an error occurs then the index of the invalid input is stored here.

items_written : location to store number of characters written, or NULL. The value stored here does not include the trailing 0 character.

error : location to store the error occuring, or NULL to ignore errors. Any of the errors in GConvertError other than G_CONVERT_ERROR_NO_CONVERSION may occur.

Returns : a pointer to a newly allocated UCS-4 string. This value must be freed with g_free(). If an error occurs, NULL will be returned and *error* set.

---

g_utf16_to_utf8 ()

```
gchar*      g_utf16_to_utf8                 (const gunichar2 *str,
                                             glong len,
                                             glong *items_read,
                                             glong *items_written,
                                             GError **error);
```

Convert a string from UTF-16 to UTF-8. The result will be terminated with a 0 byte.

Note that the input is expected to be already in native endianness, an initial byte-order-mark character is not handled specially. g_convert() can be used to convert a byte buffer of UTF-16 data of ambiguous endianess.

str : a UTF-16 encoded string

len : the maximum length of *str* to use. If *len* < 0, then the string is terminated with a 0 character.

items_read : location to store number of words read, or NULL. If NULL, then G_CONVERT_ERROR_PARTIAL_INPUT will be returned in case *str* contains a trailing partial character. If an error occurs then the index of the invalid input is stored here.

items_written : location to store number of bytes written, or NULL. The value stored here does not include the trailing 0 byte.

error : location to store the error occuring, or NULL to ignore errors. Any of the errors in GConvertError other than G_CONVERT_ERROR_NO_CONVERSION may occur.

Returns : a pointer to a newly allocated UTF-8 string. This value must be freed with g_free(). If an error occurs, NULL will be returned and *error* set.

---

g_ucs4_to_utf16 ()

```
gunichar2*  g_ucs4_to_utf16                 (const gunichar *str,
                                             glong len,
                                             glong *items_read,
                                             glong *items_written,
                                             GError **error);
```

Convert a string from UCS-4 to UTF-16. A 0 word will be added to the result after the converted text.

str : a UCS-4 encoded string

len : the maximum length of *str* to use. If *len* < 0, then the string is terminated with a 0 character.

items_read : location to store number of bytes read, or NULL. If an error occurs then the index of the invalid input is stored here.

items_written : location to store number of words written, or NULL. The value stored here does not include the trailing 0 word.

error : location to store the error occuring, or NULL to ignore errors. Any of the errors in GConvertError other than G_CONVERT_ERROR_NO_CONVERSION may occur.

Returns : a pointer to a newly allocated UTF-16 string. This value must be freed with g_free(). If an error occurs, NULL will be returned and *error* set.

---

g_ucs4_to_utf8 ()

```
gchar*      g_ucs4_to_utf8                  (const gunichar *str,
                                             glong len,
                                             glong *items_read,
                                             glong *items_written,
                                             GError **error);
```

Convert a string from a 32-bit fixed width representation as UCS-4. to UTF-8. The result will be terminated with a 0 byte.

str : a UCS-4 encoded string

len : the maximum length of *str* to use. If *len* < 0, then the string is terminated with a 0 character.

items_read : location to store number of characters read read, or NULL.

items_written : location to store number of bytes written or NULL. The value here stored does not include the trailing 0 byte.

error : location to store the error occuring, or NULL to ignore errors. Any of the errors in GConvertError other than G_CONVERT_ERROR_NO_CONVERSION may occur.

Returns : a pointer to a newly allocated UTF-8 string. This value must be freed with g_free(). If an error occurs, NULL will be returned and *error* set.

---

g_unichar_to_utf8 ()

```
gint        g_unichar_to_utf8               (gunichar c,
                                             gchar *outbuf);
```

Converts a single character to UTF-8.

c : a ISO10646 character code

outbuf : output buffer, must have at least 6 bytes of space. If NULL, the length will be computed and returned and nothing will be written to *outbuf*.

Returns : number of bytes written

See Also

g_locale_to_utf8(), g_locale_from_utf8()	Convenience functions for converting between UTF-8 and the locale encoding.

<< Character Set Conversion **Internationalization >>**

Internationalization

Internationalization — gettext support macros.

Synopsis

```
#include <glib.h>
#include <glib/gi18n.h>


#define     _                                   (String)
#define     Q_                                  (String)
#define     N_                                  (String)
G_CONST_RETURN gchar* g_strip_context       (const gchar *msgid,
                                             const gchar *msgval);

G_CONST_RETURN gchar* G_CONST_RETURN * g_get_language_names
                                            (void);
```

Description

GLib doesn't force any particular localization method upon its users. But since GLib itself is localized using the `gettext()` mechanism, it seems natural to offer the de-facto standard `gettext()` support macros in an easy-to-use form.

In order to use these macros in an application, you must include `glib/gi18n.h`. For use in a library, must include `glib/gi18n-lib.h` *after* defining the GETTEXT_PACKAGE macro suitably for your library:

```
#define GETTEXT_PACKAGE "gtk20"
#include <glib/gi18n-lib.h>
```

Details

_()

```
#define     _(String)
```

Marks a string for translation, gets replaced with the translated string at runtime.

`String` : the string to be translated

Since 2.4

Q_()

```
#define     Q_(String)
```

Like `_()`, but applies `g_strip_context()` to the translation. This has the advantage that the string can be adorned with a prefix to guarantee uniqueness and provide context to the translator.

One use case given in the gettext manual is GUI translation, where one could e.g. disambiguate two "Open" menu entries as "File|Open" and "Printer|Open". Another use case is the string "Russian" which may have to be translated differently depending on whether it's the name of a character set or a language. This could be solved by using "charset|Russian" and "language|Russian".

`String` : the string to be translated, with a '|'-separated prefix which must not be translated

Since 2.4

N_()

```
#define     N_(String)
```

Marks a string for translation, gets replaced with the untranslated string at runtime. This is useful in situations where the translated strings can't be directly used, e.g. in string array initializers.

```
{
  static const char *messages[] = {
    N_("some very meaningful message"),
    N_("and another one")
  };
  const char *string;
  ...
  string
    = index > 1 ? _("a default message") : gettext (messages[index]);

  fputs (string);
  ...
}
```

`String` : the string to be translated

Since 2.4

---

g_strip_context ()

```
G_CONST_RETURN gchar* g_strip_context       (const gchar *msgid,
                                             const gchar *msgval);
```

An auxiliary function for `gettext()` support (see `Q_()`).

msgid : a string

msgval : another string

Returns : *msgval*, unless *msgval* is identical to *msgid* and contains a '|' character, in which case a pointer to the substring of msgid after the first '|' character is returned.

Since 2.4

---

g_get_language_names ()

```
G_CONST_RETURN gchar* G_CONST_RETURN * g_get_language_names
                                            (void);
```

Computes a list of applicable locale names, which can be used to e.g. construct locale-dependent filenames or search paths. The returned list is sorted from most desirable to least desirable and always contains the default locale "C".

For example, if LANGUAGE=de:en_US, then the returned list is "de", "en_US", "en", "C".

This function consults the environment variables `LANGUAGE`, `LC_ALL`, `LC_MESSAGES` and `LANG` to find the list of locales specified by the user.

Returns : a `NULL`-terminated array of strings owned by GLib that must not be modified or freed.

Since 2.6

See Also

The gettext manual.

<< Unicode Manipulation **Date and Time Functions >>**

Date and Time Functions

Date and Time Functions — calendrical calculations and miscellaneous time stuff.

Synopsis

```
#include <glib.h>


#define     G_USEC_PER_SEC
            GTimeVal;
void        g_get_current_time              (GTimeVal *result);
void        g_usleep                        (gulong microseconds);
void        g_time_val_add                  (GTimeVal *time_,
                                             glong microseconds);

            GDate;
typedef     GTime;
enum        GDateDMY;
typedef     GDateDay;
enum        GDateMonth;
typedef     GDateYear;
enum        GDateWeekday;

#define     G_DATE_BAD_DAY
#define     G_DATE_BAD_JULIAN
#define     G_DATE_BAD_YEAR

GDate*      g_date_new                      (void);
GDate*      g_date_new_dmy                  (GDateDay day,
                                             GDateMonth month,
                                             GDateYear year);
GDate*      g_date_new_julian               (guint32 julian_day);
void        g_date_clear                    (GDate *date,
                                             guint n_dates);
void        g_date_free                     (GDate *date);

void        g_date_set_day                  (GDate *date,
                                             GDateDay day);
void        g_date_set_month                (GDate *date,
                                             GDateMonth month);
void        g_date_set_year                 (GDate *date,
                                             GDateYear year);
void        g_date_set_dmy                  (GDate *date,
                                             GDateDay day,
                                             GDateMonth month,
                                             GDateYear y);
void        g_date_set_julian               (GDate *date,
                                             guint32 julian_date);
void        g_date_set_time                 (GDate *date,
                                             GTime time_);
void        g_date_set_parse                (GDate *date,
                                             const gchar *str);

void        g_date_add_days                 (GDate *date,
```



```
                                             guint n_days);
void        g_date_subtract_days            (GDate *date,
                                             guint n_days);
void        g_date_add_months               (GDate *date,
                                             guint n_months);
void        g_date_subtract_months          (GDate *date,
                                             guint n_months);
void        g_date_add_years                (GDate *date,
                                             guint n_years);
void        g_date_subtract_years           (GDate *date,
                                             guint n_years);
gint        g_date_days_between             (const GDate *date1,
                                             const GDate *date2);
gint        g_date_compare                  (const GDate *lhs,
                                             const GDate *rhs);
void        g_date_clamp                    (GDate *date,
                                             const GDate *min_date,
                                             const GDate *max_date);
void        g_date_order                    (GDate *date1,
                                             GDate *date2);

GDateDay    g_date_get_day                  (const GDate *date);
GDateMonth  g_date_get_month                (const GDate *date);
GDateYear   g_date_get_year                 (const GDate *date);
guint32     g_date_get_julian               (const GDate *date);
GDateWeekday g_date_get_weekday             (const GDate *date);
guint       g_date_get_day_of_year          (const GDate *date);

guint8      g_date_get_days_in_month        (GDateMonth month,
                                             GDateYear year);
gboolean    g_date_is_first_of_month        (const GDate *date);
gboolean    g_date_is_last_of_month         (const GDate *date);
gboolean    g_date_is_leap_year             (GDateYear year);
guint       g_date_get_monday_week_of_year  (const GDate *date);
guint8      g_date_get_monday_weeks_in_year (GDateYear year);
guint       g_date_get_sunday_week_of_year  (const GDate *date);
guint8      g_date_get_sunday_weeks_in_year (GDateYear year);
guint       g_date_get_iso8601_week_of_year (const GDate *date);

gsize       g_date_strftime                 (gchar *s,
                                             gsize slen,
                                             const gchar *format,
                                             const GDate *date);
void        g_date_to_struct_tm             (const GDate *date,
                                             struct tm *tm);

gboolean    g_date_valid                    (const GDate *date);
gboolean    g_date_valid_day                (GDateDay day);
gboolean    g_date_valid_month              (GDateMonth month);
gboolean    g_date_valid_year               (GDateYear year);
gboolean    g_date_valid_dmy                (GDateDay day,
                                             GDateMonth month,
                                             GDateYear year);
gboolean    g_date_valid_julian             (guint32 julian_date);
gboolean    g_date_valid_weekday            (GDateWeekday weekday);
```

Description

The GDate data structure represents a day between January 1, Year 1, and sometime a few thousand years in the future (right now it will go to the year 65535 or so, but `g_date_set_parse()` only parses up to the year 8000 or so - just count on "a few thousand"). GDate is meant to represent

everyday dates, not astronomical dates or historical dates or ISO timestamps or the like. It extrapolates the current Gregorian calendar forward and backward in time; there is no attempt to change the calendar to match time periods or locations. GDate does not store time information; it represents a *day*.

The GDate implementation has several nice features; it is only a 64-bit struct, so storing large numbers of dates is very efficient. It can keep both a Julian and day-month-year representation of the date, since some calculations are much easier with one representation or the other. A Julian representation is simply a count of days since some fixed day in the past; for GDate the fixed day is January 1, 1 AD. ("Julian" dates in the GDate API aren't really Julian dates in the technical sense; technically, Julian dates count from the start of the Julian period, Jan 1, 4713 BC).

GDate is simple to use. First you need a "blank" date; you can get a dynamically allocated date from `g_date_new()`, or you can declare an automatic variable or array and initialize it to a sane state by calling `g_date_clear()`. A cleared date is sane; it's safe to call `g_date_set_dmy()` and the other mutator functions to initialize the value of a cleared date. However, a cleared date is initially *invalid*, meaning that it doesn't represent a day that exists. It is undefined to call any of the date calculation routines on an invalid date. If you obtain a date from a user or other unpredictable source, you should check its validity with the `g_date_valid()` predicate. `g_date_valid()` is also used to check for errors with `g_date_set_parse()` and other functions that can fail. Dates can be invalidated by calling `g_date_clear()` again.

It is very important to use the API to access the GDate struct. Often only the day-month-year or only the Julian representation is valid. Sometimes neither is valid. Use the API.

GLib doesn't contain any time-manipulation functions; however, there is a GTime typedef which is equivalent to time_t, and a GTimeVal struct which represents a more precise time (with microseconds). You can request the current time as a GTimeVal with `g_get_current_time()`.

Details

G_USEC_PER_SEC

```
#define G_USEC_PER_SEC 1000000
```

Number of microseconds in one second (1 million). This macro is provided for code readability.

GTimeVal

```
typedef struct {
  glong tv_sec;
  glong tv_usec;
} GTimeVal;
```

Represents a precise time, with seconds and microseconds. Same as the struct timeval returned by the `gettimeofday()` UNIX call.

glong *tv_sec*; seconds.

glong *tv_usec*; microseconds.

---

g_get_current_time ()

```
void        g_get_current_time              (GTimeVal *result);
```

Equivalent to the UNIX `gettimeofday()` function, but portable.

result : GTimeVal structure in which to store current time.

---

g_usleep ()

```
void        g_usleep                        (gulong microseconds);
```

Pauses the current thread for the given number of microseconds. There are 1 million microseconds per second (represented by the G_USEC_PER_SEC macro). `g_usleep()` may have limited precision, depending on hardware and operating system; don't rely on the exact length of the sleep.

microseconds : number of microseconds to pause.

---

g_time_val_add ()

```
void        g_time_val_add                  (GTimeVal *time_,
                                             glong microseconds);
```

Adds the given number of microseconds to *time_*. *microseconds* can also be negative to decrease the value of *time_*.

*time_* : a GTimeVal

microseconds : number of microseconds to add to *time*

GDate

```
typedef struct {
  guint julian_days : 32; /* julian days representation - we use a
                           *  bitfield hoping that 64 bit platforms
                           *  will pack this whole struct in one big
                           *  int
                           */

  guint julian : 1;    /* julian is valid */
  guint dmy    : 1;    /* dmy is valid */

  /* DMY representation */
  guint day    : 6;
  guint month  : 4;
  guint year   : 16;
```

```
} GDate;
```

Represents a day between January 1, Year 1 and a few thousand years in the future. None of its members should be accessed directly. If the GDate is obtained from `g_date_new()`, it will be safe to mutate but invalid and thus not safe for calendrical computations. If it's declared on the stack, it will contain garbage so must be initialized with `g_date_clear()`. `g_date_clear()` makes the date invalid but sane. An invalid date doesn't represent a day, it's "empty." A date becomes valid after you set it to a Julian day or you set a day, month, and year.

GTime

```
typedef gint32  GTime;
```

Simply a replacement for time_t. Unrelated to GTimer.

---

enum GDateDMY

```
typedef enum
{
  G_DATE_DAY   = 0,
  G_DATE_MONTH = 1,
  G_DATE_YEAR  = 2
} GDateDMY;
```

This enumeration isn't used in the API, but may be useful if you need to mark a number as a day, month, or year.

GDateDay

```
typedef guint8  GDateDay;   /* day of the month */
```

Integer representing a day of the month; between 1 and 31. G_DATE_BAD_DAY represents an invalid day of the month.

---

enum GDateMonth

```
typedef enum
{
  G_DATE_BAD_MONTH = 0,
  G_DATE_JANUARY   = 1,
  G_DATE_FEBRUARY  = 2,
  G_DATE_MARCH     = 3,
  G_DATE_APRIL     = 4,
  G_DATE_MAY       = 5,
```



```
  G_DATE_JUNE      = 6,
  G_DATE_JULY      = 7,
  G_DATE_AUGUST    = 8,
  G_DATE_SEPTEMBER = 9,
  G_DATE_OCTOBER   = 10,
  G_DATE_NOVEMBER  = 11,
  G_DATE_DECEMBER  = 12
} GDateMonth;
```

Enumeration representing a month; values are G_DATE_JANUARY, G_DATE_FEBRUARY, etc. G_DATE_BAD_MONTH is the invalid value.

`G_DATE_BAD_MONTH`	invalid value.
`G_DATE_JANUARY`	January.
`G_DATE_FEBRUARY`	February.
`G_DATE_MARCH`	March.
`G_DATE_APRIL`	April.
`G_DATE_MAY`	May.
`G_DATE_JUNE`	June.
`G_DATE_JULY`	July.
`G_DATE_AUGUST`	August.
`G_DATE_SEPTEMBER`	September.
`G_DATE_OCTOBER`	October.
`G_DATE_NOVEMBER`	November.
`G_DATE_DECEMBER`	December.

GDateYear

```
typedef guint16 GDateYear;
```

Integer representing a year; G_DATE_BAD_YEAR is the invalid value. The year must be 1 or higher; negative (BC) years are not allowed. The year is represented with four digits.

---

enum GDateWeekday

```
typedef enum
{
  G_DATE_BAD_WEEKDAY  = 0,
  G_DATE_MONDAY       = 1,
  G_DATE_TUESDAY      = 2,
  G_DATE_WEDNESDAY    = 3,
  G_DATE_THURSDAY     = 4,
  G_DATE_FRIDAY       = 5,
  G_DATE_SATURDAY     = 6,
  G_DATE_SUNDAY       = 7
} GDateWeekday;
```

Enumeration representing a day of the week; G_DATE_MONDAY, G_DATE_TUESDAY, etc.

G_DATE_BAD_WEEKDAY is an invalid weekday.

G_DATE_BAD_WEEKDAY	invalid value.
G_DATE_MONDAY	Monday.
G_DATE_TUESDAY	Tuesday.
G_DATE_WEDNESDAY	Wednesday.
G_DATE_THURSDAY	Thursday.
G_DATE_FRIDAY	Friday.
G_DATE_SATURDAY	Saturday.
G_DATE_SUNDAY	Sunday.

G_DATE_BAD_DAY

```
#define G_DATE_BAD_DAY    0U
```

Represents an invalid GDateDay.

G_DATE_BAD_JULIAN

```
#define G_DATE_BAD_JULIAN 0U
```

Represents an invalid Julian day number.

G_DATE_BAD_YEAR

```
#define G_DATE_BAD_YEAR   0U
```

Represents an invalid year.

---

g_date_new ()

```
GDate*      g_date_new                      (void);
```

Allocates a GDate and initializes it to a sane state. The new date will be cleared (as if you'd called g_date_clear()) but invalid (it won't represent an existing day). Free the return value with g_date_free().

Returns : a newly-allocated GDate.

---

g_date_new_dmy ()

```
GDate*      g_date_new_dmy                  (GDateDay day,
                                             GDateMonth month,
                                             GDateYear year);
```

Like g_date_new(), but also sets the value of the date. Assuming the day-month-year triplet you pass in represents an existing day, the returned date will be valid.

day : day of the month.
month : month of the year.
year : year
Returns : a newly-allocated GDate initialized with *day*, *month*, and *year*.

---

g_date_new_julian ()

```
GDate*      g_date_new_julian               (guint32 julian_day);
```

Like g_date_new(), but also sets the value of the date. Assuming the Julian day number you pass in is valid (greater than 0, less than an unreasonably large number), the returned date will be valid.

julian_day : days since January 1, Year 1.
Returns : a newly-allocated GDate initialized with *julian_day*.

---

g_date_clear ()

```
void        g_date_clear                    (GDate *date,
                                             guint n_dates);
```

Initializes one or more GDate structs to a sane but invalid state. The cleared dates will not represent an existing date, but will not contain garbage. Useful to init a date declared on the stack. Validity can be tested with g_date_valid().

date : pointer to one or more dates to clear.
n_dates : number of dates to clear.

---

g_date_free ()

```
void        g_date_free                     (GDate *date);
```

Frees a GDate returned from g_date_new().

date : a GDate.

g_date_set_day ()

```
void        g_date_set_day                  (GDate *date,
                                             GDateDay day);
```

Sets the day of the month for a GDate. If the resulting day-month-year triplet is invalid, the date will be invalid.

date : a GDate.
day : day to set.

g_date_set_month ()

```
void        g_date_set_month                (GDate *date,
                                             GDateMonth month);
```

Sets the month of the year for a GDate. If the resulting day-month-year triplet is invalid, the date will be invalid.

date : a GDate.
month : month to set.

g_date_set_year ()

```
void        g_date_set_year                 (GDate *date,
                                             GDateYear year);
```

Sets the year for a GDate. If the resulting day-month-year triplet is invalid, the date will be invalid.

date : a GDate.
year : year to set.

g_date_set_dmy ()

```
void        g_date_set_dmy                  (GDate *date,
                                             GDateDay day,
                                             GDateMonth month,
                                             GDateYear y);
```

Sets the value of a GDate from a day, month, and year. The day-month-year triplet must be valid; if you aren't sure it is, call `g_date_valid_dmy()` to check before you set it.

date : a GDate.

day : day.
month : month.
y : year.

g_date_set_julian ()

```
void        g_date_set_julian               (GDate *date,
                                             guint32 julian_date);
```

Sets the value of a GDate from a Julian day number.

date : a GDate.
julian_date : Julian day number (days since January 1, Year 1).

g_date_set_time ()

```
void        g_date_set_time                 (GDate *date,
                                             GTime time_);
```

Sets the value of a date from a GTime (time_t) value. To set the value of a date to the current day, you could write:

```
g_date_set_time (date, time (NULL));
```

date : a GDate.
*time_* : GTime value to set.

g_date_set_parse ()

```
void        g_date_set_parse                (GDate *date,
                                             const gchar *str);
```

Parses a user-inputted string *str*, and try to figure out what date it represents, taking the current locale into account. If the string is successfully parsed, the date will be valid after the call. Otherwise, it will be invalid. You should check using `g_date_valid()` to see whether the parsing succeeded.

This function is not appropriate for file formats and the like; it isn't very precise, and its exact behavior varies with the locale. It's intended to be a heuristic routine that guesses what the user means by a given string (and it does work pretty well in that capacity).

date : a GDate to fill in.
str : string to parse.

g_date_add_days ()

```
void        g_date_add_days                 (GDate *date,
                                             guint n_days);
```

Increments a date some number of days. To move forward by weeks, add weeks*7 days. The date must be valid.

date : a GDate to increment.
n_days : number of days to move the date forward.

g_date_subtract_days ()

```
void        g_date_subtract_days            (GDate *date,
                                             guint n_days);
```

Moves a date some number of days into the past. To move by weeks, just move by weeks*7 days. The date must be valid.

date : a GDate to decrement.
n_days : number of days to move.

g_date_add_months ()

```
void        g_date_add_months               (GDate *date,
                                             guint n_months);
```

Increments a date by some number of months. If the day of the month is greater than 28, this routine may change the day of the month (because the destination month may not have the current day in it). The date must be valid.

date : a GDate to increment.
n_months : number of months to move forward.

g_date_subtract_months ()

```
void        g_date_subtract_months          (GDate *date,
                                             guint n_months);
```

Moves a date some number of months into the past. If the current day of the month doesn't exist in the destination month, the day of the month may change. The date must be valid.

date : a GDate to decrement.
n_months : number of months to move.

g_date_add_years ()

```
void        g_date_add_years                (GDate *date,
                                             guint n_years);
```

Increments a date by some number of years. If the date is February 29, and the destination year is not a leap year, the date will be changed to February 28. The date must be valid.

date : a GDate to increment.
n_years : number of years to move forward.

g_date_subtract_years ()

```
void        g_date_subtract_years           (GDate *date,
                                             guint n_years);
```

Moves a date some number of years into the past. If the current day doesn't exist in the destination year (i.e. it's February 29 and you move to a non-leap-year) then the day is changed to February 29. The date must be valid.

date : a GDate to decrement.
n_years : number of years to move.

g_date_days_between ()

```
gint        g_date_days_between             (const GDate *date1,
                                             const GDate *date2);
```

Computes the number of days between two dates. If *date2* is prior to *date1*, the returned value is negative. Both dates must be valid.

date1 : the first date.
date2 : the second date.
Returns : the number of days between *date1* and *date2*.

g_date_compare ()

```
gint        g_date_compare                  (const GDate *lhs,
                                             const GDate *rhs);
```

qsort()-style comparsion function for dates. Both dates must be valid.

lhs : first date to compare.
rhs : second date to compare.
Returns : 0 for equal, less than zero if *lhs* is less than *rhs*, greater than zero if *lhs* is greater than *rhs*.

---

g_date_clamp ()

```
void        g_date_clamp                    (GDate *date,
                                             const GDate *min_date,
                                             const GDate *max_date);
```

If *date* is prior to *min_date*, sets *date* equal to *min_date*. If *date* falls after *max_date*, sets *date* equal to *max_date*. Either *min_date* and *max_date* may be NULL. All non-NULL dates must be valid.

date : a GDate to clamp.
min_date : minimum accepted value for *date*.
max_date : maximum accepted value for *date*.

---

g_date_order ()

```
void        g_date_order                    (GDate *date1,
                                             GDate *date2);
```

Checks if *date1* is less than or equal to *date2*, and swap the values if this is not the case.

date1 : the first date.
date2 : the second date.

---

g_date_get_day ()

```
GDateDay    g_date_get_day                  (const GDate *date);
```

Returns the day of the month. The date must be valid.

date : a GDate to extract the day of the month from.
Returns : day of the month.

---

g_date_get_month ()

```
GDateMonth  g_date_get_month                (const GDate *date);
```


Returns the month of the year. The date must be valid.

date : a GDate to get the month from.
Returns : month of the year as a GDateMonth.

---

g_date_get_year ()

```
GDateYear   g_date_get_year                 (const GDate *date);
```

Returns the year of a GDate. The date must be valid.

date : a GDate.
Returns : year in which the date falls.

---

g_date_get_julian ()

```
guint32     g_date_get_julian               (const GDate *date);
```

Returns the Julian day or "serial number" of the GDate. The Julian day is simply the number of days since January 1, Year 1; i.e., January 1, Year 1 is Julian day 1; January 2, Year 1 is Julian day 2, etc. The date must be valid.

date : a GDate to extract the Julian day from.
Returns : Julian day.

---

g_date_get_weekday ()

```
GDateWeekday g_date_get_weekday             (const GDate *date);
```

Returns the day of the week for a GDate. The date must be valid.

date : a GDate.
Returns : day of the week as a GDateWeekday.

---

g_date_get_day_of_year ()

```
guint       g_date_get_day_of_year          (const GDate *date);
```

Returns the day of the year, where Jan 1 is the first day of the year. The date must be valid.

date : a GDate to extract day of year from.
Returns : day of the year.

---

g_date_get_days_in_month ()

```
guint8      g_date_get_days_in_month        (GDateMonth month,
                                             GDateYear year);
```

Returns the number of days in a month, taking leap years into account.

month : month.
year : year.
Returns : number of days in *month* during the *year*.

---

g_date_is_first_of_month ()

```
gboolean    g_date_is_first_of_month        (const GDate *date);
```

Returns TRUE if the date is on the first of a month. The date must be valid.

date : a GDate to check.
Returns : TRUE if the date is the first of the month.

---

g_date_is_last_of_month ()

```
gboolean    g_date_is_last_of_month         (const GDate *date);
```

Returns TRUE if the date is the last day of the month. The date must be valid.

date : a GDate to check.
Returns : TRUE if the date is the last day of the month.

---

g_date_is_leap_year ()

```
gboolean    g_date_is_leap_year             (GDateYear year);
```

Returns TRUE if the year is a leap year.

year : year to check.
Returns : TRUE if the year is a leap year.

---

g_date_get_monday_week_of_year ()

```
guint       g_date_get_monday_week_of_year  (const GDate *date);
```

Returns the week of the year, where weeks are understood to start on Monday. If the date is before the first Monday of the year, return 0. The date must be valid.

date : a GDate.
Returns : week of the year.

---

g_date_get_monday_weeks_in_year ()

```
guint8      g_date_get_monday_weeks_in_year (GDateYear year);
```

Returns the number of weeks in the year, where weeks are taken to start on Monday. Will be 52 or 53. The date must be valid. (Years always have 52 7-day periods, plus 1 or 2 extra days depending on whether it's a leap year. This function is basically telling you how many Mondays are in the year, i.e. there are 53 Mondays if one of the extra days happens to be a Monday.)

year : a year.
Returns : number of Mondays in the year.

---

g_date_get_sunday_week_of_year ()

```
guint       g_date_get_sunday_week_of_year  (const GDate *date);
```

Returns the week of the year during which this date falls, if weeks are understood to being on Sunday. The date must be valid. Can return 0 if the day is before the first Sunday of the year.

date : a GDate.
Returns : week number.

---

g_date_get_sunday_weeks_in_year ()

```
guint8      g_date_get_sunday_weeks_in_year (GDateYear year);
```

Returns the number of weeks in the year, where weeks are taken to start on Sunday. Will be 52 or 53. The date must be valid. (Years always have 52 7-day periods, plus 1 or 2 extra days depending on whether it's a leap year. This function is basically telling you how many Sundays are in the year, i.e. there are 53 Sundays if one of the extra days happens to be a Sunday.)

year : year to count weeks in.
Returns : number of weeks.

g_date_get_iso8601_week_of_year ()

```
guint       g_date_get_iso8601_week_of_year (const GDate *date);
```

Returns the week of the year, where weeks are interpreted according to ISO 8601.

date : a valid GDate
Returns : ISO 8601 week number of the year.

Since 2.6

g_date_strftime ()

```
gsize       g_date_strftime                 (gchar *s,
                                             gsize slen,
                                             const gchar *format,
                                             const GDate *date);
```

Generates a printed representation of the date, in a locale-specific way. Works just like the standard C `strftime()` function, but only accepts date-related formats; time-related formats give undefined results. Date must be valid.

s : destination buffer.
slen : buffer size.
format : format string.
date : valid GDate.
Returns : number of characters written to the buffer, or 0 the buffer was too small.

g_date_to_struct_tm ()

```
void        g_date_to_struct_tm             (const GDate *date,
                                             struct tm *tm);
```

Fills in the date-related bits of a struct tm using the *date* value. Initializes the non-date parts with something sane but meaningless.

date : a GDate to set the struct tm from.
tm : struct tm to fill.

g_date_valid ()


```
gboolean    g_date_valid                    (const GDate *date);
```

Returns `TRUE` if the GDate represents an existing day. The date must not contain garbage; it should have been initialized with `g_date_clear()` if it wasn't allocated by one of the `g_date_new()` variants.

date : a GDate to check.
Returns : Whether the date is valid.

g_date_valid_day ()

```
gboolean    g_date_valid_day                (GDateDay day);
```

Returns `TRUE` if the day of the month is valid (a day is valid if it's between 1 and 31 inclusive).

day : day to check.
Returns : `TRUE` if the day is valid.

g_date_valid_month ()

```
gboolean    g_date_valid_month              (GDateMonth month);
```

Returns `TRUE` if the month value is valid. The 12 GDateMonth enumeration values are the only valid months.

month : month.
Returns : `TRUE` if the month is valid.

g_date_valid_year ()

```
gboolean    g_date_valid_year               (GDateYear year);
```

Returns `TRUE` if the year is valid. Any year greater than 0 is valid, though there is a 16-bit limit to what GDate will understand.

year : year.
Returns : `TRUE` if the year is valid.

g_date_valid_dmy ()

```
gboolean    g_date_valid_dmy                (GDateDay day,
                                             GDateMonth month,
```

```
                                        GDateYear year);
```

Returns TRUE if the day-month-year triplet forms a valid, existing day in the range of days GDate understands (Year 1 or later, no more than a few thousand years in the future).

day : day.
month : month.
year : year.
Returns : TRUE if the date is a valid one.

---

g_date_valid_julian ()

```
gboolean    g_date_valid_julian             (guint32 julian_date);
```

Returns TRUE if the Julian day is valid. Anything greater than zero is basically a valid Julian, though there is a 32-bit limit.

julian_date : Julian day to check.
Returns : TRUE if the Julian day is valid.

---

g_date_valid_weekday ()

```
gboolean    g_date_valid_weekday            (GDateWeekday weekday);
```

Returns TRUE if the weekday is valid. The 7 GDateWeekday enumeration values are the only valid weekdays.

weekday : weekday.
Returns : TRUE if the weekday is valid.

<< Internationalization **Random Numbers >>**

Random Numbers

Random Numbers — pseudo-random number generator.

Synopsis

```
#include <glib.h>


            GRand;
GRand*      g_rand_new_with_seed            (guint32 seed);
GRand*      g_rand_new_with_seed_array      (const guint32 *seed,
                                             guint seed_length);
GRand*      g_rand_new                      (void);
GRand*      g_rand_copy                     (GRand *rand_);
void        g_rand_free                     (GRand *rand_);
void        g_rand_set_seed                 (GRand *rand_,
                                             guint32 seed);
void        g_rand_set_seed_array           (GRand *rand_,
                                             const guint32 *seed,
                                             guint seed_length);
#define     g_rand_boolean                  (rand_)
guint32     g_rand_int                      (GRand *rand_);
gint32      g_rand_int_range                (GRand *rand_,
                                             gint32 begin,
                                             gint32 end);
gdouble     g_rand_double                   (GRand *rand_);
gdouble     g_rand_double_range             (GRand *rand_,
                                             gdouble begin,
                                             gdouble end);
void        g_random_set_seed               (guint32 seed);
#define     g_random_boolean                ()
guint32     g_random_int                    (void);
gint32      g_random_int_range              (gint32 begin,
                                             gint32 end);
gdouble     g_random_double                 (void);
gdouble     g_random_double_range           (gdouble begin,
                                             gdouble end);
```

Description

The following functions allow you to use a portable, fast and good pseudo-random number generator (PRNG). It uses the Mersenne Twister PRNG, which was originally developed by Makoto Matsumoto and Takuji Nishimura. Further information can be found at www.math.keio.ac.jp/~matumoto/emt.html.

If you just need a random number, you simply call the `g_random_*` functions, which will create a globally used GRand and use the according `g_rand_*` functions internally. Whenever you need a stream of reproducible random numbers, you better create a GRand yourself and use the `g_rand_*` functions directly, which will also be slightly faster. Initializing a GRand with a certain seed will produce exactly the same series of random numbers on all platforms. This can thus be used as a seed

for e.g. games.

The `g_rand*_range` functions will return high quality equally distributed random numbers, whereas for example the `(g_random_int()%max)` approach often doesn't yield equally distributed numbers.

GLib changed the seeding algorithm for the pseudo-random number generator Mersenne Twister, as used by GRand and GRandom. This was necessary, because some seeds would yield very bad pseudo-random streams. Also the pseudo-random integers generated by `g_rand*_int_range()` will have a slightly better equal distribution with the new version of GLib.

The original seeding and generation algorithms, as found in GLib 2.0.x, can be used instead of the new ones by setting the environment variable `G_RANDOM_VERSION` to the value of '2.0'. Use the GLib-2.0 algorithms only if you have sequences of numbers generated with Glib-2.0 that you need to reproduce exactly.

Details

GRand

```
typedef struct _GRand GRand;
```

The GRand struct is an opaque data structure. It should only be accessed through the `g_rand_*` functions.

---

g_rand_new_with_seed ()

```
GRand*      g_rand_new_with_seed            (guint32 seed);
```

Creates a new random number generator initialized with *seed*.

seed : a value to initialize the random number generator.
Returns : the new GRand.

---

g_rand_new_with_seed_array ()

```
GRand*      g_rand_new_with_seed_array      (const guint32 *seed,
                                             guint seed_length);
```

Creates a new random number generator initialized with *seed*.

seed : an array of seeds to initialize the random number generator.
seed_length : an array of seeds to initialize the random number generator.
Returns : the new GRand.

Since 2.4

g_rand_new ()

```
GRand*      g_rand_new                      (void);
```

Creates a new random number generator initialized with a seed taken either from `/dev/urandom` (if existing) or from the current time (as a fallback).

Returns : the new GRand.

g_rand_copy ()

```
GRand*      g_rand_copy                     (GRand *rand_);
```

Copies a GRand into a new one with the same exact state as before. This way you can take a snapshot of the random number generator for replaying later.

*rand_* : a GRand.
Returns : the new GRand.

Since 2.4

g_rand_free ()

```
void        g_rand_free                     (GRand *rand_);
```

Frees the memory allocated for the GRand.

*rand_* : a GRand.

g_rand_set_seed ()

```
void        g_rand_set_seed                 (GRand *rand_,
                                             guint32 seed);
```

Sets the seed for the random number generator GRand to *seed*.

*rand_* : a GRand.
seed : a value to reinitialize the random number generator.

g_rand_set_seed_array ()

```
void        g_rand_set_seed_array           (GRand *rand_,
                                             const guint32 *seed,
                                             guint seed_length);
```

Initializes the random number generator by an array of longs. Array can be of arbitrary size, though only the first 624 values are taken. This function is useful if you have many low entropy seeds, or if you require more then 32bits of actual entropy for your application.

*rand_* : a GRand.
seed : array to initialize with
seed_length : length of array

Since 2.4

g_rand_boolean()

```
#define     g_rand_boolean(rand_)
```

Returns a random gboolean from *rand_*. This corresponds to a unbiased coin toss.

*rand_* : a GRand.
Returns : a random gboolean.

g_rand_int ()

```
guint32     g_rand_int                      (GRand *rand_);
```

Returns the next random guint32 from *rand_* equally distributed over the range [0..2^32-1].

*rand_* : a GRand.
Returns : A random number.

g_rand_int_range ()

```
gint32      g_rand_int_range                (GRand *rand_,
                                             gint32 begin,
                                             gint32 end);
```

Returns the next random gint32 from *rand_* equally distributed over the range [*begin*..*end*-1].

*rand_* : a GRand.

begin : lower closed bound of the interval.
end : upper open bound of the interval.
Returns : A random number.

---

g_rand_double ()

```
gdouble     g_rand_double                   (GRand *rand_);
```

Returns the next random gdouble from *rand_* equally distributed over the range [0..1).

*rand_* : a GRand.
Returns : A random number.

---

g_rand_double_range ()

```
gdouble     g_rand_double_range             (GRand *rand_,
                                             gdouble begin,
                                             gdouble end);
```

Returns the next random gdouble from *rand_* equally distributed over the range [*begin*..*end*).

*rand_* : a GRand.
begin : lower closed bound of the interval.
end : upper open bound of the interval.
Returns : A random number.

---

g_random_set_seed ()

```
void        g_random_set_seed               (guint32 seed);
```

Sets the seed for the global random number generator, which is used by the `g_random_*` functions, to *seed*.

seed : a value to reinitialize the global random number generator.

---

g_random_boolean()

```
#define     g_random_boolean()
```

Returns a random gboolean. This corresponds to a unbiased coin toss.

Returns : a random gboolean.

---

g_random_int ()

```
guint32     g_random_int                    (void);
```

Return a random guint32 equally distributed over the range [0..2^32-1].

Returns : A random number.

---

g_random_int_range ()

```
gint32      g_random_int_range              (gint32 begin,
                                             gint32 end);
```

Returns a random gint32 equally distributed over the range [*begin*..*end*-1].

begin : lower closed bound of the interval.
end : upper open bound of the interval.
Returns : A random number.

---

g_random_double ()

```
gdouble     g_random_double                 (void);
```

Returns a random gdouble equally distributed over the range [0..1).

Returns : A random number.

---

g_random_double_range ()

```
gdouble     g_random_double_range           (gdouble begin,
                                             gdouble end);
```

Returns a random gdouble equally distributed over the range [*begin*..*end*).

begin : lower closed bound of the interval.
end : upper open bound of the interval.
Returns : A random number.

<< Date and Time Functions **Hook Functions >>**

Hook Functions

Hook Functions — support for manipulating lists of hook functions.

Synopsis

```
#include <glib.h>


            GHookList;
void        (*GHookFinalizeFunc)            (GHookList *hook_list,
                                             GHook *hook);
            GHook;
void        (*GHookFunc)                    (gpointer data);
gboolean    (*GHookCheckFunc)               (gpointer data);

void        g_hook_list_init                (GHookList *hook_list,
                                             guint hook_size);
void        g_hook_list_invoke              (GHookList *hook_list,
                                             gboolean may_recurse);
void        g_hook_list_invoke_check        (GHookList *hook_list,
                                             gboolean may_recurse);
void        g_hook_list_marshal             (GHookList *hook_list,
                                             gboolean may_recurse,
                                             GHookMarshaller marshaller,
                                             gpointer marshal_data);
void        (*GHookMarshaller)              (GHook *hook,
                                             gpointer marshal_data);
void        g_hook_list_marshal_check       (GHookList *hook_list,
                                             gboolean may_recurse,
                                             GHookCheckMarshaller marshaller,
                                             gpointer marshal_data);
gboolean    (*GHookCheckMarshaller)         (GHook *hook,
                                             gpointer marshal_data);
void        g_hook_list_clear               (GHookList *hook_list);

GHook*      g_hook_alloc                    (GHookList *hook_list);
#define     g_hook_append                   ( hook_list, hook )
void        g_hook_prepend                  (GHookList *hook_list,
                                             GHook *hook);
void        g_hook_insert_before            (GHookList *hook_list,
                                             GHook *sibling,
                                             GHook *hook);
void        g_hook_insert_sorted            (GHookList *hook_list,
                                             GHook *hook,
                                             GHookCompareFunc func);
gint        (*GHookCompareFunc)             (GHook *new_hook,
                                             GHook *sibling);
gint        g_hook_compare_ids              (GHook *new_hook,
                                             GHook *sibling);

GHook*      g_hook_get                      (GHookList *hook_list,
                                             gulong hook_id);
GHook*      g_hook_find                     (GHookList *hook_list,
                                             gboolean need_valids,
```



```
                                             GHookFindFunc func,
                                             gpointer data);
gboolean    (*GHookFindFunc)                (GHook *hook,
                                             gpointer data);
GHook*      g_hook_find_data                (GHookList *hook_list,
                                             gboolean need_valids,
                                             gpointer data);
GHook*      g_hook_find_func                (GHookList *hook_list,
                                             gboolean need_valids,
                                             gpointer func);
GHook*      g_hook_find_func_data           (GHookList *hook_list,
                                             gboolean need_valids,
                                             gpointer func,
                                             gpointer data);

GHook*      g_hook_first_valid              (GHookList *hook_list,
                                             gboolean may_be_in_call);
GHook*      g_hook_next_valid               (GHookList *hook_list,
                                             GHook *hook,
                                             gboolean may_be_in_call);
enum        GHookFlagMask;
#define     G_HOOK_FLAGS                    (hook)
#define     G_HOOK_FLAG_USER_SHIFT

#define     G_HOOK                          (hook)
#define     G_HOOK_IS_VALID                 (hook)
#define     G_HOOK_ACTIVE                   (hook)
#define     G_HOOK_IN_CALL                  (hook)
#define     G_HOOK_IS_UNLINKED              (hook)

GHook*      g_hook_ref                      (GHookList *hook_list,
                                             GHook *hook);
void        g_hook_unref                    (GHookList *hook_list,
                                             GHook *hook);
void        g_hook_free                     (GHookList *hook_list,
                                             GHook *hook);
gboolean    g_hook_destroy                  (GHookList *hook_list,
                                             gulong hook_id);
void        g_hook_destroy_link             (GHookList *hook_list,
                                             GHook *hook);
```

Description

The GHookList, GHook and their related functions provide support for lists of hook functions. Functions can be added and removed from the lists, and the list of hook functions can be invoked.

Details

GHookList

```
typedef struct {
  gulong            seq_id;
  guint             hook_size : 16;
  guint             is_setup : 1;
  GHook            *hooks;
  GMemChunk        *hook_memchunk;
  GHookFinalizeFunc finalize_hook;
  gpointer          dummy[2];
} GHookList;
```

The GHookList struct represents a list of hook functions.

gulong *seq_id*;	the next free GHook id.
guint *hook_size* : 16;	the size of the GHookList elements, in bytes.
guint *is_setup* : 1;	1 if the GHookList has been initialized.
GHook **hooks*;	the first GHook element in the list.
GMemChunk **hook_memchunk*;	the GMemChunk used for allocating the GHook elements.
GHookFinalizeFunc *finalize_hook*;	the function to call to finalize a GHook element. The default behaviour is to call the hooks `destroy` function.
gpointer *dummy*[2];	

GHookFinalizeFunc ()

```
void        (*GHookFinalizeFunc)            (GHookList *hook_list,
                                             GHook *hook);
```

Defines the type of function to be called when a hook in a list of hooks gets finalized.

hook_list : a GHookList.
hook : the hook in *hook_list* that gets finalized.

GHook

```
typedef struct {
  gpointer       data;
  GHook         *next;
  GHook         *prev;
  guint          ref_count;
  gulong         hook_id;
  guint          flags;
  gpointer       func;
  GDestroyNotify destroy;
} GHook;
```

The GHook struct represents a single hook function in a GHookList.

gpointer *data*;	data which is passed to func when this hook is invoked.
GHook **next*;	pointer to the next hook in the list.
GHook **prev*;	pointer to the previous hook in the list.
guint *ref_count*;	the reference count of this hook.
gulong *hook_id*;	the id of this hook, which is unique within its list.
guint *flags*;	flags which are set for this hook. See GHookFlagMask for predefined flags.
gpointer *func*;	the function to call when this hook is invoked. The possible signatures for this function are GHookFunc and GHookCheckFunc.
GDestroyNotify *destroy*;	the default `finalize_hook` function of a GHookList calls this member of the hook that is being finalized.

GHookFunc ()

```
void        (*GHookFunc)                    (gpointer data);
```

Defines the type of a hook function that can be invoked by `g_hook_list_invoke()`.

data : the data field of the GHook is passed to the hook function here.

GHookCheckFunc ()

```
gboolean    (*GHookCheckFunc)               (gpointer data);
```

Defines the type of a hook function that can be invoked by `g_hook_list_invoke_check()`.

data : the data field of the GHook is passed to the hook function here.
Returns : `FALSE` if the GHook should be destroyed.

---

g_hook_list_init ()

```
void        g_hook_list_init                (GHookList *hook_list,
                                             guint hook_size);
```

Initializes a GHookList. This must be called before the GHookList is used.

hook_list : a GHookList.
hook_size : the size of each element in the GHookList, typically `sizeof (GHook)`.

---

g_hook_list_invoke ()

```
void        g_hook_list_invoke              (GHookList *hook_list,
                                             gboolean may_recurse);
```

Calls all of the GHook functions in a GHookList.

hook_list : a GHookList.
may_recurse : `TRUE` if functions which are already running (e.g. in another thread) can be called. If set to `FALSE`, these are skipped.

---

g_hook_list_invoke_check ()

```
void        g_hook_list_invoke_check        (GHookList *hook_list,
                                             gboolean may_recurse);
```

Calls all of the GHook functions in a GHookList. Any function which returns TRUE is removed from the GHookList.

hook_list : a GHookList.
may_recurse : TRUE if functions which are already running (e.g. in another thread) can be called. If set to FALSE, these are skipped.

---

g_hook_list_marshal ()

```
void        g_hook_list_marshal             (GHookList *hook_list,
                                             gboolean may_recurse,
                                             GHookMarshaller marshaller,
                                             gpointer marshal_data);
```

Calls a function on each valid GHook.

hook_list : a GHookList.
may_recurse : TRUE if hooks which are currently running (e.g. in another thread) are considered valid. If set to FALSE, these are skipped.
marshaller : the function to call for each GHook.
marshal_data : data to pass to *marshaller*.

GHookMarshaller ()

```
void        (*GHookMarshaller)              (GHook *hook,
                                             gpointer marshal_data);
```

Defines the type of function used by g_hook_list_marshal().

hook : a GHook.
marshal_data : user data.

---

g_hook_list_marshal_check ()

```
void        g_hook_list_marshal_check       (GHookList *hook_list,
                                             gboolean may_recurse,
                                             GHookCheckMarshaller marshaller,
                                             gpointer marshal_data);
```

Calls a function on each valid GHook and destroys it if the function returns FALSE.

hook_list : a GHookList.
may_recurse : TRUE if hooks which are currently running (e.g. in another thread) are considered valid. If set to FALSE, these are skipped.
marshaller : the function to call for each GHook.
marshal_data : data to pass to *marshaller*.

GHookCheckMarshaller ()

```
gboolean    (*GHookCheckMarshaller)         (GHook *hook,
                                             gpointer marshal_data);
```

Defines the type of function used by g_hook_list_marshal_check().

hook : a GHook.
marshal_data : user data.
Returns : FALSE if *hook* should be destroyed.

---

g_hook_list_clear ()

```
void        g_hook_list_clear               (GHookList *hook_list);
```

Removes all the GHook elements from a GHookList.

hook_list : a GHookList.

---

g_hook_alloc ()

```
GHook*      g_hook_alloc                    (GHookList *hook_list);
```

Allocates space for a GHook and initializes it.

hook_list : a GHookList.
Returns : a new GHook.

---

g_hook_append()

```
#define     g_hook_append( hook_list, hook )
```

Appends a GHook onto the end of a GHookList.

hook_list : a GHookList.

hook : the GHook to add to the end of *hook_list*.

---

g_hook_prepend ()

```
void        g_hook_prepend                  (GHookList *hook_list,
                                             GHook *hook);
```

Prepends a GHook on the start of a GHookList.

hook_list : a GHookList.
hook : the GHook to add to the start of *hook_list*.

---

g_hook_insert_before ()

```
void        g_hook_insert_before            (GHookList *hook_list,
                                             GHook *sibling,
                                             GHook *hook);
```

Inserts a GHook into a GHookList, before a given GHook.

hook_list : a GHookList.
sibling : the GHook to insert the new GHook before.
hook : the GHook to insert.

---

g_hook_insert_sorted ()

```
void        g_hook_insert_sorted            (GHookList *hook_list,
                                             GHook *hook,
                                             GHookCompareFunc func);
```

Inserts a GHook into a GHookList, sorted by the given function.

hook_list : a GHookList.
hook : the GHook to insert.
func : the comparison function used to sort the GHook elements.

GHookCompareFunc ()

```
gint        (*GHookCompareFunc)             (GHook *new_hook,
                                             GHook *sibling);
```

Defines the type of function used to compare GHook elements in `g_hook_insert_sorted()`.

new_hook : the GHook being inserted.
sibling : the GHook to compare with *new_hook*.
Returns : a value <= 0 if *new_hook* should be before *sibling*.

---

g_hook_compare_ids ()

```
gint        g_hook_compare_ids              (GHook *new_hook,
                                             GHook *sibling);
```

Compares the ids of two GHook elements, returning a negative value if the second id is greater than the first.

new_hook : a GHook.
sibling : a GHook to compare with *new_hook*.
Returns : a value <= 0 if the id of *sibling* is >= the id of *new_hook*.

---

g_hook_get ()

```
GHook*      g_hook_get                      (GHookList *hook_list,
                                             gulong hook_id);
```

Returns the GHook with the given id, or `NULL` if it is not found.

hook_list : a GHookList.
hook_id : a hook id.
Returns : the GHook with the given id, or `NULL` if it is not found.

---

g_hook_find ()

```
GHook*      g_hook_find                     (GHookList *hook_list,
                                             gboolean need_valids,
                                             GHookFindFunc func,
                                             gpointer data);
```

Finds a GHook in a GHookList using the given function to test for a match.

hook_list : a GHookList.
need_valids : `TRUE` if GHook elements which have been destroyed should be skipped.
func : the function to call for each GHook, which should return `TRUE` when the GHook has been found.
data : the data to pass to *func*.
Returns : the found GHook or `NULL` if no matching GHook is found.

GHookFindFunc ()

```
gboolean    (*GHookFindFunc)                (GHook *hook,
                                             gpointer data);
```

Defines the type of the function passed to g_hook_find().

hook : a GHook.
data : user data passed to g_hook_find_func().
Returns : TRUE if the required GHook has been found.

---

g_hook_find_data ()

```
GHook*      g_hook_find_data                (GHookList *hook_list,
                                             gboolean need_valids,
                                             gpointer data);
```

Finds a GHook in a GHookList with the given data.

hook_list : a GHookList.
need_valids : TRUE if GHook elements which have been destroyed should be skipped.
data : the data to find.
Returns : the GHook with the given *data* or NULL if no matching GHook is found.

---

g_hook_find_func ()

```
GHook*      g_hook_find_func                (GHookList *hook_list,
                                             gboolean need_valids,
                                             gpointer func);
```

Finds a GHook in a GHookList with the given function.

hook_list : a GHookList.
need_valids : TRUE if GHook elements which have been destroyed should be skipped.
func : the function to find.
Returns : the GHook with the given *func* or NULL if no matching GHook is found.

---

g_hook_find_func_data ()

```
GHook*      g_hook_find_func_data           (GHookList *hook_list,
                                             gboolean need_valids,
                                             gpointer func,
                                             gpointer data);
```


Finds a GHook in a GHookList with the given function and data.

hook_list : a GHookList.
need_valids : TRUE if GHook elements which have been destroyed should be skipped.
func : the function to find.
data : the data to find.
Returns : the GHook with the given *func* and *data* or NULL if no matching GHook is found.

---

g_hook_first_valid ()

```
GHook*      g_hook_first_valid              (GHookList *hook_list,
                                             gboolean may_be_in_call);
```

Returns the first GHook in a GHookList which has not been destroyed. The reference count for the GHook is incremented, so you must call g_hook_unref() to restore it when no longer needed. (Or call g_hook_next_valid() if you are stepping through the GHookList.)

hook_list : a GHookList.
may_be_in_call : TRUE if hooks which are currently running (e.g. in another thread) are considered valid. If set to FALSE, these are skipped.
Returns : the first valid GHook, or NULL if none are valid.

---

g_hook_next_valid ()

```
GHook*      g_hook_next_valid               (GHookList *hook_list,
                                             GHook *hook,
                                             gboolean may_be_in_call);
```

Returns the next GHook in a GHookList which has not been destroyed. The reference count for the GHook is incremented, so you must call g_hook_unref() to restore it when no longer needed. (Or continue to call g_hook_next_valid() until NULL is returned.)

hook_list : a GHookList.
hook : the current GHook.
may_be_in_call : TRUE if hooks which are currently running (e.g. in another thread) are considered valid. If set to FALSE, these are skipped.
Returns : the next valid GHook, or NULL if none are valid.

---

enum GHookFlagMask

```
typedef enum
{
  G_HOOK_FLAG_ACTIVE        = 1 << 0,
  G_HOOK_FLAG_IN_CALL       = 1 << 1,
```

```
  G_HOOK_FLAG_MASK          = 0x0f
} GHookFlagMask;
```

Flags used internally in the GHook implementation.

`G_HOOK_FLAG_ACTIVE` set if the hook has not been destroyed.
`G_HOOK_FLAG_IN_CALL` set if the hook is currently being run.
`G_HOOK_FLAG_MASK`

G_HOOK_FLAGS()

```
#define G_HOOK_FLAGS(hook)              (G_HOOK (hook)->flags)
```

Returns the flags of a hook.

hook : a GHook.

G_HOOK_FLAG_USER_SHIFT

```
#define G_HOOK_FLAG_USER_SHIFT  (4)
```

The position of the first bit which is not reserved for internal use be the GHook implementation, i.e. `1 << G_HOOK_FLAG_USER_SHIFT` is the first bit which can be used for application-defined flags.

G_HOOK()

```
#define G_HOOK(hook)                    ((GHook*) (hook))
```

Casts a pointer to a `GHook*`.

hook : a pointer.

G_HOOK_IS_VALID()

```
#define     G_HOOK_IS_VALID(hook)
```

Returns `TRUE` if the GHook is valid, i.e. it is in a GHookList, it is active and it has not been destroyed.

hook : a GHook.
Returns : `TRUE` if the GHook is valid.

G_HOOK_ACTIVE()

```
#define     G_HOOK_ACTIVE(hook)
```

Returns `TRUE` if the GHook is active, which is normally `TRUE` until the GHook is destroyed.

hook : a GHook.
Returns : `TRUE` if the GHook is active.

G_HOOK_IN_CALL()

```
#define     G_HOOK_IN_CALL(hook)
```

Returns `TRUE` if the GHook function is currently executing.

hook : a GHook.
Returns : `TRUE` if the GHook function is currently executing.

G_HOOK_IS_UNLINKED()

```
#define     G_HOOK_IS_UNLINKED(hook)
```

Returns `TRUE` if the GHook is not in a GHookList.

hook : a GHook.
Returns : `TRUE` if the GHook is not in a GHookList.

---

g_hook_ref ()

```
GHook*      g_hook_ref                      (GHookList *hook_list,
                                             GHook *hook);
```

Increments the reference count for a GHook.

hook_list : a GHookList.
hook : the GHook to increment the reference count of.
Returns : the *hook* that was passed in (since 2.6)

---

g_hook_unref ()

```
void        g_hook_unref                    (GHookList *hook_list,
                                             GHook *hook);
```

Decrements the reference count of a GHook. If the reference count falls to 0, the GHook is removed from the GHookList and `g_hook_free()` is called to free it.

hook_list : a GHookList.
hook : the GHook to unref.

---

g_hook_free ()

```
void        g_hook_free                     (GHookList *hook_list,
                                             GHook *hook);
```

Calls the GHookList *hook_free* function if it exists, and frees the memory allocated for the GHook.

hook_list : a GHookList.
hook : the GHook to free.

---

g_hook_destroy ()

```
gboolean    g_hook_destroy                  (GHookList *hook_list,
                                             gulong hook_id);
```

Destroys a GHook, given its ID.

hook_list : a GHookList.
hook_id : a hook ID.
Returns : `TRUE` if the GHook was found in the GHookList and destroyed.

---

g_hook_destroy_link ()

```
void        g_hook_destroy_link             (GHookList *hook_list,
                                             GHook *hook);
```

Removes one GHook from a GHookList, marking it inactive and calling `g_hook_unref()` on it.

hook_list : a GHookList.
hook : the GHook to remove.

<< Random Numbers **Miscellaneous Utility Functions >>**

Miscellaneous Utility Functions

Miscellaneous Utility Functions — a selection of portable utility functions.

Synopsis

```
#include <glib.h>


G_CONST_RETURN gchar* g_get_application_name
                                        (void);
void        g_set_application_name      (const gchar *application_name);
gchar*      g_get_prgname               (void);
void        g_set_prgname               (const gchar *prgname);
G_CONST_RETURN gchar* g_getenv          (const gchar *variable);
gboolean    g_setenv                    (const gchar *variable,
                                         const gchar *value,
                                         gboolean overwrite);
void        g_unsetenv                  (const gchar *variable);
G_CONST_RETURN gchar* g_get_user_name    (void);
G_CONST_RETURN gchar* g_get_real_name    (void);
G_CONST_RETURN gchar* g_get_user_cache_dir  (void);
G_CONST_RETURN gchar* g_get_user_data_dir   (void);
G_CONST_RETURN gchar* g_get_user_config_dir (void);
G_CONST_RETURN gchar* G_CONST_RETURN * g_get_system_data_dirs
                                        (void);
G_CONST_RETURN gchar* G_CONST_RETURN * g_get_system_config_dirs
                                        (void);

G_CONST_RETURN gchar* g_get_home_dir     (void);
G_CONST_RETURN gchar* g_get_tmp_dir      (void);
gchar*      g_get_current_dir           (void);
G_CONST_RETURN gchar* g_basename         (const gchar *file_name);
#define     g_dirname
gboolean    g_path_is_absolute          (const gchar *file_name);
G_CONST_RETURN gchar* g_path_skip_root   (const gchar *file_name);
gchar*      g_path_get_basename         (const gchar *file_name);
gchar*      g_path_get_dirname          (const gchar *file_name);
gchar*      g_build_filename            (const gchar *first_element,
                                         ...);
gchar*      g_build_path                (const gchar *separator,
                                         const gchar *first_element,
                                         ...);

gchar*      g_find_program_in_path      (const gchar *program);

gint        g_bit_nth_lsf               (gulong mask,
                                         gint nth_bit);
gint        g_bit_nth_msf               (gulong mask,
                                         gint nth_bit);
guint       g_bit_storage               (gulong number);

guint       g_spaced_primes_closest     (guint num);

void        g_atexit                    (GVoidFunc func);
```



```
guint       g_parse_debug_string        (const gchar *string,
                                         const GDebugKey *keys,
                                         guint nkeys);
            GDebugKey;

void        (*GVoidFunc)                (void);
void        (*GFreeFunc)                (gpointer data);

void        g_qsort_with_data           (gconstpointer pbase,
                                         gint total_elems,
                                         gsize size,
                                         GCompareDataFunc compare_func,
                                         gpointer user_data);

void        g_nullify_pointer           (gpointer *nullify_location);
```

Description

These are portable utility functions.

Details

g_get_application_name ()

```
G_CONST_RETURN gchar* g_get_application_name
                                        (void);
```

Gets a human-readable name for the application, as set by `g_set_application_name()`. This name should be localized if possible, and is intended for display to the user. Contrast with `g_get_prgname()`, which gets a non-localized name. If `g_set_application_name()` has not been called, returns the result of `g_get_prgname()` (which may be `NULL` if `g_set_prgname()` has also not been called).

Returns : human-readable application name. may return `NULL`

Since 2.2

---

g_set_application_name ()

```
void        g_set_application_name      (const gchar *application_name);
```

Sets a human-readable name for the application. This name should be localized if possible, and is intended for display to the user. Contrast with `g_set_prgname()`, which sets a non-localized name. `g_set_prgname()` will be called automatically by `gtk_init()`, but `g_set_application_name()` will not.

Note that for thread safety reasons, this function can only be called once.

The application name will be used in contexts such as error messages, or when displaying an

application's name in the task list.

application_name : localized name of the application

---

g_get_prgname ()

```
gchar*      g_get_prgname                   (void);
```

Gets the name of the program. This name should NOT be localized, contrast with g_get_application_name(). (If you are using GDK or GTK+ the program name is set in gdk_init(), which is called by gtk_init(). The program name is found by taking the last component of argv[0].)

Returns : the name of the program.

---

g_set_prgname ()

```
void        g_set_prgname                   (const gchar *prgname);
```

Sets the name of the program. This name should NOT be localized, contrast with g_set_application_name(). Note that for thread-safety reasons this function can only be called once.

prgname : the name of the program.

---

g_getenv ()

```
G_CONST_RETURN gchar* g_getenv              (const gchar *variable);
```

Returns an environment variable.

variable : the environment variable to get.
Returns : the value of the environment variable, or NULL if the environment variable is not found. The returned string may be overwritten by the next call to g_getenv(), g_setenv() or g_unsetenv().

---

g_setenv ()

```
gboolean    g_setenv                        (const gchar *variable,
                                             const gchar *value,
                                             gboolean overwrite);
```

Sets an environment variable.

Note that on some systems, the memory used for the variable and its value can't be reclaimed later.

variable : the environment variable to set, must not contain '='.
value : the value for to set the variable to.
overwrite : whether to change the variable if it already exists.
Returns : FALSE if the environment variable couldn't be set.

Since 2.4

---

g_unsetenv ()

```
void        g_unsetenv                      (const gchar *variable);
```

Removes an environment variable from the environment.

Note that on some systems, the memory used for the variable and its value can't be reclaimed. Furthermore, this function can't be guaranteed to operate in a threadsafe way.

variable : the environment variable to remove, must not contain '='.

Since 2.4

---

g_get_user_name ()

```
G_CONST_RETURN gchar* g_get_user_name       (void);
```

Gets the user name of the current user.

Returns : the user name of the current user.

---

g_get_real_name ()

```
G_CONST_RETURN gchar* g_get_real_name       (void);
```

Gets the real name of the user. This usually comes from the user's entry in the passwd file. The encoding of the returned string is system defined. If the real user name cannot be determined, the string "Unknown" is returned.

Returns : the user's real name.

---

g_get_user_cache_dir ()

```
G_CONST_RETURN gchar* g_get_user_cache_dir  (void);
```

Returns a base directory in which to store non-essential, cached data specific to particular user.

On Unix platforms this is determined using the mechanisms described in the XDG Base Directory Specification

Returns : a string owned by GLib that must not be modified or freed.

Since 2.6

---

g_get_user_data_dir ()

```
G_CONST_RETURN gchar* g_get_user_data_dir   (void);
```

Returns a base directory in which to access application data such as icons that is customized for a particular user.

On Unix platforms this is determined using the mechanisms described in the XDG Base Directory Specification

Returns : a string owned by GLib that must not be modified or freed.

Since 2.6

---

g_get_user_config_dir ()

```
G_CONST_RETURN gchar* g_get_user_config_dir (void);
```

Returns a base directory in which to store user-specific application configuration information such as user preferences and settings.

On Unix platforms this is determined using the mechanisms described in the XDG Base Directory Specification

Returns : a string owned by GLib that must not be modified or freed.

Since 2.6

---

g_get_system_data_dirs ()

```
G_CONST_RETURN gchar* G_CONST_RETURN * g_get_system_data_dirs
```



```
                                                (void);
```

Returns an ordered list of base directories in which to access system-wide application data.

On Unix platforms this is determined using the mechanisms described in the XDG Base Directory Specification

Returns : a `NULL`-terminated array of strings owned by GLib that must not be modified or freed.

Since 2.6

---

g_get_system_config_dirs ()

```
G_CONST_RETURN gchar* G_CONST_RETURN * g_get_system_config_dirs
                                                (void);
```

Returns an ordered list of base directories in which to access system-wide configuration information.

On Unix platforms this is determined using the mechanisms described in the XDG Base Directory Specification

Returns : a `NULL`-terminated array of strings owned by GLib that must not be modified or freed.

Since 2.6

---

g_get_home_dir ()

```
G_CONST_RETURN gchar* g_get_home_dir        (void);
```

Gets the current user's home directory.

Note that in contrast to traditional Unix tools, this function prefers `passwd` entries over the `HOME` environment variable.

Returns : the current user's home directory.

---

g_get_tmp_dir ()

```
G_CONST_RETURN gchar* g_get_tmp_dir         (void);
```

Gets the directory to use for temporary files. This is found from inspecting the environment variables

TMPDIR, TMP, and TEMP in that order. If none of those are defined "/tmp" is returned on UNIX and "C:\" on Windows.

Returns : the directory to use for temporary files.

---

g_get_current_dir ()

```
gchar*      g_get_current_dir               (void);
```

Gets the current directory. The returned string should be freed when no longer needed.

Returns : the current directory.

---

g_basename ()

```
G_CONST_RETURN gchar* g_basename            (const gchar *file_name);
```

Warning

g_basename is deprecated and should not be used in newly-written code. Use g_path_get_basename() instead, but notice that g_path_get_basename() allocates new memory for the returned string, unlike this function which returns a pointer into the argument.

Gets the name of the file without any leading directory components. It returns a pointer into the given file name string.

file_name : the name of the file.
Returns : the name of the file without any leading directory components.

---

g_dirname

```
#define     g_dirname
```

Warning

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

This function is deprecated and will be removed in the next major release of GLib. Use g_path_get_dirname() instead.

Gets the directory components of a file name. If the file name has no directory components "." is returned. The returned string should be freed when no longer needed.

Returns : the directory components of the file.

---

g_path_is_absolute ()

```
gboolean    g_path_is_absolute              (const gchar *file_name);
```

Returns TRUE if the given *file_name* is an absolute file name, i.e. it contains a full path from the root directory such as '/usr/local' on UNIX or 'C:\windows' on Windows systems.

file_name : a file name.
Returns : TRUE if *file_name* is an absolute path.

---

g_path_skip_root ()

```
G_CONST_RETURN gchar* g_path_skip_root      (const gchar *file_name);
```

Returns a pointer into *file_name* after the root component, i.e. after the '/' in UNIX or 'C:\' under Windows. If *file_name* is not an absolute path it returns NULL.

file_name : a file name.
Returns : a pointer into *file_name* after the root component.

---

g_path_get_basename ()

```
gchar*      g_path_get_basename             (const gchar *file_name);
```

Gets the last component of the filename. If *file_name* ends with a directory separator it gets the component before the last slash. If *file_name* consists only of directory separators (and on Windows, possibly a drive letter), a single separator is returned. If *file_name* is empty, it gets ".".

file_name : the name of the file.
Returns : a newly allocated string containing the last component of the filename.

---

g_path_get_dirname ()

```
gchar*      g_path_get_dirname              (const gchar *file_name);
```

Gets the directory components of a file name. If the file name has no directory components "." is returned. The returned string should be freed when no longer needed.

file_name : the name of the file.
Returns : the directory components of the file.

---

g_build_filename ()

```
gchar*      g_build_filename                (const gchar *first_element,
                                             ...);
```

Creates a filename from a series of elements using the correct separator for filenames.

On Unix, this function behaves identically to `g_build_path (G_DIR_SEPARATOR_S, first_element, ....)`.

On Windows, it takes into account that either the backslash (\ or slash (/) can be used as separator in filenames, but otherwise behaves as on Unix. When file pathname separators need to be inserted, the one that last previously occurred in the parameters (reading from left to right) is used.

No attempt is made to force the resulting filename to be an absolute path. If the first element is a relative path, the result will be a relative path.

first_element : the first element in the path
... : remaining elements in path, terminated by `NULL`
Returns : a newly-allocated string that must be freed with `g_free()`.

---

g_build_path ()

```
gchar*      g_build_path                    (const gchar *separator,
                                             const gchar *first_element,
                                             ...);
```

Creates a path from a series of elements using *separator* as the separator between elements. At the boundary between two elements, any trailing occurrences of separator in the first element, or leading occurrences of separator in the second element are removed and exactly one copy of the separator is inserted.

Empty elements are ignored.

The number of leading copies of the separator on the result is the same as the number of leading copies of the separator on the first non-empty element.

The number of trailing copies of the separator on the result is the same as the number of trailing copies of the separator on the last non-empty element. (Determination of the number of trailing copies is done without stripping leading copies, so if the separator is ABA, ABABA has 1 trailing copy.)

However, if there is only a single non-empty element, and there are no characters in that element not part of the leading or trailing separators, then the result is exactly the original value of that element.

Other than for determination of the number of leading and trailing copies of the separator, elements

consisting only of copies of the separator are ignored.

separator : a string used to separator the elements of the path.
first_element : the first element in the path
... : remaining elements in path, terminated by `NULL`
Returns : a newly-allocated string that must be freed with `g_free()`.

---

g_find_program_in_path ()

```
gchar*      g_find_program_in_path          (const gchar *program);
```

Locates the first executable named *program* in the user's path, in the same way that `execvp()` would locate it. Returns an allocated string with the absolute path name, or NULL if the program is not found in the path. If *program* is already an absolute path, returns a copy of *program* if *program* exists and is executable, and NULL otherwise.

On Windows, if *program* does not have a file type suffix, tries to append the suffixes in the PATHEXT environment variable (if that doesn't exists, the suffixes .com, .exe, and .bat) in turn, and then look for the resulting file name in the same way as `CreateProcess()` would. This means first in the directory where the program was loaded from, then in the current directory, then in the Windows 32-bit system directory, then in the Windows directory, and finally in the directories in the PATH environment variable. If the program is found, the return value contains the full name including the type suffix.

program : a program name
Returns : absolute path, or NULL

---

g_bit_nth_lsf ()

```
gint        g_bit_nth_lsf                   (gulong mask,
                                             gint nth_bit);
```

Find the position of the first bit set in *mask*, searching from (but not including) *nth_bit* upwards. Bits are numbered from 0 (least significant) to sizeof(gulong) * 8 - 1 (31 or 63, usually). To start searching from the 0th bit, set *nth_bit* to -1.

mask : a gulong containing flags.
nth_bit : the index of the bit to start the search from.
Returns : the index of the first bit set which is higher than *nth_bit*.

---

g_bit_nth_msf ()

```
gint        g_bit_nth_msf                   (gulong mask,
                                             gint nth_bit);
```

Find the position of the first bit set in *mask*, searching from (but not including) *nth_bit* downwards. Bits are numbered from 0 (least significant) to sizeof(gulong) * 8 - 1 (31 or 63, usually). To start searching from the last bit, set *nth_bit* to -1 or GLIB_SIZEOF_LONG * 8.

mask : a gulong containing flags.
nth_bit : the index of the bit to start the search from.
Returns : the index of the first bit set which is lower than *nth_bit*.

---

g_bit_storage ()

```
guint       g_bit_storage                   (gulong number);
```

Gets the number of bits used to hold *number*, e.g. if *number* is 4, 3 bits are needed.

number : a guint.
Returns : the number of bits used to hold *number*.

---

g_spaced_primes_closest ()

```
guint       g_spaced_primes_closest         (guint num);
```

Gets the smallest prime number from a built-in array of primes which is larger than *num*. This is used within GLib to calculate the optimum size of a GHashTable.

The built-in array of primes ranges from 11 to 13845163 such that each prime is approximately 1.5-2 times the previous prime.

num : a guint.
Returns : the smallest prime number from a built-in array of primes which is larger than *num*.

---

g_atexit ()

```
void        g_atexit                        (GVoidFunc func);
```

Specifies a function to be called at normal program termination.

func : the function to call on normal program termination.

---

g_parse_debug_string ()

```
guint       g_parse_debug_string            (const gchar *string,
                                             const GDebugKey *keys,
                                             guint nkeys);
```

Parses a string containing debugging options separated by ':' into a guint containing bit flags. This is used within GDK and GTK+ to parse the debug options passed on the command line or through environment variables.

string : a list of debug options separated by ':' or "all" to set all flags.
keys : pointer to an array of GDebugKey which associate strings with bit flags.
nkeys : the number of GDebugKey in the array.
Returns : the combined set of bit flags.

GDebugKey

```
typedef struct {
  gchar *key;
  guint  value;
} GDebugKey;
```

Associates a string with a bit flag. Used in `g_parse_debug_string()`.

GVoidFunc ()

```
void        (*GVoidFunc)                    (void);
```

Declares a type of function which takes no arguments and has no return value. It is used to specify the type function passed to `g_atexit()`.

GFreeFunc ()

```
void        (*GFreeFunc)                    (gpointer data);
```

Declares a type of function which takes an arbitrary data pointer argument and has no return value. It is not currently used in GLib or GTK+.

data : a data pointer.

---

g_qsort_with_data ()

```
void        g_qsort_with_data               (gconstpointer pbase,
                                             gint total_elems,
                                             gsize size,
                                             GCompareDataFunc compare_func,
                                             gpointer user_data);
```


http://developer.gnome.org/doc/API/2.0/glib/glib-Miscellaneous-Utility-Functions.html 16/11/2004

This is just like the standard C `qsort()` function, but the comparison routine accepts a user data argument.

pbase : start of array to sort
total_elems : elements in the array
size : size of each element
compare_func : function to compare elements
user_data : data to pass to *compare_func*

---

g_nullify_pointer ()

```
void        g_nullify_pointer               (gpointer *nullify_location);
```

Set the pointer at the specified location to `NULL`.

nullify_location : the memory address of the pointer.

<< Hook Functions **Lexical Scanner >>**

Lexical Scanner

Lexical Scanner — a general purpose lexical scanner.

Synopsis

```
#include <glib.h>


            GScanner;
GScanner*   g_scanner_new                   (const GScannerConfig *config_templ)
            GScannerConfig;

void        g_scanner_input_file            (GScanner *scanner,
                                             gint input_fd);
void        g_scanner_sync_file_offset      (GScanner *scanner);
void        g_scanner_input_text            (GScanner *scanner,
                                             const      gchar *text,
                                             guint text_len);
GTokenType  g_scanner_peek_next_token       (GScanner *scanner);
GTokenType  g_scanner_get_next_token        (GScanner *scanner);

guint       g_scanner_cur_line              (GScanner *scanner);
guint       g_scanner_cur_position          (GScanner *scanner);
GTokenType  g_scanner_cur_token             (GScanner *scanner);
GTokenValue g_scanner_cur_value             (GScanner *scanner);
gboolean    g_scanner_eof                   (GScanner *scanner);

guint       g_scanner_set_scope             (GScanner *scanner,
                                             guint scope_id);
void        g_scanner_scope_add_symbol      (GScanner *scanner,
                                             guint scope_id,
                                             const gchar *symbol,
                                             gpointer value);
void        g_scanner_scope_foreach_symbol  (GScanner *scanner,
                                             guint scope_id,
                                             GHFunc func,
                                             gpointer user_data);
gpointer    g_scanner_scope_lookup_symbol   (GScanner *scanner,
                                             guint scope_id,
                                             const gchar *symbol);
void        g_scanner_scope_remove_symbol   (GScanner *scanner,
                                             guint scope_id,
                                             const gchar *symbol);

#define     g_scanner_freeze_symbol_table   (scanner)
#define     g_scanner_thaw_symbol_table     (scanner)
gpointer    g_scanner_lookup_symbol         (GScanner *scanner,
                                             const gchar *symbol);

void        g_scanner_warn                  (GScanner *scanner,
                                             const gchar *format,
                                             ...);
void        g_scanner_error                 (GScanner *scanner,
                                             const gchar *format,
```



```
                                             ...);
void        g_scanner_unexp_token           (GScanner *scanner,
                                             GTokenType expected_token,
                                             const gchar *identifier_spec,
                                             const gchar *symbol_spec,
                                             const gchar *symbol_name,
                                             const gchar *message,
                                             gint is_error);
void        (*GScannerMsgFunc)              (GScanner *scanner,
                                             gchar *message,
                                             gboolean error);

void        g_scanner_destroy               (GScanner *scanner);

enum        GTokenType;
union       GTokenValue;
enum        GErrorType;
#define     G_CSET_a_2_z
#define     G_CSET_A_2_Z
#define     G_CSET_DIGITS
#define     G_CSET_LATINC
#define     G_CSET_LATINS

#define     g_scanner_add_symbol            ( scanner, symbol, value )
#define     g_scanner_remove_symbol         ( scanner, symbol )
#define     g_scanner_foreach_symbol        ( scanner, func, data )
```

Description

The GScanner and its associated functions provide a general purpose lexical scanner.

FIXME: really needs an example and more detail, but I don't completely understand it myself. Look at gtkrc.c for some code using the scanner.

Details

GScanner

```
typedef struct {
  /* unused fields */
  gpointer              user_data;
  guint                 max_parse_errors;

  /* g_scanner_error() increments this field */
  guint                 parse_errors;

  /* name of input stream, featured by the default message handler */
  const gchar           *input_name;

  /* quarked data */
  GData                 *qdata;

  /* link into the scanner configuration */
  GScannerConfig        *config;

  /* fields filled in after g_scanner_get_next_token() */
  GTokenType            token;
  GTokenValue           value;
  guint                 line;
```

```
  guint                position;

  /* fields filled in after g_scanner_peek_next_token() */
  GTokenType           next_token;
  GTokenValue          next_value;
  guint                next_line;
  guint                next_position;

  /* to be considered private */
  GHashTable           *symbol_table;
  gint                 input_fd;
  const gchar          *text;
  const gchar          *text_end;
  gchar                *buffer;
  guint                scope_id;

  /* handler function for _warn and _error */
  GScannerMsgFunc      msg_handler;
} GScanner;
```

The data structure representing a lexical scanner.

You should set *input_name* after creating the scanner, since it is used by the default message handler when displaying warnings and errors. If you are scanning a file, the file name would be a good choice.

The *user_data* and *max_parse_errors* fields are not used. If you need to associate extra data with the scanner you can place them here.

If you want to use your own message handler you can set the *msg_handler* field. The type of the message handler function is declared by GScannerMsgFunc.

---

g_scanner_new ()

```
GScanner*   g_scanner_new                   (const GScannerConfig *config_templ)
```

Creates a new GScanner. The *config_templ* structure specifies the initial settings of the scanner, which are copied into the GScanner *config* field. If you pass NULL then the default settings are used.

config_templ : the initial scanner settings.

Returns : the new GScanner.

GScannerConfig

```
typedef struct {
  /* Character sets
   */
  gchar         *cset_skip_characters;          /* default: " \t\n" */
  gchar         *cset_identifier_first;
  gchar         *cset_identifier_nth;
  gchar         *cpair_comment_single;          /* default: "#\n" */
```



```
  /* Should symbol lookup work case sensitive?
   */
  guint         case_sensitive : 1;

  /* Boolean values to be adjusted "on the fly"
   * to configure scanning behaviour.
   */
  guint         skip_comment_multi : 1;         /* C like comment */
  guint         skip_comment_single : 1;        /* single line comment */
  guint         scan_comment_multi : 1;         /* scan multi line comments? */
  guint         scan_identifier : 1;
  guint         scan_identifier_1char : 1;
  guint         scan_identifier_NULL : 1;
  guint         scan_symbols : 1;
  guint         scan_binary : 1;
  guint         scan_octal : 1;
  guint         scan_float : 1;
  guint         scan_hex : 1;                   /* `0x0ff0' */
  guint         scan_hex_dollar : 1;            /* `$0ff0' */
  guint         scan_string_sq : 1;             /* string: 'anything' */
  guint         scan_string_dq : 1;             /* string: "\\-escapes!\n" */
  guint         numbers_2_int : 1;              /* bin, octal, hex => int */
  guint         int_2_float : 1;                /* int => G_TOKEN_FLOAT? */
  guint         identifier_2_string : 1;
  guint         char_2_token : 1;               /* return G_TOKEN_CHAR? */
  guint         symbol_2_token : 1;
  guint         scope_0_fallback : 1;           /* try scope 0 on lookups? */
  guint         store_int64 : 1;                /* use value.v_int64 rather than
  guint         padding_dummy;
} GScannerConfig;
```

Specifies the GScanner settings.

cset_skip_characters specifies which characters should be skipped by the scanner (the default is the whitespace characters: space, tab, carriage-return and line-feed).

cset_identifier_first specifies the characters which can start identifiers (the default is G_CSET_a_2_z, "_", and G_CSET_A_2_Z).

cset_identifier_nth specifies the characters which can be used in identifiers, after the first character (the default is G_CSET_a_2_z, "_0123456789", G_CSET_A_2_Z, G_CSET_LATINS, G_CSET_LATINC).

cpair_comment_single specifies the characters at the start and end of single-line comments. The default is "#\n" which means that single-line comments start with a '#' and continue until a '\n' (end of line).

case_sensitive specifies if symbols are case sensitive (the default is FALSE).

skip_comment_multi specifies if multi-line comments are skipped and not returned as tokens (the default is TRUE).

skip_comment_single specifies if single-line comments are skipped and not returned as tokens (the default is TRUE).

scan_comment_multi specifies if multi-line comments are recognized (the default is TRUE).

scan_identifier specifies if identifiers are recognized (the default is TRUE).

scan_identifier_1char specifies if single-character identifiers are recognized (the default is FALSE).

scan_identifier_NULL specifies if NULL is reported as G_TOKEN_IDENTIFIER_NULL. (the default is FALSE).

scan_symbols specifies if symbols are recognized (the default is TRUE).

scan_binary specifies if binary numbers are recognized (the default is FALSE).

scan_octal specifies if octal numbers are recognized (the default is TRUE).

scan_float specifies if floating point numbers are recognized (the default is TRUE).

scan_hex specifies if hexadecimal numbers are recognized (the default is TRUE).

scan_hex_dollar specifies if '$' is recognized as a prefix for hexadecimal numbers (the default is FALSE).

scan_string_sq specifies if strings can be enclosed in single quotes (the default is TRUE).

scan_string_dq specifies if strings can be enclosed in double quotes (the default is TRUE).

numbers_2_int specifies if binary, octal and hexadecimal numbers are reported as G_TOKEN_INT (the default is TRUE).

int_2_float specifies if all numbers are reported as G_TOKEN_FLOAT (the default is FALSE).

identifier_2_string specifies if identifiers are reported as strings (the default is FALSE).

char_2_token specifies if characters are reported by setting token = ch or as G_TOKEN_CHAR (the default is TRUE).

symbol_2_token specifies if symbols are reported by setting token = v_symbol or as G_TOKEN_SYMBOL (the default is FALSE).

scope_0_fallback specifies if a symbol is searched for in the default scope in addition to the current scope (the default is FALSE).

---

g_scanner_input_file ()

```
void        g_scanner_input_file            (GScanner *scanner,
                                             gint input_fd);
```

Prepares to scan a file.

scanner : a GScanner.
input_fd : a file descriptor.

---

g_scanner_sync_file_offset ()

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

Rewinds the filedescriptor to the current buffer position and blows the file read ahead buffer. This is useful for third party uses of the scanners filedescriptor, which hooks onto the current scanning position.

scanner : a GScanner.

---

g_scanner_input_text ()

```
void        g_scanner_input_text            (GScanner *scanner,
                                             const      gchar *text,
                                             guint text_len);
```

Prepares to scan a text buffer.

scanner : a GScanner.
text : the text buffer to scan.
text_len : the length of the text buffer.

---

g_scanner_peek_next_token ()

```
GTokenType  g_scanner_peek_next_token       (GScanner *scanner);
```

Gets the next token, without removing it from the input stream. The token data is placed in the *next_token*, *next_value*, *next_line*, and *next_position* fields of the GScanner structure.

scanner : a GScanner.
Returns : the type of the token.

---

g_scanner_get_next_token ()

```
GTokenType  g_scanner_get_next_token        (GScanner *scanner);
```

Gets the next token, removing it from the input stream. The token data is placed in the *token*, *value*, *line*, and *position* fields of the GScanner structure.

scanner : a GScanner.
Returns : the type of the token.

---

g_scanner_cur_line ()

```
guint       g_scanner_cur_line              (GScanner *scanner);
```

Gets the current line in the input stream (counting from 1).

scanner : a GScanner.
Returns : the current line.

---

g_scanner_cur_position ()

```
guint       g_scanner_cur_position          (GScanner *scanner);
```

Gets the current position in the current line (counting from 0).

scanner : a GScanner.
Returns : the current position on the line.

---

g_scanner_cur_token ()

```
GTokenType  g_scanner_cur_token             (GScanner *scanner);
```

Gets the current token type. This is simply the *token* field in the GScanner structure.

scanner : a GScanner.
Returns : the current token type.

---

g_scanner_cur_value ()

```
GTokenValue g_scanner_cur_value             (GScanner *scanner);
```

Gets the current token value. This is simply the *value* field in the GScanner structure.

scanner : a GScanner.
Returns : the current token value.

---

g_scanner_eof ()

```
gboolean    g_scanner_eof                   (GScanner *scanner);
```

Returns TRUE if the scanner has reached the end of the file or text buffer.

scanner : a GScanner.
Returns : TRUE if the scanner has reached the end of the file or text buffer.

---

g_scanner_set_scope ()

```
guint       g_scanner_set_scope             (GScanner *scanner,
                                             guint scope_id);
```

Sets the current scope.

scanner : a GScanner.
scope_id : the new scope id.
Returns : the old scope id.

---

g_scanner_scope_add_symbol ()

```
void        g_scanner_scope_add_symbol      (GScanner *scanner,
                                             guint scope_id,
                                             const gchar *symbol,
                                             gpointer value);
```

Adds a symbol to the given scope.

scanner : a GScanner.
scope_id : the scope id.
symbol : the symbol to add.
value : the value of the symbol.

---

g_scanner_scope_foreach_symbol ()

```
void        g_scanner_scope_foreach_symbol  (GScanner *scanner,
                                             guint scope_id,
                                             GHFunc func,
                                             gpointer user_data);
```

Calls the given function for each of the symbol/value pairs in the given scope of the GScanner. The function is passed the symbol and value of each pair, and the given *user_data* parameter.

scanner : a GScanner.
scope_id : the scope id.
func : the function to call for each symbol/value pair.
user_data : user data to pass to the function.

---

g_scanner_scope_lookup_symbol ()

```
gpointer    g_scanner_scope_lookup_symbol   (GScanner *scanner,
                                             guint scope_id,
                                             const gchar *symbol);
```

Looks up a symbol in a scope and return its value. If the symbol is not bound in the scope, NULL is returned.

scanner : a GScanner.
scope_id : the scope id.
symbol : the symbol to look up.
Returns : the value of *symbol* in the given scope, or NULL if *symbol* is not bound in the given scope.

---

g_scanner_scope_remove_symbol ()

```
void        g_scanner_scope_remove_symbol   (GScanner *scanner,
                                             guint scope_id,
                                             const gchar *symbol);
```

Removes a symbol from a scope.

scanner : a GScanner.
scope_id : the scope id.
symbol : the symbol to remove.

---

g_scanner_freeze_symbol_table()

```
#define     g_scanner_freeze_symbol_table(scanner)
```

Warning

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

This function is deprecated and will be removed in the next major release of GLib. It does nothing.

scanner : a GScanner.

---

g_scanner_thaw_symbol_table()

```
#define     g_scanner_thaw_symbol_table(scanner)
```


Warning

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

This function is deprecated and will be removed in the next major release of GLib. It does nothing.

scanner : a GScanner.

---

g_scanner_lookup_symbol ()

```
gpointer    g_scanner_lookup_symbol         (GScanner *scanner,
                                             const gchar *symbol);
```

Looks up a symbol in the current scope and return its value. If the symbol is not bound in the current scope, NULL is returned.

scanner : a GScanner.
symbol : the symbol to look up.
Returns : the value of *symbol* in the current scope, or NULL if *symbol* is not bound in the current scope.

---

g_scanner_warn ()

```
void        g_scanner_warn                  (GScanner *scanner,
                                             const gchar *format,
                                             ...);
```

Outputs a warning message, via the GScanner message handler.

scanner : a GScanner.
format : the message format. See the printf() documentation.
... : the parameters to insert into the format string.

---

g_scanner_error ()

```
void        g_scanner_error                 (GScanner *scanner,
                                             const gchar *format,
                                             ...);
```

Outputs an error message, via the GScanner message handler.

scanner : a GScanner.
format : the message format. See the printf() documentation.

`...` : the parameters to insert into the format string.

---

g_scanner_unexp_token ()

```
void        g_scanner_unexp_token           (GScanner *scanner,
                                             GTokenType expected_token,
                                             const gchar *identifier_spec,
                                             const gchar *symbol_spec,
                                             const gchar *symbol_name,
                                             const gchar *message,
                                             gint is_error);
```

Outputs a message through the scanner's msg_handler, resulting from an unexpected token in the input stream. Note that you should not call `g_scanner_peek_next_token()` followed by `g_scanner_unexp_token()` without an intermediate call to `g_scanner_get_next_token()`, as `g_scanner_unexp_token()` evaluates the scanner's current token (not the peeked token) to construct part of the message.

scanner : a GScanner.

expected_token : the expected token.

identifier_spec : a string describing how the scanner's user refers to identifiers (`NULL` defaults to "identifier"). This is used if *expected_token* is G_TOKEN_IDENTIFIER or G_TOKEN_IDENTIFIER_NULL.

symbol_spec : a string describing how the scanner's user refers to symbols (`NULL` defaults to "symbol"). This is used if *expected_token* is G_TOKEN_SYMBOL or any token value greater than G_TOKEN_LAST.

symbol_name : the name of the symbol, if the scanner's current token is a symbol.

message : a message string to output at the end of the warning/error, or `NULL`.

is_error : if `TRUE` it is output as an error. If `FALSE` it is output as a warning.

GScannerMsgFunc ()

```
void        (*GScannerMsgFunc)              (GScanner *scanner,
                                             gchar *message,
                                             gboolean error);
```

Specifies the type of the message handler function.

scanner : a GScanner.

message : the message.

error : `TRUE` if the message signals an error, `FALSE` if it signals a warning.

---

g_scanner_destroy ()


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

Frees all memory used by the GScanner.

scanner : a GScanner.

---

enum GTokenType

```
typedef enum
{
  G_TOKEN_EOF                   =   0,

  G_TOKEN_LEFT_PAREN            = '(',
  G_TOKEN_RIGHT_PAREN           = ')',
  G_TOKEN_LEFT_CURLY            = '{',
  G_TOKEN_RIGHT_CURLY           = '}',
  G_TOKEN_LEFT_BRACE            = '[',
  G_TOKEN_RIGHT_BRACE           = ']',
  G_TOKEN_EQUAL_SIGN            = '=',
  G_TOKEN_COMMA                 = ',',

  G_TOKEN_NONE                  = 256,

  G_TOKEN_ERROR,

  G_TOKEN_CHAR,
  G_TOKEN_BINARY,
  G_TOKEN_OCTAL,
  G_TOKEN_INT,
  G_TOKEN_HEX,
  G_TOKEN_FLOAT,
  G_TOKEN_STRING,

  G_TOKEN_SYMBOL,
  G_TOKEN_IDENTIFIER,
  G_TOKEN_IDENTIFIER_NULL,

  G_TOKEN_COMMENT_SINGLE,
  G_TOKEN_COMMENT_MULTI,
  G_TOKEN_LAST
} GTokenType;
```

The possible types of token returned from each `g_scanner_get_next_token()` call.

`G_TOKEN_EOF` the end of the file.

`G_TOKEN_LEFT_PAREN` a '(' character.

`G_TOKEN_LEFT_CURLY` a '{' character.

`G_TOKEN_RIGHT_CURLY` a '}' character.

---

union GTokenValue

```
union GTokenValue
{
```

```
  gpointer      v_symbol;
  gchar         *v_identifier;
  gulong        v_binary;
  gulong        v_octal;
  gulong        v_int;
  guint64       v_int64;
  gdouble       v_float;
  gulong        v_hex;
  gchar         *v_string;
  gchar         *v_comment;
  guchar        v_char;
  guint         v_error;
};
```

A union holding the value of the token.

---

enum GErrorType

```
typedef enum
{
  G_ERR_UNKNOWN,
  G_ERR_UNEXP_EOF,
  G_ERR_UNEXP_EOF_IN_STRING,
  G_ERR_UNEXP_EOF_IN_COMMENT,
  G_ERR_NON_DIGIT_IN_CONST,
  G_ERR_DIGIT_RADIX,
  G_ERR_FLOAT_RADIX,
  G_ERR_FLOAT_MALFORMED
} GErrorType;
```

The possible errors, used in the *v_error* field of GTokenValue, when the token is a G_TOKEN_ERROR.

G_ERR_UNKNOWN	unknown error.
G_ERR_UNEXP_EOF	unexpected end of file.
G_ERR_UNEXP_EOF_IN_STRING	unterminated string constant.
G_ERR_UNEXP_EOF_IN_COMMENT	unterminated comment.
G_ERR_NON_DIGIT_IN_CONST	non-digit character in a number.
G_ERR_DIGIT_RADIX	digit beyond radix in a number.
G_ERR_FLOAT_RADIX	non-decimal floating point number.
G_ERR_FLOAT_MALFORMED	malformed floating point number.

G_CSET_a_2_z

```
#define G_CSET_a_2_z    "abcdefghijklmnopqrstuvwxyz"
```

The set of lowercase ASCII alphabet characters. Used for specifying valid identifier characters in GScannerConfig.

G_CSET_A_2_Z

```
#define G_CSET_A_2_Z    "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
```

The set of uppercase ASCII alphabet characters. Used for specifying valid identifier characters in GScannerConfig.

G_CSET_DIGITS

```
#define G_CSET_DIGITS   "0123456789"
```

The set of digits. Used for specifying valid identifier characters in GScannerConfig.

G_CSET_LATINC

```
#define     G_CSET_LATINC
```

The set of uppercase ISO 8859-1 alphabet characters which are not ASCII characters. Used for specifying valid identifier characters in GScannerConfig.

G_CSET_LATINS

```
#define     G_CSET_LATINS
```

The set of lowercase ISO 8859-1 alphabet characters which are not ASCII characters. Used for specifying valid identifier characters in GScannerConfig.

---

g_scanner_add_symbol()

```
#define     g_scanner_add_symbol( scanner, symbol, value )
```

Warning

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

Adds a symbol to the default scope.

scanner : a GScanner.
symbol : the symbol to add.

value : the value of the symbol.

---

g_scanner_remove_symbol()

```
#define     g_scanner_remove_symbol( scanner, symbol )
```

Warning

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

Removes a symbol from the default scope.

scanner : a GScanner.
symbol : the symbol to remove.

---

g_scanner_foreach_symbol()

```
#define     g_scanner_foreach_symbol( scanner, func, data )
```

Warning

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

Calls a function for each symbol in the default scope.

scanner : a GScanner.
func : the function to call with each symbol.
data : data to pass to the function.

<< Miscellaneous Utility Functions **Automatic String Completion >>**

Automatic String Completion

Automatic String Completion — support for automatic completion using a group of target strings.

Synopsis

```
#include <glib.h>


            GCompletion;
GCompletion* g_completion_new               (GCompletionFunc func);
gchar*      (*GCompletionFunc)              (gpointer);
void        g_completion_add_items          (GCompletion *cmp,
                                             GList *items);
void        g_completion_remove_items       (GCompletion *cmp,
                                             GList *items);
void        g_completion_clear_items        (GCompletion *cmp);
GList*      g_completion_complete           (GCompletion *cmp,
                                             const gchar *prefix,
                                             gchar **new_prefix);
GList*      g_completion_complete_utf8      (GCompletion *cmp,
                                             const gchar *prefix,
                                             gchar **new_prefix);
void        g_completion_set_compare        (GCompletion *cmp,
                                             GCompletionStrncmpFunc strncmp_func
gint        (*GCompletionStrncmpFunc)       (const gchar *s1,
                                             const gchar *s2,
                                             gsize n);
void        g_completion_free               (GCompletion *cmp);
```

Description

GCompletion provides support for automatic completion of a string using any group of target strings. It is typically used for file name completion as is common in many UNIX shells.

A GCompletion is created using `g_completion_new()`. Target items are added and removed with `g_completion_add_items()`, `g_completion_remove_items()` and `g_completion_clear_items()`. A completion attempt is requested with `g_completion_complete()` or `g_completion_complete_utf8()`. When no longer needed, the GCompletion is freed with `g_completion_free()`.

Items in the completion can be simple strings (e.g. filenames), or pointers to arbitrary data structures. If data structures are used you must provide a GCompletionFunc in `g_completion_new()`, which retrieves the item's string from the data structure. You can change the way in which strings are compared by setting a different GCompletionStrncmpFunc in `g_completion_set_compare()`.

Details

GCompletion


```
typedef struct {
  GList* items;
  GCompletionFunc func;

  gchar* prefix;
  GList* cache;
  GCompletionStrncmpFunc strncmp_func;
} GCompletion;
```

The data structure used for automatic completion.

GList *items;	list of target items (strings or data structures).
GCompletionFunc func;	function which is called to get the string associated with a target item. It is NULL if the target items are strings.
gchar *prefix;	the last prefix passed to `g_completion_complete()` or `g_completion_complete_utf8()`.
GList *cache;	the list of items which begin with *prefix*.
GCompletionStrncmpFunc strncmp_func;	

---

g_completion_new ()

```
GCompletion* g_completion_new               (GCompletionFunc func);
```

Creates a new GCompletion.

func : the function to be called to return the string representing an item in the GCompletion, or NULL if strings are going to be used as the GCompletion items.

Returns : the new GCompletion.

GCompletionFunc ()

```
gchar*      (*GCompletionFunc)              (gpointer);
```

Specifies the type of the function passed to `g_completion_new()`. It should return the string corresponding to the given target item. This is used when you use data structures as GCompletion items.

Param1 : the completion item.

Returns : the string corresponding to the item.

---

g_completion_add_items ()

```
void        g_completion_add_items          (GCompletion *cmp,
```

```
                                        GList *items);
```

Adds items to the GCompletion.

cmp : the GCompletion.
items : the list of items to add.

---

g_completion_remove_items ()

```
void        g_completion_remove_items       (GCompletion *cmp,
                                             GList *items);
```

Removes items from a GCompletion.

cmp : the GCompletion.
items : the items to remove.

---

g_completion_clear_items ()

```
void        g_completion_clear_items        (GCompletion *cmp);
```

Removes all items from the GCompletion.

cmp : the GCompletion.

---

g_completion_complete ()

```
GList*      g_completion_complete           (GCompletion *cmp,
                                             const gchar *prefix,
                                             gchar **new_prefix);
```

Attempts to complete the string *prefix* using the GCompletion target items.

cmp : the GCompletion.
prefix : the prefix string, typically typed by the user, which is compared with each of the items.
new_prefix : if non-NULL, returns the longest prefix which is common to all items that matched *prefix*, or NULL if no items matched *prefix*. This string should be freed when no longer needed.
Returns : the list of items whose strings begin with *prefix*. This should not be changed.

---

g_completion_complete_utf8 ()


```
GList*      g_completion_complete_utf8      (GCompletion *cmp,
                                             const gchar *prefix,
                                             gchar **new_prefix);
```

Attempts to complete the string *prefix* using the GCompletion target items. In contrast to g_completion_complete(), this function returns the largest common prefix that is a valid UTF-8 string, omitting a possible common partial character.

You should use this function instead of g_completion_complete() if your items are UTF-8 strings.

cmp : the GCompletion
prefix : the prefix string, typically used by the user, which is compared with each of the items
new_prefix : if non-NULL, returns the longest prefix which is common to all items that matched *prefix*, or NULL if no items matched *prefix*. This string should be freed when no longer needed.
Returns : the list of items whose strings begin with *prefix*. This should not be changed.

Since 2.4

---

g_completion_set_compare ()

```
void        g_completion_set_compare        (GCompletion *cmp,
                                             GCompletionStrncmpFunc strncmp_func
```

Sets the function to use for string comparisons. The default string comparison function is strncmp ().

cmp : a GCompletion.
strncmp_func : the string comparison function.

GCompletionStrncmpFunc ()

```
gint        (*GCompletionStrncmpFunc)       (const gchar *s1,
                                             const gchar *s2,
                                             gsize n);
```

Specifies the type of the function passed to g_completion_set_compare(). This is used when you use strings as GCompletion items.

s1 : string to compare with *s2*.
s2 : string to compare with *s1*.
n : maximal number of bytes to compare.
Returns : an integer less than, equal to, or greater than zero if the first *n* bytes of *s1* is

found, respectively, to be less than, to match, or to be greater than the first *n* bytes of *s2*.

---

g_completion_free ()

```
void        g_completion_free               (GCompletion *cmp);
```

Frees all memory used by the GCompletion.

cmp : the GCompletion.

<< Lexical Scanner **Timers >>**

Timers

Timers — keep track of elapsed time.

Synopsis

```
#include <glib.h>


            GTimer;
GTimer*     g_timer_new                     (void);
void        g_timer_start                   (GTimer *timer);
void        g_timer_stop                    (GTimer *timer);
void        g_timer_continue                (GTimer *timer);
gdouble     g_timer_elapsed                 (GTimer *timer,
                                             gulong *microseconds);
void        g_timer_reset                   (GTimer *timer);
void        g_timer_destroy                 (GTimer *timer);
```

Description

GTimer records a start time, and counts microseconds elapsed since that time. This is done somewhat differently on different platforms, and can be tricky to get exactly right, so GTimer provides a portable/convenient interface.

Details

GTimer

```
typedef struct _GTimer GTimer;
```

Opaque datatype that records a start time.

---

g_timer_new ()

```
GTimer*     g_timer_new                     (void);
```

Creates a new timer, and starts timing (i.e. `g_timer_start()` is implicitly called for you).

Returns : a new GTimer.

---

g_timer_start ()


```
void        g_timer_start                   (GTimer *timer);
```

Marks a start time, so that future calls to `g_timer_elapsed()` will report the time since `g_timer_start()` was called. `g_timer_new()` automatically marks the start time, so no need to call `g_timer_start()` immediately after creating the timer.

timer : a GTimer.

---

g_timer_stop ()

```
void        g_timer_stop                    (GTimer *timer);
```

Marks an end time, so calls to `g_timer_elapsed()` will return the difference between this end time and the start time.

timer : a GTimer.

---

g_timer_continue ()

```
void        g_timer_continue                (GTimer *timer);
```

Resumes a timer that has previously been stopped with `g_timer_stop()`. `g_timer_stop()` must be called before using this function.

timer : a GTimer.

Since 2.4

---

g_timer_elapsed ()

```
gdouble     g_timer_elapsed                 (GTimer *timer,
                                             gulong *microseconds);
```

If *timer* has been started but not stopped, obtains the time since the timer was started. If *timer* has been stopped, obtains the elapsed time between the time it was started and the time it was stopped. The return value is the number of seconds elapsed, including any fractional part. The *microseconds* out parameter is essentially useless.

timer : a GTimer.
microseconds : fractional part of seconds elapsed, in microseconds (that is, the total number of microseconds elapsed, modulo 1000000)
Returns : seconds elapsed as a floating point value, including any fractional part.

---

g_timer_reset ()

```
void        g_timer_reset                   (GTimer *timer);
```

This function is useless; it's fine to call `g_timer_start()` on an already-started timer to reset the start time, so `g_timer_reset()` serves no purpose.

timer : a GTimer.

---

g_timer_destroy ()

```
void        g_timer_destroy                 (GTimer *timer);
```

Destroys a timer, freeing associated resources.

timer : a GTimer to destroy.

<< Automatic String Completion **Spawning Processes >>**

Spawning Processes

Spawning Processes — process launching with `fork()`/`exec()`.

Synopsis

```
#include <glib.h>


enum        GSpawnError;
#define     G_SPAWN_ERROR
enum        GSpawnFlags;
void        (*GSpawnChildSetupFunc)         (gpointer user_data);
gboolean    g_spawn_async_with_pipes        (const gchar *working_directory,
                                             gchar **argv,
                                             gchar **envp,
                                             GSpawnFlags flags,
                                             GSpawnChildSetupFunc child_setup,
                                             gpointer user_data,
                                             GPid *child_pid,
                                             gint *standard_input,
                                             gint *standard_output,
                                             gint *standard_error,
                                             GError **error);
gboolean    g_spawn_async                   (const gchar *working_directory,
                                             gchar **argv,
                                             gchar **envp,
                                             GSpawnFlags flags,
                                             GSpawnChildSetupFunc child_setup,
                                             gpointer user_data,
                                             GPid *child_pid,
                                             GError **error);
gboolean    g_spawn_sync                    (const gchar *working_directory,
                                             gchar **argv,
                                             gchar **envp,
                                             GSpawnFlags flags,
                                             GSpawnChildSetupFunc child_setup,
                                             gpointer user_data,
                                             gchar **standard_output,
                                             gchar **standard_error,
                                             gint *exit_status,
                                             GError **error);
gboolean    g_spawn_command_line_async      (const gchar *command_line,
                                             GError **error);
gboolean    g_spawn_command_line_sync       (const gchar *command_line,
                                             gchar **standard_output,
                                             gchar **standard_error,
                                             gint *exit_status,
                                             GError **error);
void        g_spawn_close_pid               (GPid pid);
```

Description

Details

enum GSpawnError

```
typedef enum
{
  G_SPAWN_ERROR_FORK,   /* fork failed due to lack of memory */
  G_SPAWN_ERROR_READ,   /* read or select on pipes failed */
  G_SPAWN_ERROR_CHDIR,  /* changing to working dir failed */
  G_SPAWN_ERROR_ACCES,  /* execv() returned EACCES */
  G_SPAWN_ERROR_PERM,   /* execv() returned EPERM */
  G_SPAWN_ERROR_2BIG,   /* execv() returned E2BIG */
  G_SPAWN_ERROR_NOEXEC, /* execv() returned ENOEXEC */
  G_SPAWN_ERROR_NAMETOOLONG, /* ""  "" ENAMETOOLONG */
  G_SPAWN_ERROR_NOENT,       /* ""  "" ENOENT */
  G_SPAWN_ERROR_NOMEM,       /* ""  "" ENOMEM */
  G_SPAWN_ERROR_NOTDIR,      /* ""  "" ENOTDIR */
  G_SPAWN_ERROR_LOOP,        /* ""  "" ELOOP   */
  G_SPAWN_ERROR_TXTBUSY,     /* ""  "" ETXTBUSY */
  G_SPAWN_ERROR_IO,          /* ""  "" EIO */
  G_SPAWN_ERROR_NFILE,       /* ""  "" ENFILE */
  G_SPAWN_ERROR_MFILE,       /* ""  "" EMFLE */
  G_SPAWN_ERROR_INVAL,       /* ""  "" EINVAL */
  G_SPAWN_ERROR_ISDIR,       /* ""  "" EISDIR */
  G_SPAWN_ERROR_LIBBAD,      /* ""  "" ELIBBAD */
  G_SPAWN_ERROR_FAILED       /* other fatal failure, error->message
                              * should explain
                              */
} GSpawnError;
```

Error codes returned by spawning processes.

`G_SPAWN_ERROR_FORK`	Fork failed due to lack of memory.
`G_SPAWN_ERROR_READ`	Read or select on pipes failed.
`G_SPAWN_ERROR_CHDIR`	Changing to working directory failed.
`G_SPAWN_ERROR_ACCES`	`execv()` returned `EACCES`.
`G_SPAWN_ERROR_PERM`	`execv()` returned `EPERM`.
`G_SPAWN_ERROR_2BIG`	`execv()` returned `E2BIG`.
`G_SPAWN_ERROR_NOEXEC`	`execv()` returned `ENOEXEC`.
`G_SPAWN_ERROR_NAMETOOLONG`	`execv()` returned `ENAMETOOLONG`.
`G_SPAWN_ERROR_NOENT`	`execv()` returned `ENOENT`.
`G_SPAWN_ERROR_NOMEM`	`execv()` returned `ENOMEM`.
`G_SPAWN_ERROR_NOTDIR`	`execv()` returned `ENOTDIR`.
`G_SPAWN_ERROR_LOOP`	`execv()` returned `ELOOP`.
`G_SPAWN_ERROR_TXTBUSY`	`execv()` returned `ETXTBUSY`.
`G_SPAWN_ERROR_IO`	`execv()` returned `EIO`.
`G_SPAWN_ERROR_NFILE`	`execv()` returned `ENFILE`.
`G_SPAWN_ERROR_MFILE`	`execv()` returned `EMFILE`.
`G_SPAWN_ERROR_INVAL`	`execv()` returned `EINVAL`.
`G_SPAWN_ERROR_ISDIR`	`execv()` returned `EISDIR`.
`G_SPAWN_ERROR_LIBBAD`	`execv()` returned `ELIBBAD`.
`G_SPAWN_ERROR_FAILED`	

Some other fatal failure, `error->message` should explain.

G_SPAWN_ERROR

```
#define G_SPAWN_ERROR g_spawn_error_quark ()
```

Error domain for spawning processes. Errors in this domain will be from the GSpawnError enumeration. See GError for information on error domains.

---

enum GSpawnFlags

```
typedef enum
{
  G_SPAWN_LEAVE_DESCRIPTORS_OPEN = 1 << 0,
  G_SPAWN_DO_NOT_REAP_CHILD      = 1 << 1,
  /* look for argv[0] in the path i.e. use execvp() */
  G_SPAWN_SEARCH_PATH            = 1 << 2,
  /* Dump output to /dev/null */
  G_SPAWN_STDOUT_TO_DEV_NULL     = 1 << 3,
  G_SPAWN_STDERR_TO_DEV_NULL     = 1 << 4,
  G_SPAWN_CHILD_INHERITS_STDIN   = 1 << 5,
  G_SPAWN_FILE_AND_ARGV_ZERO     = 1 << 6
} GSpawnFlags;
```

Flags passed to `g_spawn_sync()`, `g_spawn_async()` and `g_spawn_async_with_pipes()`.

`G_SPAWN_LEAVE_DESCRIPTORS_OPEN`	the parent's open file descriptors will be inherited by the child; otherwise all descriptors except stdin/stdout/stderr will be closed before calling `exec()` in the child.
`G_SPAWN_DO_NOT_REAP_CHILD`	the child will not be automatically reaped; you must call `waitpid()` or handle `SIGCHLD` yourself, or the child will become a zombie.
`G_SPAWN_SEARCH_PATH`	`argv[0]` need not be an absolute path, it will be looked for in the user's `PATH`.
`G_SPAWN_STDOUT_TO_DEV_NULL`	the child's standad output will be discarded, instead of going to the same location as the parent's standard output.
`G_SPAWN_STDERR_TO_DEV_NULL`	the child's standard error will be discarded.
`G_SPAWN_CHILD_INHERITS_STDIN`	the child will inherit the parent's standard input (by default, the child's standard input is attached to `/dev/null`).
`G_SPAWN_FILE_AND_ARGV_ZERO`	the first element of `argv` is the file to execute, while the remaining elements are the actual argument vector to pass to the file. Normally `g_spawn_async_with_pipes()` uses `argv[0]` as the file to execute, and passes all of `argv` to the child.

GSpawnChildSetupFunc ()

```
void        (*GSpawnChildSetupFunc)         (gpointer user_data);
```

Specifies the type of the setup function passed to `g_spawn_async()`, `g_spawn_sync()` and `g_spawn_async_with_pipes()`. It is called in the child after GLib has performed all the setup it plans to perform but before calling `exec()`. Obviously, actions taken in this function will only affect the child, not the parent.

`user_data` : user data to pass to the function.

---

g_spawn_async_with_pipes ()

```
gboolean    g_spawn_async_with_pipes        (const gchar *working_directory,
                                             gchar **argv,
                                             gchar **envp,
                                             GSpawnFlags flags,
                                             GSpawnChildSetupFunc child_setup,
                                             gpointer user_data,
                                             GPid *child_pid,
                                             gint *standard_input,
                                             gint *standard_output,
                                             gint *standard_error,
                                             GError **error);
```

Executes a child program asynchronously (your program will not block waiting for the child to exit). The child program is specified by the only argument that must be provided, *`argv`*. *`argv`* should be a `NULL`-terminated array of strings, to be passed as the argument vector for the child. The first string in *`argv`* is of course the name of the program to execute. By default, the name of the program must be a full path; the `PATH` shell variable will only be searched if you pass the `G_SPAWN_SEARCH_PATH` flag.

On Windows, the low-level child process creation API (`CreateProcess()`)doesn't use argument vectors, but a command line. The C runtime library's `spawn*()` family of functions (which `g_spawn_async_with_pipes()` eventually calls) paste the argument vector elements into a command line, and the C runtime startup code does a corresponding recostruction of an argument vector from the command line, to be passed to `main()`. Complications arise when you have argument vector elements that contain spaces of double quotes. The `spawn*()` functions don't do any quoting or escaping, but on the other hand the startup code does do unquoting and unescaping in order to enable receiving arguments with embedded spaces or double quotes. To work around this asymmetry, `g_spawn_async_with_pipes()` will do quoting and escaping on argument vector elements that need it before calling the C runtime `spawn()` function.

`envp` is a `NULL`-terminated array of strings, where each string has the form `KEY=VALUE`. This will become the child's environment. If *`envp`* is `NULL`, the child inherits its parent's environment.

`flags` should be the bitwise OR of any flags you want to affect the function's behavior. On Unix, the `G_SPAWN_DO_NOT_REAP_CHILD` means that the child will not be automatically reaped; you must call `waitpid()` or handle `SIGCHLD` yourself, or the child will become a zombie. On Windows, the flag means that a handle to the child will be returned *`child_pid`*. You must call `CloseHandle()` on it eventually (or exit the process), or the child processs will continue to take up some table space even after its death. Quite similar to zombies on Unix, actually.

G_SPAWN_LEAVE_DESCRIPTORS_OPEN means that the parent's open file descriptors will be inherited by the child; otherwise all descriptors except stdin/stdout/stderr will be closed before calling exec() in the child. G_SPAWN_SEARCH_PATH means that argv[0] need not be an absolute path, it will be looked for in the user's PATH. G_SPAWN_STDOUT_TO_DEV_NULL means that the child's standard output will be discarded, instead of going to the same location as the parent's standard output. If you use this flag, *standard_output* must be NULL. G_SPAWN_STDERR_TO_DEV_NULL means that the child's standard error will be discarded, instead of going to the same location as the parent's standard error. If you use this flag, *standard_error* must be NULL. G_SPAWN_CHILD_INHERITS_STDIN means that the child will inherit the parent's standard input (by default, the child's standard input is attached to /dev/null). If you use this flag, *standard_input* must be NULL. G_SPAWN_FILE_AND_ARGV_ZERO means that the first element of *argv* is the file to execute, while the remaining elements are the actual argument vector to pass to the file. Normally g_spawn_async_with_pipes() uses *argv*[0] as the file to execute, and passes all of *argv* to the child.

child_setup and *user_data* are a function and user data. On POSIX platforms, the function is called in the child after GLib has performed all the setup it plans to perform (including creating pipes, closing file descriptors, etc.) but before calling exec(). That is, *child_setup* is called just before calling exec() in the child. Obviously actions taken in this function will only affect the child, not the parent. On Windows, there is no separate fork() and exec() functionality. Child processes are created and run right away with one API call, CreateProcess(). *child_setup* is called in the parent process just before creating the child process. You should carefully consider what you do in *child_setup* if you intend your software to be portable to Windows.

If non-NULL, *child_pid* will on Unix be filled with the child's process ID. You can use the process ID to send signals to the child, or to waitpid() if you specified the G_SPAWN_DO_NOT_REAP_CHILD flag. On Windows, *child_pid* will be filled with a handle to the child process only if you specified the G_SPAWN_DO_NOT_REAP_CHILD flag. You can then access the child process using the Win32 API, for example wait for its termination with the WaitFor*() functions, or examine its exit code with GetExitCodeProcess(). You should close the handle with CloseHandle() when you no longer need it.

If non-NULL, the *standard_input*, *standard_output*, *standard_error* locations will be filled with file descriptors for writing to the child's standard input or reading from its standard output or standard error. The caller of g_spawn_async_with_pipes() must close these file descriptors when they are no longer in use. If these parameters are NULL, the corresponding pipe won't be created.

If *standard_input* is NULL, the child's standard input is attached to /dev/null unless G_SPAWN_CHILD_INHERITS_STDIN is set.

If *standard_error* is NULL, the child's standard error goes to the same location as the parent's standard error unless G_SPAWN_STDERR_TO_DEV_NULL is set.

If *standard_output* is NULL, the child's standard output goes to the same location as the parent's standard output unless G_SPAWN_STDOUT_TO_DEV_NULL is set.

error can be NULL to ignore errors, or non-NULL to report errors. If an error is set, the function returns FALSE. Errors are reported even if they occur in the child (for example if the executable in argv[0] is not found). Typically the message field of returned errors should be displayed to users. Possible errors are those from the G_SPAWN_ERROR domain.

If an error occurs, *child_pid*, *standard_input*, *standard_output*, and *standard_error* will not be filled with valid values.

If *child_pid* is not NULL and an error does not occur then the returned pid must be closed using g_spawn_close_pid().

working_directory :	child's current working directory, or NULL to inherit parent's
argv :	child's argument vector
envp :	child's environment, or NULL to inherit parent's
flags :	flags from GSpawnFlags
child_setup :	function to run in the child just before exec()
user_data :	user data for *child_setup*
child_pid :	return location for child process ID, or NULL
standard_input :	return location for file descriptor to write to child's stdin, or NULL
standard_output :	return location for file descriptor to read child's stdout, or NULL
standard_error :	return location for file descriptor to read child's stderr, or NULL
error :	return location for error
Returns :	TRUE on success, FALSE if an error was set

---

g_spawn_async ()

```
gboolean    g_spawn_async                   (const gchar *working_directory,
                                             gchar **argv,
                                             gchar **envp,
                                             GSpawnFlags flags,
                                             GSpawnChildSetupFunc child_setup,
                                             gpointer user_data,
                                             GPid *child_pid,
                                             GError **error);
```

See g_spawn_async_with_pipes() for a full description; this function simply calls the g_spawn_async_with_pipes() without any pipes.

working_directory :	child's current working directory, or NULL to inherit parent's
argv :	child's argument vector
envp :	child's environment, or NULL to inherit parent's
flags :	flags from GSpawnFlags
child_setup :	function to run in the child just before exec()
user_data :	user data for *child_setup*
child_pid :	return location for child process ID, or NULL
error :	return location for error
Returns :	TRUE on success, FALSE if error is set

---

g_spawn_sync ()

```
gboolean    g_spawn_sync                    (const gchar *working_directory,
                                             gchar **argv,
                                             gchar **envp,
                                             GSpawnFlags flags,
```

```
                                        GSpawnChildSetupFunc child_setup,
                                        gpointer user_data,
                                        gchar **standard_output,
                                        gchar **standard_error,
                                        gint *exit_status,
                                        GError **error);
```

Executes a child synchronously (waits for the child to exit before returning). All output from the child is stored in *standard_output* and *standard_error*, if those parameters are non-NULL. If *exit_status* is non-NULL, the exit status of the child is stored there as it would be returned by waitpid(); standard UNIX macros such as WIFEXITED() and WEXITSTATUS() must be used to evaluate the exit status. If an error occurs, no data is returned in *standard_output*, *standard_error*, or *exit_status*.

This function calls g_spawn_async_with_pipes() internally; see that function for full details on the other parameters.

working_directory : child's current working directory, or NULL to inherit parent's
argv : child's argument vector
envp : child's environment, or NULL to inherit parent's
flags : flags from GSpawnFlags
child_setup : function to run in the child just before exec()
user_data : user data for *child_setup*
standard_output : return location for child output
standard_error : return location for child error messages
exit_status : child exit status, as returned by waitpid()
error : return location for error
Returns : TRUE on success, FALSE if an error was set.

---

g_spawn_command_line_async ()

```
gboolean    g_spawn_command_line_async      (const gchar *command_line,
                                             GError **error);
```

A simple version of g_spawn_async() that parses a command line with g_shell_parse_argv() and passes it to g_spawn_async(). Runs a command line in the background. Unlike g_spawn_async(), the G_SPAWN_SEARCH_PATH flag is enabled, other flags are not. Note that G_SPAWN_SEARCH_PATH can have security implications, so consider using g_spawn_async() directly if appropriate. Possible errors are those from g_shell_parse_argv() and g_spawn_async().

The same concerns on Windows apply as for g_spawn_command_line_sync().

command_line : a command line
error : return location for errors
Returns : TRUE on success, FALSE if error is set.

---

g_spawn_command_line_sync ()


```
gboolean    g_spawn_command_line_sync       (const gchar *command_line,
                                             gchar **standard_output,
                                             gchar **standard_error,
                                             gint *exit_status,
                                             GError **error);
```

A simple version of g_spawn_sync() with little-used parameters removed, taking a command line instead of an argument vector. See g_spawn_sync() for full details. *command_line* will be parsed by g_shell_parse_argv(). Unlike g_spawn_sync(), the G_SPAWN_SEARCH_PATH flag is enabled. Note that G_SPAWN_SEARCH_PATH can have security implications, so consider using g_spawn_sync() directly if appropriate. Possible errors are those from g_spawn_sync() and those from g_shell_parse_argv().

On Windows, please note the implications of g_shell_parse_argv() parsing *command_line*. Space is a separator, and backslashes are special. Thus you cannot simply pass a *command_line* containing canonical Windows paths, like "c:\\program files\\app\\app.exe", as the backslashes will be eaten, and the space will act as a separator. You need to enclose such paths with single quotes, like "'c:\\program files\\app\\app.exe' 'e:\\folder\\argument.txt'".

command_line : a command line
standard_output : return location for child output
standard_error : return location for child errors
exit_status : return location for child exit status
error : return location for errors
Returns : TRUE on success, FALSE if an error was set

---

g_spawn_close_pid ()

```
void        g_spawn_close_pid               (GPid pid);
```

On some platforms, notably WIN32, the GPid type represents a resource which must be closed to prevent resource leaking. g_spawn_close_pid() is provided for this purpose. It should be used on all platforms, even though it doesn't do anything under UNIX.

pid : The process identifier to close

<< Timers **File Utilities >>**

File Utilities

File Utilities — various file-related functions.

Synopsis

```
#include <glib.h>
#include <glib/gstdio.h>


enum        GFileError;
#define     G_FILE_ERROR
enum        GFileTest;
GFileError  g_file_error_from_errno         (gint err_no);
gboolean    g_file_get_contents             (const gchar *filename,
                                             gchar **contents,
                                             gsize *length,
                                             GError **error);
gboolean    g_file_test                     (const gchar *filename,
                                             GFileTest test);
gint        g_mkstemp                       (gchar *tmpl);
gint        g_file_open_tmp                 (const gchar *tmpl,
                                             gchar **name_used,
                                             GError **error);
gchar*      g_file_read_link                (const gchar *filename,
                                             GError **error);

            GDir;
GDir*       g_dir_open                      (const gchar *path,
                                             guint flags,
                                             GError **error);
G_CONST_RETURN gchar* g_dir_read_name       (GDir *dir);
void        g_dir_rewind                    (GDir *dir);
void        g_dir_close                     (GDir *dir);

int         g_open                          (const gchar *filename,
                                             int flags,
                                             int mode);
int         g_rename                        (const gchar *oldfilename,
                                             const gchar *newfilename);
int         g_mkdir                         (const gchar *filename,
                                             int mode);
int         g_stat                          (const gchar *filename,
                                             struct stat *buf);
int         g_lstat                         (const gchar *filename,
                                             struct stat *buf);
int         g_unlink                        (const gchar *filename);
int         g_remove                        (const gchar *filename);
FILE*       g_fopen                         (const gchar *filename,
                                             const gchar *mode);
FILE*       g_freopen                       (const gchar *filename,
                                             const gchar *mode,
                                             FILE *stream);
```


Description

There is a group of functions which wrap the common POSIX functions dealing with filenames (`g_open()`, `g_rename()`, `g_mkdir()`, `g_stat()`, `g_unlink()`, `g_remove()`, `g_fopen()`, `g_freopen()`). The point of these wrappers is to make it possible to handle file names with any Unicode characters in them on Windows without having to use ifdefs and the wide character API in the application code.

The pathname argument should be in the GLib file name encoding. On POSIX this is the actual on-disk encoding which might correspond to the locale settings of the process (or the `G_FILENAME_ENCODING` environment variable), or not.

On Windows the GLib file name encoding is UTF-8. Note that the Microsoft C library does not use UTF-8, but has separate APIs for current system code page and wide characters (UTF-16). The GLib wrappers call the wide character API if present (on modern Windows systems), otherwise convert to/from the system code page.

Another group of functions allows to open and read directories in the GLib file name encoding. These are `g_dir_open()`, `g_dir_read_name()`, `g_dir_rewind()`, `g_dir_close()`.

Details

enum GFileError

```
typedef enum
{
  G_FILE_ERROR_EXIST,
  G_FILE_ERROR_ISDIR,
  G_FILE_ERROR_ACCES,
  G_FILE_ERROR_NAMETOOLONG,
  G_FILE_ERROR_NOENT,
  G_FILE_ERROR_NOTDIR,
  G_FILE_ERROR_NXIO,
  G_FILE_ERROR_NODEV,
  G_FILE_ERROR_ROFS,
  G_FILE_ERROR_TXTBSY,
  G_FILE_ERROR_FAULT,
  G_FILE_ERROR_LOOP,
  G_FILE_ERROR_NOSPC,
  G_FILE_ERROR_NOMEM,
  G_FILE_ERROR_MFILE,
  G_FILE_ERROR_NFILE,
  G_FILE_ERROR_BADF,
  G_FILE_ERROR_INVAL,
  G_FILE_ERROR_PIPE,
  G_FILE_ERROR_AGAIN,
  G_FILE_ERROR_INTR,
  G_FILE_ERROR_IO,
  G_FILE_ERROR_PERM,
  G_FILE_ERROR_NOSYS,
  G_FILE_ERROR_FAILED
} GFileError;
```

Values corresponding to `errno` codes returned from file operations on UNIX. Unlike `errno` codes, GFileError values are available on all systems, even Windows. The exact meaning of each code depends on what sort of file operation you were performing; the UNIX documentation gives more

details. The following error code descriptions come from the GNU C Library manual, and are under the copyright of that manual.

It's not very portable to make detailed assumptions about exactly which errors will be returned from a given operation. Some errors don't occur on some systems, etc., sometimes there are subtle differences in when a system will report a given error, etc.

`G_FILE_ERROR_EXIST`	Operation not permitted; only the owner of the file (or other resource) or processes with special privileges can perform the operation.
`G_FILE_ERROR_ISDIR`	File is a directory; you cannot open a directory for writing, or create or remove hard links to it.
`G_FILE_ERROR_ACCES`	Permission denied; the file permissions do not allow the attempted operation.
`G_FILE_ERROR_NAMETOOLONG`	Filename too long.
`G_FILE_ERROR_NOENT`	No such file or directory. This is a "file doesn't exist" error for ordinary files that are referenced in contexts where they are expected to already exist.
`G_FILE_ERROR_NOTDIR`	A file that isn't a directory was specified when a directory is required.
`G_FILE_ERROR_NXIO`	No such device or address. The system tried to use the device represented by a file you specified, and it couldn't find the device. This can mean that the device file was installed incorrectly, or that the physical device is missing or not correctly attached to the computer.
`G_FILE_ERROR_NODEV`	This file is of a type that doesn't support mapping.
`G_FILE_ERROR_ROFS`	The directory containing the new link can't be modified because it's on a read-only file system.
`G_FILE_ERROR_TXTBSY`	Text file busy.
`G_FILE_ERROR_FAULT`	You passed in a pointer to bad memory. (GLib won't reliably return this, don't pass in pointers to bad memory.)
`G_FILE_ERROR_LOOP`	Too many levels of symbolic links were encountered in looking up a file name. This often indicates a cycle of symbolic links.
`G_FILE_ERROR_NOSPC`	No space left on device; write operation on a file failed because the disk is full.
`G_FILE_ERROR_NOMEM`	No memory available. The system cannot allocate more virtual memory because its capacity is full.
`G_FILE_ERROR_MFILE`	The current process has too many files open and can't open any more. Duplicate descriptors do count toward this limit.
`G_FILE_ERROR_NFILE`	There are too many distinct file openings in the entire system.
`G_FILE_ERROR_BADF`	Bad file descriptor; for example, I/O on a descriptor that has been closed or reading from a descriptor open only for writing (or vice versa).
`G_FILE_ERROR_INVAL`	Invalid argument. This is used to indicate various kinds of problems with passing the wrong argument to a library function.
`G_FILE_ERROR_PIPE`	Broken pipe; there is no process reading from the other end of a pipe. Every library function that returns this error code also generates a `SIGPIPE' signal; this signal terminates the program if not handled or blocked. Thus, your program will never actually see this code unless it has handled or blocked `SIGPIPE'.
`G_FILE_ERROR_AGAIN`	Resource temporarily unavailable; the call might work if you try again later.
`G_FILE_ERROR_INTR`	Interrupted function call; an asynchronous signal occurred and prevented completion of the call. When this happens, you should try the call again.
`G_FILE_ERROR_IO`	Input/output error; usually used for physical read or write errors. i.e. the disk or other physical device hardware is returning errors.
`G_FILE_ERROR_PERM`	Operation not permitted; only the owner of the file (or other resource) or processes with special privileges can perform the operation.
`G_FILE_ERROR_NOSYS`	Function not implemented; this indicates that the system is missing some functionality.
`G_FILE_ERROR_FAILED`	Does not correspond to a UNIX error code; this is the standard "failed for unspecified reason" error code present in all GError error code enumerations. Returned if no specific code applies.

G_FILE_ERROR

```
#define G_FILE_ERROR g_file_error_quark ()
```

Error domain for file operations. Errors in this domain will be from the GFileError enumeration. See GError for information on error domains.

enum GFileTest

```
typedef enum
{
  G_FILE_TEST_IS_REGULAR    = 1 << 0,
  G_FILE_TEST_IS_SYMLINK    = 1 << 1,
  G_FILE_TEST_IS_DIR        = 1 << 2,
  G_FILE_TEST_IS_EXECUTABLE = 1 << 3,
  G_FILE_TEST_EXISTS        = 1 << 4
} GFileTest;
```

A test to perform on a file using `g_file_test()`.

`G_FILE_TEST_IS_REGULAR`	`TRUE` if the file is a regular file (not a symlink or directory)
`G_FILE_TEST_IS_SYMLINK`	`TRUE` if the file is a symlink.
`G_FILE_TEST_IS_DIR`	`TRUE` if the file is a directory.
`G_FILE_TEST_IS_EXECUTABLE`	`TRUE` if the file is executable.
`G_FILE_TEST_EXISTS`	`TRUE` if the file exists. It may or may not be a regular file.

g_file_error_from_errno ()

```
GFileError  g_file_error_from_errno         (gint err_no);
```

Gets a GFileError constant based on the passed-in *errno*. For example, if you pass in EEXIST this function returns G_FILE_ERROR_EXIST. Unlike *errno* values, you can portably assume that all GFileError values will exist.

Normally a GFileError value goes into a GError returned from a function that manipulates files. So you would use g_file_error_from_errno() when constructing a GError.

err_no : an "errno" value
Returns : GFileError corresponding to the given *errno*

g_file_get_contents ()

```
gboolean    g_file_get_contents             (const gchar *filename,
                                             gchar **contents,
                                             gsize *length,
                                             GError **error);
```

Reads an entire file into allocated memory, with good error checking. If *error* is set, FALSE is returned, and *contents* is set to NULL. If TRUE is returned, *error* will not be set, and *contents* will be set to the file contents. The string stored in *contents* will be nul-terminated, so for text files you can pass NULL for the *length* argument. The error domain is G_FILE_ERROR. Possible error codes are those in the GFileError enumeration.

filename : name of a file to read contents from, in the GLib file name encoding
contents : location to store an allocated string
length : location to store length in bytes of the contents
error : return location for a GError
Returns : TRUE on success, FALSE if error is set

g_file_test ()

```
gboolean    g_file_test                     (const gchar *filename,
                                             GFileTest test);
```

Returns TRUE if any of the tests in the bitfield *test* are TRUE. For example, (G_FILE_TEST_EXISTS | G_FILE_TEST_IS_DIR) will return TRUE if the file exists; the check whether it's a directory doesn't matter since the existence test is TRUE. With the current set of available tests, there's no point passing in more than one test at a time.

Apart from G_FILE_TEST_IS_SYMLINK all tests follow symbolic links, so for a symbolic link to a regular file g_file_test() will return TRUE for both G_FILE_TEST_IS_SYMLINK and

G_FILE_TEST_IS_REGULAR.

Note, that for a dangling symbolic link g_file_test() will return TRUE for G_FILE_TEST_IS_SYMLINK and FALSE for all other flags.

You should never use g_file_test() to test whether it is safe to perform an operaton, because there is always the possibility of the condition changing before you actually perform the operation. For example, you might think you could use G_FILE_TEST_IS_SYMLINK to know whether it is is safe to write to a file without being tricked into writing into a different location. It doesn't work!

```
/* DON'T DO THIS */
 if (!g_file_test (filename, G_FILE_TEST_IS_SYMLINK)) {
   fd = g_open (filename, O_WRONLY);
   /* write to fd */
 }
```

Another thing to note is that G_FILE_TEST_EXISTS and G_FILE_TEST_IS_EXECUTABLE are implemented using the access() system call. This usually doesn't matter, but if your program is setuid or setgid it means that these tests will give you the answer for the real user ID and group ID , rather than the effective user ID and group ID.

filename : a filename to test
test : bitfield of GFileTest flags
Returns : whether a test was TRUE

g_mkstemp ()

```
gint        g_mkstemp                       (gchar *tmpl);
```

Opens a temporary file. See the mkstemp() documentation on most UNIX-like systems. This is a portability wrapper, which simply calls mkstemp() on systems that have it, and implements it in GLib otherwise.

The parameter is a string that should match the rules for mkstemp(), i.e. end in "XXXXXX". The X string will be modified to form the name of a file that didn't exist. The string should be in the GLib file name encoding. Most importantly, on Windows it should be in UTF-8.

tmpl : template filename
Returns : A file handle (as from open()) to the file opened for reading and writing. The file is opened in binary mode on platforms where there is a difference. The file handle should be closed with close(). In case of errors, -1 is returned.

g_file_open_tmp ()

```
gint        g_file_open_tmp                 (const gchar *tmpl,
                                             gchar **name_used,
                                             GError **error);
```

Opens a file for writing in the preferred directory for temporary files (as returned by g_get_tmp_dir ()).

tmpl should be a string in the GLib file name encoding ending with six 'X' characters, as the parameter to g_mkstemp() (or mkstemp()). However, unlike these functions, the template should only be a basename, no directory components are allowed. If template is NULL, a default template is used.

Note that in contrast to g_mkstemp() (and mkstemp()) *tmpl* is not modified, and might thus be a read-only literal string.

The actual name used is returned in *name_used* if non-NULL. This string should be freed with g_free () when not needed any longer. The returned name is in the GLib file name encoding.

tmpl : Template for file name, as in g_mkstemp(), basename only
name_used : location to store actual name used
error : return location for a GError
Returns : A file handle (as from open()) to the file opened for reading and writing. The file is opened in binary mode on platforms where there is a difference. The file handle should be closed with close(). In case of errors, -1 is returned and *error* will be set.

---

g_file_read_link ()

```
gchar*      g_file_read_link                (const gchar *filename,
                                             GError **error);
```

Reads the contents of the symbolic link *filename* like the POSIX readlink() function. The returned string is in the encoding used for filenames. Use g_filename_to_utf8() to convert it to UTF-8.

filename : the symbolic link
error : return location for a GError
Returns : A newly allocated string with the contents of the symbolic link, or NULL if an error occurred.

Since 2.4

GDir

```
typedef struct _GDir GDir;
```

An opaque structure representing an opened directory.

---

g_dir_open ()

```
GDir*       g_dir_open                      (const gchar *path,
                                             guint flags,
                                             GError **error);
```

Opens a directory for reading. The names of the files in the directory can then be retrieved using g_dir_read_name().

path : the path to the directory you are interested in. On Unix in the on-disk encoding. On Windows in UTF-8
flags : Currently must be set to 0. Reserved for future use.
error : return location for a GError, or NULL. If non-NULL, an error will be set if and only if g_dir_open_fails.
Returns : a newly allocated GDir on success, NULL on failure. If non-NULL, you must free the result with g_dir_close() when you are finished with it.

---

g_dir_read_name ()

```
G_CONST_RETURN gchar* g_dir_read_name       (GDir *dir);
```

Retrieves the name of the next entry in the directory. The '.' and '..' entries are omitted. On Windows, the returned name is in UTF-8. On Unix, it is in the on-disk encoding.

dir : a GDir* created by g_dir_open()
Returns : The entry's name or NULL if there are no more entries. The return value is owned by GLib and must not be modified or freed.

---

g_dir_rewind ()

```
void        g_dir_rewind                    (GDir *dir);
```

Resets the given directory. The next call to g_dir_read_name() will return the first entry again.

dir : a GDir* created by g_dir_open()

---

g_dir_close ()

```
void        g_dir_close                     (GDir *dir);
```

Closes the directory and deallocates all related resources.

dir : a GDir* created by g_dir_open()

g_open ()

```
int         g_open                          (const gchar *filename,
                                             int flags,
                                             int mode);
```

A wrapper for the POSIX `open()` function. The `open()` function is used to convert a pathname into a file descriptor. Note that on POSIX systems file descriptors are implemented by the operating system. On Windows, it's the C library that implements `open()` and file descriptors. The actual Windows API for opening files is something different.

See the C library manual for more details about `open()`.

`filename` : a pathname in the GLib file name encoding
`flags` : as in `open()`
`mode` : as in `open()`
Returns : a new file descriptor, or -1 if an error occurred. The return value can be used exactly like the return value from `open()`.

Since 2.6

---

g_rename ()

```
int         g_rename                        (const gchar *oldfilename,
                                             const gchar *newfilename);
```

A wrapper for the POSIX `rename()` function. The `rename()` function renames a file, moving it between directories if required.

See the C library manual for more details about `rename()`.

`oldfilename` : a pathname in the GLib file name encoding
`newfilename` : a pathname in the GLib file name encoding
Returns : 0 if the renaming succeeded, -1 if an error occurred

Since 2.6

---

g_mkdir ()

```
int         g_mkdir                         (const gchar *filename,
                                             int mode);
```

A wrapper for the POSIX `mkdir()` function. The `mkdir()` function attempts to create a directory

with the given name and permissions.

See the C library manual for more details about `mkdir()`.

`filename` : a pathname in the GLib file name encoding
`mode` : permissions to use for the newly created directory
Returns : 0 if the directory was successfully created, -1 if an error occurred

Since 2.6

---

g_stat ()

```
int         g_stat                          (const gchar *filename,
                                             struct stat *buf);
```

A wrapper for the POSIX `stat()` function. The `stat()` function returns information about a file.

See the C library manual for more details about `stat()`.

`filename` : a pathname in the GLib file name encoding
`buf` : a pointer to a stat struct, which will be filled with the file information
Returns : 0 if the information was successfully retrieved, -1 if an error occurred

Since 2.6

---

g_lstat ()

```
int         g_lstat                         (const gchar *filename,
                                             struct stat *buf);
```

A wrapper for the POSIX `lstat()` function. The `lstat()` function is like `stat()` except that in the case of symbolic links, it returns information about the symbolic link itself and not the file that it refers to. If the system does not support symbolic links `g_lstat()` is identical to `g_stat()`.

See the C library manual for more details about `lstat()`.

`filename` : a pathname in the GLib file name encoding
`buf` : a pointer to a stat struct, which will be filled with the file information
Returns : 0 if the information was successfully retrieved, -1 if an error occurred

Since 2.6

---

g_unlink ()

```
int         g_unlink                        (const gchar *filename);
```

A wrapper for the POSIX `unlink()` function. The `unlink()` function deletes a name from the filesystem. If this was the last link to the file and no processes have it opened, the diskspace occupied by the file is freed.

See the C library manual for more details about `unlink()`.

`filename` : a pathname in the GLib file name encoding
Returns : 0 if the directory was successfully created, -1 if an error occurred

Since 2.6

---

g_remove ()

```
int         g_remove                        (const gchar *filename);
```

A wrapper for the POSIX `remove()` function. The `remove()` function deletes a name from the filesystem. It calls `unlink()` for files and `rmdir()` for directories.

See the C library manual for more details about `remove()`.

`filename` : a pathname in the GLib file name encoding
Returns : 0 if the directory was successfully created, -1 if an error occurred

Since 2.6

---

g_fopen ()

```
FILE*       g_fopen                         (const gchar *filename,
                                             const gchar *mode);
```

A wrapper for the POSIX `fopen()` function. The `fopen()` function opens a file and associates a new stream with it.

See the C library manual for more details about `fopen()`.

`filename` : a pathname in the GLib file name encoding
`mode` : a string describing the mode in which the file should be opened
Returns : A <typename>FILE</typename> pointer if the file was successfully opened, or `NULL` if an error occurred

Since 2.6

---

g_freopen ()

```
FILE*       g_freopen                       (const gchar *filename,
                                             const gchar *mode,
                                             FILE *stream);
```

A wrapper for the POSIX `freopen()` function. The `freopen()` function opens a file and associates it with an existing stream.

See the C library manual for more details about `freopen()`.

`filename` : a pathname in the GLib file name encoding
`mode` : a string describing the mode in which the file should be opened
`stream` : an existing stream which will be reused, or `NULL`
Returns : A <typename>FILE</typename> pointer if the file was successfully opened, or `NULL` if an error occurred.

Since 2.6

<< Spawning Processes **Shell-related Utilities >>**

Shell-related Utilities

Shell-related Utilities — shell-like commandline handling.

Synopsis

```
#include <glib.h>


enum        GShellError;
#define     G_SHELL_ERROR
gboolean    g_shell_parse_argv              (const gchar *command_line,
                                             gint *argcp,
                                             gchar ***argvp,
                                             GError **error);
gchar*      g_shell_quote                   (const gchar *unquoted_string);
gchar*      g_shell_unquote                 (const gchar *quoted_string,
                                             GError **error);
```

Description

Details

enum GShellError

```
typedef enum
{
  /* mismatched or otherwise mangled quoting */
  G_SHELL_ERROR_BAD_QUOTING,
  /* string to be parsed was empty */
  G_SHELL_ERROR_EMPTY_STRING,
  G_SHELL_ERROR_FAILED
} GShellError;
```

Error codes returned by shell functions.

G_SHELL_ERROR_BAD_QUOTING	Mismatched or otherwise mangled quoting.
G_SHELL_ERROR_EMPTY_STRING	String to be parsed was empty.
G_SHELL_ERROR_FAILED	Some other error.

G_SHELL_ERROR

```
#define G_SHELL_ERROR g_shell_error_quark ()
```


Error domain for shell functions. Errors in this domain will be from the GShellError enumeration. See GError for information on error domains.

---

g_shell_parse_argv ()

```
gboolean    g_shell_parse_argv              (const gchar *command_line,
                                             gint *argcp,
                                             gchar ***argvp,
                                             GError **error);
```

Parses a command line into an argument vector, in much the same way the shell would, but without many of the expansions the shell would perform (variable expansion, globs, operators, filename expansion, etc. are not supported). The results are defined to be the same as those you would get from a UNIX98 /bin/sh, as long as the input contains none of the unsupported shell expansions. If the input does contain such expansions, they are passed through literally. Possible errors are those from the G_SHELL_ERROR domain. Free the returned vector with `g_strfreev()`.

command_line :	command line to parse
argcp :	return location for number of args
argvp :	return location for array of args
error :	return location for error
Returns :	TRUE on success, FALSE if error set

---

g_shell_quote ()

```
gchar*      g_shell_quote                   (const gchar *unquoted_string);
```

Quotes a string so that the shell (/bin/sh) will interpret the quoted string to mean *unquoted_string*. If you pass a filename to the shell, for example, you should first quote it with this function. The return value must be freed with `g_free()`. The quoting style used is undefined (single or double quotes may be used).

unquoted_string :	a literal string
Returns :	quoted string

---

g_shell_unquote ()

```
gchar*      g_shell_unquote                 (const gchar *quoted_string,
                                             GError **error);
```

Unquotes a string as the shell (/bin/sh) would. Only handles quotes; if a string contains file globs, arithmetic operators, variables, backticks, redirections, or other special-to-the-shell features, the result will be different from the result a real shell would produce (the variables, backticks, etc. will be passed through literally instead of being expanded). This function is guaranteed to succeed if applied to the result of `g_shell_quote()`. If it fails, it returns NULL and sets the error. The

quoted_string need not actually contain quoted or escaped text; g_shell_unquote() simply goes through the string and unquotes/unescapes anything that the shell would. Both single and double quotes are handled, as are escapes including escaped newlines. The return value must be freed with g_free(). Possible errors are in the G_SHELL_ERROR domain.

Shell quoting rules are a bit strange. Single quotes preserve the literal string exactly. escape sequences are not allowed; not even \' - if you want a ' in the quoted text, you have to do something like 'foo'\"bar'. Double quotes allow $, `, ", \, and newline to be escaped with backslash. Otherwise double quotes preserve things literally.

quoted_string : shell-quoted string
error : error return location or NULL
Returns : an unquoted string

<< File Utilities **Commandline option parser >>**

Commandline option parser

Commandline option parser — parses commandline options

Synopsis

```
#include <glib.h>


enum        GOptionError;
#define     G_OPTION_ERROR
gboolean    (*GOptionArgFunc)               (const gchar *option_name,
                                             const gchar *value,
                                             gpointer data,
                                             GError **error);
            GOptionContext;
GOptionContext* g_option_context_new        (const gchar *parameter_string);
void        g_option_context_free           (GOptionContext *context);
gboolean    g_option_context_parse          (GOptionContext *context,
                                             gint *argc,
                                             gchar ***argv,
                                             GError **error);
void        g_option_context_set_help_enabled
                                            (GOptionContext *context,
                                             gboolean help_enabled);
gboolean    g_option_context_get_help_enabled
                                            (GOptionContext *context);
void        g_option_context_set_ignore_unknown_options
                                            (GOptionContext *context,
                                             gboolean ignore_unknown);
gboolean    g_option_context_get_ignore_unknown_options
                                            (GOptionContext *context);
enum        GOptionArg;
enum        GOptionFlags;
#define     G_OPTION_REMAINING
            GOptionEntry;
void        g_option_context_add_main_entries
                                            (GOptionContext *context,
                                             const GOptionEntry *entries,
                                             const gchar *translation_domain);
            GOptionGroup;
void        g_option_context_add_group      (GOptionContext *context,
                                             GOptionGroup *group);
void        g_option_context_set_main_group (GOptionContext *context,
                                             GOptionGroup *group);
GOptionGroup* g_option_context_get_main_group
                                            (GOptionContext *context);
GOptionGroup* g_option_group_new            (const gchar *name,
                                             const gchar *description,
                                             const gchar *help_description,
                                             gpointer user_data,
                                             GDestroyNotify destroy);
void        g_option_group_free             (GOptionGroup *group);
void        g_option_group_add_entries      (GOptionGroup *group,
                                             const GOptionEntry *entries);
```



```
gboolean    (*GOptionParseFunc)             (GOptionContext *context,
                                             GOptionGroup *group,
                                             gpointer data,
                                             GError **error);
void        g_option_group_set_parse_hooks  (GOptionGroup *group,
                                             GOptionParseFunc pre_parse_func,
                                             GOptionParseFunc post_parse_func);
void        (*GOptionErrorFunc)             (GOptionContext *context,
                                             GOptionGroup *group,
                                             gpointer data,
                                             GError **error);
void        g_option_group_set_error_hook   (GOptionGroup *group,
                                             GOptionErrorFunc error_func);
const gchar* (*GTranslateFunc)              (const gchar *str,
                                             gpointer data);
void        g_option_group_set_translate_func
                                            (GOptionGroup *group,
                                             GTranslateFunc func,
                                             gpointer data,
                                             GDestroyNotify destroy_notify);
void        g_option_group_set_translation_domain
                                            (GOptionGroup *group,
                                             const gchar *domain);
```

Description

The GOption commandline parser is intended to be a simpler replacement for the popt library. It supports short and long commandline options, as shown in the following example:

```
testtreemodel -r 1 --max-size 20 --rand --display=:1.0 -vb -- file1 file2
```

The example demonstrates a number of features of the GOption commandline parser

- Options can be single letters, prefixed by a single dash. Multiple short options can be grouped behind a single dash.
- Long options are prefixed by two consecutive dashes.
- Options can have an extra argument, which can be a number, a string or a filename. For long options, the extra argument can be appended with an equals sign after the option name.
- Non-option arguments are returned to the application as rest arguments.
- An argument consisting solely of two dashes turns off further parsing, any remaining arguments (even those starting with a dash) are returned to the application as rest arguments.

Another important feature of GOption is that it can automatically generate nicely formatted help output. Unless it is explicitly turned off with `g_option_context_set_help_enabled()`, GOption will recognize the `--help`, `-?`, `--help-all` and `--help-`*`groupname`* options (where *`groupname`* is the name of a GOptionGroup) and write a text similar to the one shown in the following example to stdout.

```
Usage:
  testtreemodel [OPTION...] - test tree model performance

Help Options:
```

```
  --help                 Show help options
  --help-all             Show all help options
  --help-gtk             Show GTK+ Options

Application Options:
  -r, --repeats=N        Average over N repetitions
  -m, --max-size=M       Test up to 2^M items
  --display=DISPLAY      X display to use
  -v, --verbose          Be verbose
  -b, --beep             Beep when done
  --rand                 Randomize the data
```

GOption groups options in GOptionGroups, which makes it easy to incorporate options from multiple sources. The intended use for this is to let applications collect option groups from the libraries it uses, add them to their GOptionContext, and parse all options by a single call to `g_option_context_parse()`. See `gtk_get_option_group()` for an example.

If an option is declared to be of type string or filename, GOption takes care of converting it to the right encoding; strings are returned in UTF-8, filenames are returned in the GLib filename encoding.

Here is a complete example of setting up GOption to parse the example commandline above and produce the example help output.

```
static gint repeats = 2;
static gint max_size = 8;
static gboolean verbose = FALSE;
static gboolean beep = FALSE;
static gboolean rand = FALSE;

static GOptionEntry entries[] =
{
  { "repeats", 'r', 0, G_OPTION_ARG_INT, &repeats, "Average over N repetitions",
  { "max-size", 'm', 0, G_OPTION_ARG_INT, &max_size, "Test up to 2^M items", "M"
  { "verbose", 'v', 0, G_OPTION_ARG_NONE, &verbose, "Be verbose", NULL },
  { "beep", 'b', 0, G_OPTION_ARG_NONE, &beep, "Beep when done", NULL },
  { "rand", 0, 0, G_OPTION_ARG_NONE, &rand, "Randomize the data", NULL },
  { NULL }
};

int
main (int argc, char *argv[])
{
  GError *error = NULL;

  context = g_option_context_new ("- test tree model performance");
  g_option_context_add_main_entries (context, entries, GETTEXT_PACKAGE);
  g_option_context_add_group (context, gtk_get_option_group (TRUE));
  g_option_context_parse (context, &argc, &argv, &error);

  /* ... */

}
```

Details

enum GOptionError

```
typedef enum
{
  G_OPTION_ERROR_UNKNOWN_OPTION,
  G_OPTION_ERROR_BAD_VALUE,
  G_OPTION_ERROR_FAILED
} GOptionError;
```

Error codes returned by option parsing.

`G_OPTION_ERROR_UNKNOWN_OPTION`	An option was not known to the parser. This error will only be reported, if the parser hasn't been instructed to ignore unknown options, see `g_option_context_set_ignore_unknown_options()`.
`G_OPTION_ERROR_BAD_VALUE`	A value couldn't be parsed.
`G_OPTION_ERROR_FAILED`	A GOptionArgFunc callback failed.

G_OPTION_ERROR

```
#define G_OPTION_ERROR (g_option_error_quark ())
```

Error domain for option parsing. Errors in this domain will be from the GOptionError enumeration. See GError for information on error domains.

GOptionArgFunc ()

```
gboolean    (*GOptionArgFunc)               (const gchar *option_name,
                                             const gchar *value,
                                             gpointer data,
                                             GError **error);
```

The type of function to be passed as callback for `G_OPTION_ARG_CALLBACK` options.

option_name :	The name of the option being parsed. This will be either a single dash followed by a single letter (for a short name) or two dashes followed by a long option name.
value :	The value to be parsed.
data :	User data added to the GOptionGroup containing the option when it was created with `g_option_group_new()`
error :	A return location for errors. The error code `G_OPTION_ERROR_FAILED` is intended to be used for errors in GOptionArgFunc callbacks.
Returns :	`TRUE` if the option was successfully parsed, `FALSE` if an error occurred

---

GOptionContext

```
typedef struct _GOptionContext GOptionContext;
```

A GOptionContext struct defines which options are accepted by the commandline option parser. The struct has only private fields and should not be directly accessed.

---

g_option_context_new ()

```
GOptionContext* g_option_context_new        (const gchar *parameter_string);
```

Creates a new option context.

parameter_string : a string which is displayed in the first line of `--help` output, after *programname* `[OPTION...]`

Returns : a newly created GOptionContext, which must be freed with `g_option_context_free()` after use.

Since 2.6

---

g_option_context_free ()

```
void        g_option_context_free           (GOptionContext *context);
```

Frees context and all the groups which have been added to it.

context : a GOptionContext

Since 2.6

---

g_option_context_parse ()

```
gboolean    g_option_context_parse          (GOptionContext *context,
                                             gint *argc,
                                             gchar ***argv,
                                             GError **error);
```

Parses the command line arguments, recognizing options which have been added to *context*. A side-effect of calling this function is that `g_set_prgname()` will be called.

If the parsing is successful, any parsed arguments are removed from the array and *argc* and *argv* are updated accordingly. In case of an error, *argc* and *argv* are left unmodified.

If automatic `--help` support is enabled (see `g_option_context_set_help_enabled()`), and the *argv* array contains one of the recognized help options, this function will produce help output to stdout and call `exit (0)`.

context : a GOptionContext
argc : a pointer to the number of command line arguments.
argv : a pointer to the array of command line arguments.
error : a return location for errors
Returns : `TRUE` if the parsing was successful, `FALSE` if an error occurred

Since 2.6

---

g_option_context_set_help_enabled ()

```
void        g_option_context_set_help_enabled
                                            (GOptionContext *context,
                                             gboolean help_enabled);
```

Enables or disables automatic generation of `--help` output. By default, `g_option_context_parse()` recognizes `--help`, `-?`, `--help-all` and `--help-`*groupname* and creates suitable output to stdout.

context : a GOptionContext
help_enabled : `TRUE` to enable `--help`, `FALSE` to disable it

Since 2.6

---

g_option_context_get_help_enabled ()

```
gboolean    g_option_context_get_help_enabled
                                            (GOptionContext *context);
```

Returns whether automatic `--help` generation is turned on for *context*. See `g_option_context_set_help_enabled()`.

context : a GOptionContext
Returns : `TRUE` if automatic help generation is turned on.

Since 2.6

---

g_option_context_set_ignore_unknown_options ()

```
void        g_option_context_set_ignore_unknown_options
                                            (GOptionContext *context,
                                             gboolean ignore_unknown);
```

Sets whether to ignore unknown options or not. If an argument is ignored, it is left in the *argv* array after parsing. By default, g_option_context_parse() treats unknown options as error.

This setting does not affect non-option arguments (i.e. arguments which don't start with a dash). But note that GOption cannot reliably determine whether a non-option belongs to a preceding unknown option.

context : a GOptionContext
ignore_unknown : TRUE to ignore unknown options, FALSE to produce an error when unknown options are met

Since 2.6

---

g_option_context_get_ignore_unknown_options ()

```
gboolean    g_option_context_get_ignore_unknown_options
                                        (GOptionContext *context);
```

Returns whether unknown options are ignored or not. See g_option_context_set_ignore_unknown_options().

context : a GOptionContext
Returns : TRUE if unknown options are ignored.

Since 2.6

---

enum GOptionArg

```
typedef enum
{
  G_OPTION_ARG_NONE,
  G_OPTION_ARG_STRING,
  G_OPTION_ARG_INT,
  G_OPTION_ARG_CALLBACK,
  G_OPTION_ARG_FILENAME,
  G_OPTION_ARG_STRING_ARRAY,
  G_OPTION_ARG_FILENAME_ARRAY
} GOptionArg;
```

The GOptionArg enum values determine which type of extra argument the options expect to find. If an option expects an extra argument, it can be specified in several ways; with a short option: `-x arg`, with a long option: `--name arg` or combined in a single argument: `--name=arg`.

G_OPTION_ARG_NONE	No extra argument. This is useful for simple flags.
G_OPTION_ARG_STRING	The option takes a string argument.
G_OPTION_ARG_INT	The option takes an integer argument.

G_OPTION_ARG_CALLBACK	The option provides a callback to parse the extra argument.
G_OPTION_ARG_FILENAME	The option takes a filename as argument.
G_OPTION_ARG_STRING_ARRAY	The option takes a string argument, multiple uses of the option are collected into an array of strings.
G_OPTION_ARG_FILENAME_ARRAY	The option takes a filename as argument, multiple uses of the option are collected into an array of strings.

---

enum GOptionFlags

```
typedef enum
{
  G_OPTION_FLAG_HIDDEN       = 1 << 0,
  G_OPTION_FLAG_IN_MAIN      = 1 << 1,
  G_OPTION_FLAG_REVERSE      = 1 << 2
} GOptionFlags;
```

Flags which modify individual options.

G_OPTION_FLAG_HIDDEN	The option doesn't appear in `--help` output.
G_OPTION_FLAG_IN_MAIN	The option appears in the main section of the `--help` output, even if it is defined in a group.
G_OPTION_FLAG_REVERSE	For options of the G_OPTION_ARG_NONE kind, this flag indicates that the sense of the option is reversed.

G_OPTION_REMAINING

```
#define G_OPTION_REMAINING ""
```

If a long option in the main group has this name, it is not treated as a regular option. Instead it collects all non-option arguments which would otherwise be left in `argv`. The option must be of type `G_OPTION_ARG_STRING_ARRAY` or `G_OPTION_ARG_FILENAME_ARRAY`.

Using G_OPTION_REMAINING instead of simply scanning `argv` for leftover arguments has the advantage that GOption takes care of necessary encoding conversions for strings or filenames.

Since 2.6

GOptionEntry

```
typedef struct {
  const gchar *long_name;
  gchar        short_name;
  gint         flags;

  GOptionArg   arg;
  gpointer     arg_data;
```

```
  const gchar *description;
  const gchar *arg_description;
} GOptionEntry;
```

A GOptionEntry defines a single option. To have an effect, they must be added to a GOptionGroup with g_option_context_add_main_entries() or g_option_group_add_entries().

const gchar *long_name; The long name of an option can be used to specify it in a commandline as --long_name. Every option must have a long name. To resolve conflicts if multiple option groups contain the same long name, it is also possible to specify the option as --groupname-long_name.

gchar short_name; If an option has a short name, it can be specified -short_name in a commandline.

gint flags; Flags from GOptionEntryFlags.

GOptionArg arg; The type of the option, as a GOptionArg.

gpointer arg_data; If the arg type is G_OPTION_ARG_CALLBACK, then arg_data must point to a GOptionArgFunc callback function, which will be called to handle the extra argument. Otherwise, arg_data is a pointer to a location to store the value, the required type of the location depends on the arg type:

G_OPTION_ARG_NONE	gboolean
G_OPTION_ARG_STRING	gchar*
G_OPTION_ARG_INT	gint
G_OPTION_ARG_FILENAME	gchar*
G_OPTION_ARG_STRING_ARRAY	gchar**
G_OPTION_ARG_FILENAME_ARRAY	gchar**

const gchar *description; the description for the option in --help output. The description is translated using the translate_func of the group, see g_option_group_set_translation_domain().

const gchar *arg_description; The placeholder to use for the extra argument parsed by the option in --help output. The arg_description is translated using the translate_func of the group, see g_option_group_set_translation_domain().

---

g_option_context_add_main_entries ()

```
void        g_option_context_add_main_entries
                                        (GOptionContext *context,
                                         const GOptionEntry *entries,
                                         const gchar *translation_domain);
```

A convenience function which creates a main group if it doesn't exist, adds the entries to it and sets the translation domain.

context : a GOptionContext

entries : a NULL-terminated array of GOptionEntrys

translation_domain : a translation domain to use for translating the --help output for the options in entries with gettext(), or NULL

Since 2.6

GOptionGroup

```
typedef struct _GOptionGroup GOptionGroup;
```

A GOptionGroup struct defines the options in a single group. The struct has only private fields and should not be directly accessed.

All options in a group share the same translation function. Libaries which need to parse commandline options are expected to provide a function for getting a GOptionGroup holding their options, which the application can then add to its GOptionContext.

---

g_option_context_add_group ()

```
void        g_option_context_add_group      (GOptionContext *context,
                                             GOptionGroup *group);
```

Adds a GOptionGroup to the context, so that parsing with context will recognize the options in the group. Note that the group will be freed together with the context when g_option_context_free() is called, so you must not free the group yourself after adding it to a context.

context : a GOptionContext

group : the group to add

Since 2.6

---

g_option_context_set_main_group ()

```
void        g_option_context_set_main_group (GOptionContext *context,
                                             GOptionGroup *group);
```

Sets a GOptionGroup as main group of the context. This has the same effect as calling g_option_context_add_group(), the only difference is that the options in the main group are treated differently when generating --help output.

context : a GOptionContext
group : the group to set as main group

Since 2.6

---

g_option_context_get_main_group ()

```
GOptionGroup* g_option_context_get_main_group
                                        (GOptionContext *context);
```

Returns a pointer to the main group of *context*.

context : a GOptionContext
Returns : the main group of *context*, or NULL if *context* doesn't have a main group. Note that group belongs to *context* and should not be modified or freed.

Since 2.6

---

g_option_group_new ()

```
GOptionGroup* g_option_group_new            (const gchar *name,
                                             const gchar *description,
                                             const gchar *help_description,
                                             gpointer user_data,
                                             GDestroyNotify destroy);
```

Creates a new GOptionGroup.

name : the name for the option group, this is used to provide help for the options in this group with --help-*name*
description : a description for this group to be shown in --help. This string is translated using the translation domain or translation function of the group
help_description : a description for the --help-*name* option. This string is translated using the translation domain or translation function of the group
user_data : user data that will be passed to the pre- and post-parse hooks, the error hook and to callbacks of G_OPTION_ARG_CALLBACK options, or NULL
destroy : a function that will be called to free *user_data*, or NULL
Returns : a newly created option group. It should be added to a GOptionContext or freed with g_option_group_free().

Since 2.6

---

g_option_group_free ()

```
void        g_option_group_free             (GOptionGroup *group);
```

Frees a GOptionGroup. Note that you must *not* free groups which have been added to a GOptionContext.

group : a GOptionGroup

Since 2.6

---

g_option_group_add_entries ()

```
void        g_option_group_add_entries      (GOptionGroup *group,
                                             const GOptionEntry *entries);
```

Adds the options specified in *entries* to *group*.

group : a GOptionGroup
entries : a NULL-terminated array of GOptionEntrys

Since 2.6

GOptionParseFunc ()

```
gboolean    (*GOptionParseFunc)             (GOptionContext *context,
                                             GOptionGroup *group,
                                             gpointer data,
                                             GError **error);
```

The type of function that can be called before and after parsing.

context The active GOptionContext
context :
group : The group to which the function belongs
data : User data added to the GOptionGroup containing the option when it was created with g_option_group_new()
error : A return location for error details
Returns : TRUE if the function completed successfully, FALSE if an error occurred

---

g_option_group_set_parse_hooks ()

```
void        g_option_group_set_parse_hooks  (GOptionGroup *group,
                                             GOptionParseFunc pre_parse_func,
                                             GOptionParseFunc post_parse_func);
```

Associates two functions with *group* which will be called from g_option_context_parse() before the first option is parsed and after the last option has been parsed, respectively.

Note that the user data to be passed to *pre_parse_func* and *post_parse_func* can be specified when constructing the group with g_option_group_new().

group : a GOptionGroup
pre_parse_func : a function to call before parsing, or NULL
post_parse_func : a function to call after parsing, or NULL

Since 2.6

GOptionErrorFunc ()

```
void        (*GOptionErrorFunc)             (GOptionContext *context,
                                             GOptionGroup *group,
                                             gpointer data,
                                             GError **error);
```

The type of function to be used as callback when a parse error occurs.

context The active GOptionContext

context :
group : The group to which the function belongs
data : User data added to the GOptionGroup containing the option when it was created with g_option_group_new()
error : The GError containing details about the parse error

---

g_option_group_set_error_hook ()

```
void        g_option_group_set_error_hook   (GOptionGroup *group,
                                             GOptionErrorFunc error_func);
```

Associates a function with *group* which will be called from g_option_context_parse() when an error occurs.

Note that the user data to be passed to *pre_parse_func* and *post_parse_func* can be specified when constructing the group with g_option_group_new().

group : a GOptionGroup
error_func : a function to call when an error occurs

Since 2.6

GTranslateFunc ()

```
const gchar* (*GTranslateFunc)              (const gchar *str,
                                             gpointer data);
```

The type of functions which are used to translate user-visible strings, for --help output.

str : the untranslated string
data : user data specified when installing the function, e.g. in g_option_group_set_translate_func()
Returns : a translation of the string for the current locale. The returned string is owned by GLib and must not be freed.

---

g_option_group_set_translate_func ()

```
void        g_option_group_set_translate_func
                                            (GOptionGroup *group,
                                             GTranslateFunc func,
                                             gpointer data,
                                             GDestroyNotify destroy_notify);
```

Sets the function which is used to translate user-visible strings, for --help output. Different groups can use different GTranslateFuncs. If *func* is NULL, strings are not translated.

If you are using gettext(), you only need to set the translation domain, see g_option_group_set_translation_domain().

group : a GOptionGroup
func : the GTranslateFunc, or NULL
data : user data to pass to *func*, or NULL
destroy_notify : a function which gets called to free *data*, or NULL

Since 2.6

---

g_option_group_set_translation_domain ()

```
void        g_option_group_set_translation_domain
                                            (GOptionGroup *group,
                                             const gchar *domain);
```

A convenience function to use gettext() for translating user-visible strings.

group : a GOptionGroup
domain : the domain to use

Since 2.6

<< Shell-related Utilities **Glob-style pattern matching >>**

Glob-style pattern matching

Glob-style pattern matching — matches strings against patterns containing '*' (wildcard) and '?' (joker).

Synopsis

```
#include <glib.h>


            GPatternSpec;
GPatternSpec* g_pattern_spec_new            (const gchar *pattern);
void        g_pattern_spec_free             (GPatternSpec *pspec);
gboolean    g_pattern_spec_equal            (GPatternSpec *pspec1,
                                             GPatternSpec *pspec2);
gboolean    g_pattern_match                 (GPatternSpec *pspec,
                                             guint string_length,
                                             const gchar *string,
                                             const gchar *string_reversed);
gboolean    g_pattern_match_string          (GPatternSpec *pspec,
                                             const gchar *string);
gboolean    g_pattern_match_simple          (const gchar *pattern,
                                             const gchar *string);
```

Description

The g_pattern_match* functions match a string against a pattern containing '*' and '?' wildcards with similar semantics as the standard glob() function: '*' matches an arbitrary, possibly empty, string, '?' matches an arbitrary character.

Note that in contrast to glob(), the '/' character *can* be matched by the wildcards, there are no '[...]' character ranges and '*' and '?' can *not* be escaped to include them literally in a pattern.

When multiple strings must be matched against the same pattern, it is better to compile the pattern to a GPatternSpec using g_pattern_spec_new() and use g_pattern_match_string() instead of g_pattern_match_simple(). This avoids the overhead of repeated pattern compilation.

Details

GPatternSpec

```
typedef struct _GPatternSpec GPatternSpec;
```

A GPatternSpec is the 'compiled' form of a pattern. This structure is opaque and its fields cannot be accessed directly.

---

g_pattern_spec_new ()

```
GPatternSpec* g_pattern_spec_new            (const gchar *pattern);
```

Compiles a pattern to a GPatternSpec.

pattern : a zero-terminated UTF-8 encoded string.
Returns : a newly-allocated GPatternSpec.

---

g_pattern_spec_free ()

```
void        g_pattern_spec_free             (GPatternSpec *pspec);
```

Frees the memory allocated for the GPatternSpec.

pspec : a GPatternSpec.

---

g_pattern_spec_equal ()

```
gboolean    g_pattern_spec_equal            (GPatternSpec *pspec1,
                                             GPatternSpec *pspec2);
```

Compares two compiled pattern specs and returns whether they will match the same set of strings.

pspec1 : a GPatternSpec.
pspec2 : another GPatternSpec.
Returns : Whether the compiled patterns are equal.

---

g_pattern_match ()

```
gboolean    g_pattern_match                 (GPatternSpec *pspec,
                                             guint string_length,
                                             const gchar *string,
                                             const gchar *string_reversed);
```

Matches a string against a compiled pattern. Passing the correct length of the string given is mandatory. The reversed string can be omitted by passing NULL, this is more efficient if the reversed version of the string to be matched is not at hand, as g_pattern_match() will only construct it if the compiled pattern requires reverse matches.

Note that, if the user code will (possibly) match a string against a multitude of patterns containing wildcards, chances are high that some patterns will require a reversed string. In this case, it's more efficient to provide the reversed string to avoid multiple constructions thereof in the various calls to g_pattern_match().

Note also that the reverse of a UTF-8 encoded string can in general *not* be obtained by `g_strreverse()`. This works only if the string doesn't contain any multibyte characters. Glib offers the `g_utf_strreverse()` function to reverse UTF-8 encoded strings.

pspec : a GPatternSpec.
string_length : the length of *string*.
string : the UTF-8 encoded string to match.
string_reversed : the reverse of *string* or `NULL`.
Returns : `TRUE` if *string* matches *pspec*.

---

g_pattern_match_string ()

```
gboolean    g_pattern_match_string          (GPatternSpec *pspec,
                                             const gchar *string);
```

Matches a string against a compiled pattern. If the string is to be matched against more than one pattern, consider using `g_pattern_match()` instead while supplying the reversed string.

pspec : a GPatternSpec.
string : the UTF-8 encoded string to match.
Returns : `TRUE` if *string* matches *pspec*.

---

g_pattern_match_simple ()

```
gboolean    g_pattern_match_simple          (const gchar *pattern,
                                             const gchar *string);
```

Matches a string against a pattern given as a string. If this function is to be called in a loop, it's more efficient to compile the pattern once with `g_pattern_spec_new()` and call `g_pattern_match_string()` repetively.

pattern : the UTF-8 encoded pattern.
string : the UTF-8 encoded string to match.
Returns : `TRUE` if *string* matches *pspec*.

<< Commandline option parser **Simple XML Subset Parser >>**

Simple XML Subset Parser

Simple XML Subset Parser — parses a subset of XML.

Synopsis

```
#include <glib.h>


enum        GMarkupError;
#define     G_MARKUP_ERROR
enum        GMarkupParseFlags;
            GMarkupParseContext;
            GMarkupParser;
gchar*      g_markup_escape_text            (const gchar *text,
                                             gssize length);
gchar*      g_markup_printf_escaped         (const char *format,
                                             ...);
gchar*      g_markup_vprintf_escaped        (const char *format,
                                             va_list args);
gboolean    g_markup_parse_context_end_parse
                                            (GMarkupParseContext *context,
                                             GError **error);
void        g_markup_parse_context_free     (GMarkupParseContext *context);
void        g_markup_parse_context_get_position
                                            (GMarkupParseContext *context,
                                             gint *line_number,
                                             gint *char_number);
G_CONST_RETURN gchar* g_markup_parse_context_get_element
                                            (GMarkupParseContext *context);
GMarkupParseContext* g_markup_parse_context_new
                                            (const GMarkupParser *parser,
                                             GMarkupParseFlags flags,
                                             gpointer user_data,
                                             GDestroyNotify user_data_dnotify);
gboolean    g_markup_parse_context_parse    (GMarkupParseContext *context,
                                             const gchar *text,
                                             gssize text_len,
                                             GError **error);
```

Description

The "GMarkup" parser is intended to parse a simple markup format that's a subset of XML. This is a small, efficient, easy-to-use parser. It should not be used if you expect to interoperate with other applications generating full-scale XML. However, it's very useful for application data files, config files, etc. where you know your application will be the only one writing the file. Full-scale XML parsers should be able to parse the subset used by GMarkup, so you can easily migrate to full-scale XML at a later time if the need arises.

GMarkup is not guaranteed to signal an error on all invalid XML; the parser may accept documents that an XML parser would not. However, invalid XML documents are not considered valid

GMarkup documents.

Simplifications to XML include:

- Only UTF-8 encoding is allowed.
- No user-defined entities.
- Processing instructions, comments and the doctype declaration are "passed through" but are not interpreted in any way.
- No DTD or validation.

The markup format does support:

- Elements
- Attributes
- 5 standard entities: `&` `<` `>` `&quot;` `&apos;`
- Character references
- Sections marked as CDATA

Details

enum GMarkupError

```
typedef enum
{
  G_MARKUP_ERROR_BAD_UTF8,
  G_MARKUP_ERROR_EMPTY,
  G_MARKUP_ERROR_PARSE,
  /* These three are primarily intended for specific GMarkupParser
   * implementations to set.
   */
  G_MARKUP_ERROR_UNKNOWN_ELEMENT,
  G_MARKUP_ERROR_UNKNOWN_ATTRIBUTE,
  G_MARKUP_ERROR_INVALID_CONTENT
} GMarkupError;
```

Error codes returned by markup parsing.

`G_MARKUP_ERROR_BAD_UTF8`	text being parsed was not valid UTF-8
`G_MARKUP_ERROR_EMPTY`	document contained nothing, or only whitespace
`G_MARKUP_ERROR_PARSE`	document was ill-formed
`G_MARKUP_ERROR_UNKNOWN_ELEMENT`	error should be set by GMarkupParser functions; element wasn't known
`G_MARKUP_ERROR_UNKNOWN_ATTRIBUTE`	error should be set by GMarkupParser functions; attribute wasn't known
`G_MARKUP_ERROR_INVALID_CONTENT`	error should be set by GMarkupParser functions;

something was wrong with contents of the document, e.g. invalid attribute value

---

G_MARKUP_ERROR

```
#define G_MARKUP_ERROR g_markup_error_quark ()
```

Error domain for markup parsing. Errors in this domain will be from the GMarkupError enumeration. See GError for information on error domains.

---

enum GMarkupParseFlags

```
typedef enum
{
  /* Hmm, can't think of any at the moment */
  G_MARKUP_DO_NOT_USE_THIS_UNSUPPORTED_FLAG = 1 << 0

} GMarkupParseFlags;
```

There are no flags right now. Pass "0" for the flags argument to all functions.

`G_MARKUP_DO_NOT_USE_THIS_UNSUPPORTED_FLAG` flag you should not use.

GMarkupParseContext

```
typedef struct _GMarkupParseContext GMarkupParseContext;
```

A parse context is used to parse a stream of bytes that you expect to contain marked-up text. See `g_markup_parse_context_new()`, GMarkupParser, and so on for more details.

GMarkupParser

```
typedef struct {
  /* Called for open tags <foo bar="baz"> */
  void (*start_element)  (GMarkupParseContext *context,
                          const gchar         *element_name,
                          const gchar        **attribute_names,
                          const gchar        **attribute_values,
                          gpointer            user_data,
                          GError             **error);

  /* Called for close tags </foo> */
  void (*end_element)    (GMarkupParseContext *context,
                          const gchar         *element_name,
                          gpointer            user_data,
                          GError             **error);
```



```
  /* Called for character data */
  /* text is not nul-terminated */
  void (*text)           (GMarkupParseContext *context,
                          const gchar         *text,
                          gsize                text_len,
                          gpointer             user_data,
                          GError             **error);

  /* Called for strings that should be re-saved verbatim in this same
   * position, but are not otherwise interpretable.  At the moment
   * this includes comments and processing instructions.
   */
  /* text is not nul-terminated. */
  void (*passthrough)    (GMarkupParseContext *context,
                          const gchar         *passthrough_text,
                          gsize                text_len,
                          gpointer             user_data,
                          GError             **error);

  /* Called on error, including one set by other
   * methods in the vtable. The GError should not be freed.
   */
  void (*error)          (GMarkupParseContext *context,
                          GError              *error,
                          gpointer             user_data);
} GMarkupParser;
```

Any of the fields in GMarkupParser can be `NULL`, in which case they will be ignored. Except for the *`error`* function, any of these callbacks can set an error; in particular the `G_MARKUP_ERROR_UNKNOWN_ELEMENT`, `G_MARKUP_ERROR_UNKNOWN_ATTRIBUTE`, and `G_MARKUP_ERROR_INVALID_CONTENT` errors are intended to be set from these callbacks. If you set an error from a callback, `g_markup_parse_context_parse()` will report that error back to its caller.

`start_element` ()	Callback to invoke when the opening tag of an element is seen.
`end_element` ()	Callback to invoke when the closing tag of an element is seen
`text` ()	Callback to invoke when some text is seen (text is always inside an element)
`passthrough` ()	Callback to invoke for comments, processing instructions and doctype declarations; if you're re-writing the parsed document, write the passthrough text back out in the same position
`error` ()	Callback to invoke when an error occurs

---

g_markup_escape_text ()

```
gchar*      g_markup_escape_text            (const gchar *text,
                                             gssize length);
```

Escapes text so that the markup parser will parse it verbatim. Less than, greater than, ampersand, etc. are replaced with the corresponding entities. This function would typically be used when writing out a file to be parsed with the markup parser.

Note that this function doesn't protect whitespace and line endings from being processed according to the XML rules for normalization of line endings and attribute values.

text : some valid UTF-8 text
length : length of *text* in bytes
Returns : escaped text

---

g_markup_printf_escaped ()

```
gchar*      g_markup_printf_escaped         (const char *format,
                                             ...);
```

Formats arguments according to *format*, escaping all string and character arguments in the fashion of g_markup_escape_text(). This is useful when you want to insert literal strings into XML-style markup output, without having to worry that the strings might themselves contain markup.

```
const char *store = "Fortnum & Mason";
const char *item = "Tea";
char *output;

output = g_markup_printf_escaped ("<purchase>"
                                  "<store>%s</store>"
                                  "<item>%s</item>"
                                  "</purchase>",
                                  store, item);
```

format : printf() style format string
... : the arguments to insert in the format string
Returns : newly allocated result from formatting operation. Free with g_free().

Since 2.4

---

g_markup_vprintf_escaped ()

```
gchar*      g_markup_vprintf_escaped        (const char *format,
                                             va_list args);
```

Formats the data in *args* according to *format*, escaping all string and character arguments in the fashion of g_markup_escape_text(). See g_markup_printf_escaped().

format : printf() style format string
args : variable argument list, similar to vprintf()
Returns : newly allocated result from formatting operation. Free with g_free().

Since 2.4

---

g_markup_parse_context_end_parse ()


```
gboolean    g_markup_parse_context_end_parse
                                            (GMarkupParseContext *context,
                                             GError **error);
```

Signals to the GMarkupParseContext that all data has been fed into the parse context with g_markup_parse_context_parse(). This function reports an error if the document isn't complete, for example if elements are still open.

context : a GMarkupParseContext
error : return location for a GError
Returns : TRUE on success, FALSE if an error was set

---

g_markup_parse_context_free ()

```
void        g_markup_parse_context_free     (GMarkupParseContext *context);
```

Frees a GMarkupParseContext. Can't be called from inside one of the GMarkupParser functions.

context : a GMarkupParseContext

---

g_markup_parse_context_get_position ()

```
void        g_markup_parse_context_get_position
                                            (GMarkupParseContext *context,
                                             gint *line_number,
                                             gint *char_number);
```

Retrieves the current line number and the number of the character on that line. Intended for use in error messages; there are no strict semantics for what constitutes the "current" line number other than "the best number we could come up with for error messages."

context : a GMarkupParseContext
line_number : return location for a line number, or NULL
char_number : return location for a char-on-line number, or NULL

---

g_markup_parse_context_get_element ()

```
G_CONST_RETURN gchar* g_markup_parse_context_get_element
                                            (GMarkupParseContext *context);
```

Retrieves the name of the currently open element.

context : a GMarkupParseContext
Returns : the name of the currently open element, or NULL

Since 2.2

---

g_markup_parse_context_new ()

```
GMarkupParseContext* g_markup_parse_context_new
                                        (const GMarkupParser *parser,
                                         GMarkupParseFlags flags,
                                         gpointer user_data,
                                         GDestroyNotify user_data_dnotify);
```

Creates a new parse context. A parse context is used to parse marked-up documents. You can feed any number of documents into a context, as long as no errors occur; once an error occurs, the parse context can't continue to parse text (you have to free it and create a new parse context).

parser :	a GMarkupParser
flags :	one or more GMarkupParseFlags
user_data :	user data to pass to GMarkupParser functions
user_data_dnotify :	user data destroy notifier called when the parse context is freed
Returns :	a new GMarkupParseContext

---

g_markup_parse_context_parse ()

```
gboolean    g_markup_parse_context_parse    (GMarkupParseContext *context,
                                             const gchar *text,
                                             gssize text_len,
                                             GError **error);
```

Feed some data to the GMarkupParseContext. The data need not be valid UTF-8; an error will be signaled if it's invalid. The data need not be an entire document; you can feed a document into the parser incrementally, via multiple calls to this function. Typically, as you receive data from a network connection or file, you feed each received chunk of data into this function, aborting the process if an error occurs. Once an error is reported, no further data may be fed to the GMarkupParseContext; all errors are fatal.

context :	a GMarkupParseContext
text :	chunk of text to parse
text_len :	length of *text* in bytes
error :	return location for a GError
Returns :	FALSE if an error occurred, TRUE on success

<< Glob-style pattern matching **Key-value file parser >>**

Key-value file parser

Key-value file parser — parses `.ini`-like config files

Synopsis

```
#include <glib.h>


            GKeyFile;
#define     G_KEY_FILE_ERROR
enum        GKeyFileError;
enum        GKeyFileFlags;
GKeyFile*   g_key_file_new                  (void);
void        g_key_file_free                 (GKeyFile *key_file);
void        g_key_file_set_list_separator   (GKeyFile *key_file,
                                             gchar separator);
gboolean    g_key_file_load_from_file       (GKeyFile *key_file,
                                             const gchar *file,
                                             GKeyFileFlags flags,
                                             GError **error);
gboolean    g_key_file_load_from_data       (GKeyFile *key_file,
                                             const gchar *data,
                                             gsize length,
                                             GKeyFileFlags flags,
                                             GError **error);
gboolean    g_key_file_load_from_data_dirs  (GKeyFile *key_file,
                                             const gchar *file,
                                             gchar **full_path,
                                             GKeyFileFlags flags,
                                             GError **error);
gchar*      g_key_file_to_data              (GKeyFile *key_file,
                                             gsize *length,
                                             GError **error);
gchar*      g_key_file_get_start_group      (GKeyFile *key_file);
gchar**     g_key_file_get_groups           (GKeyFile *key_file,
                                             gsize *length);
gchar**     g_key_file_get_keys             (GKeyFile *key_file,
                                             const gchar *group_name,
                                             gsize *length,
                                             GError **error);
gboolean    g_key_file_has_group            (GKeyFile *key_file,
                                             const gchar *group_name);
gboolean    g_key_file_has_key              (GKeyFile *key_file,
                                             const gchar *group_name,
                                             const gchar *key,
                                             GError **error);

gchar*      g_key_file_get_value            (GKeyFile *key_file,
                                             const gchar *group_name,
                                             const gchar *key,
                                             GError **error);
gchar*      g_key_file_get_string           (GKeyFile *key_file,
                                             const gchar *group_name,
                                             const gchar *key,
```



```
                                             GError **error);
gchar*      g_key_file_get_locale_string    (GKeyFile *key_file,
                                             const gchar *group_name,
                                             const gchar *key,
                                             const gchar *locale,
                                             GError **error);
gboolean    g_key_file_get_boolean          (GKeyFile *key_file,
                                             const gchar *group_name,
                                             const gchar *key,
                                             GError **error);
gint        g_key_file_get_integer          (GKeyFile *key_file,
                                             const gchar *group_name,
                                             const gchar *key,
                                             GError **error);
gchar**     g_key_file_get_string_list      (GKeyFile *key_file,
                                             const gchar *group_name,
                                             const gchar *key,
                                             gsize *length,
                                             GError **error);
gchar**     g_key_file_get_locale_string_list
                                            (GKeyFile *key_file,
                                             const gchar *group_name,
                                             const gchar *key,
                                             const gchar *locale,
                                             gsize *length,
                                             GError **error);
gboolean*   g_key_file_get_boolean_list     (GKeyFile *key_file,
                                             const gchar *group_name,
                                             const gchar *key,
                                             gsize *length,
                                             GError **error);
gint*       g_key_file_get_integer_list     (GKeyFile *key_file,
                                             const gchar *group_name,
                                             const gchar *key,
                                             gsize *length,
                                             GError **error);
gchar*      g_key_file_get_comment          (GKeyFile *key_file,
                                             const gchar *group_name,
                                             const gchar *key,
                                             GError **error);

void        g_key_file_set_value            (GKeyFile *key_file,
                                             const gchar *group_name,
                                             const gchar *key,
                                             const gchar *value);
void        g_key_file_set_string           (GKeyFile *key_file,
                                             const gchar *group_name,
                                             const gchar *key,
                                             const gchar *string);
void        g_key_file_set_locale_string    (GKeyFile *key_file,
                                             const gchar *group_name,
                                             const gchar *key,
                                             const gchar *locale,
                                             const gchar *string);
void        g_key_file_set_boolean          (GKeyFile *key_file,
                                             const gchar *group_name,
                                             const gchar *key,
                                             gboolean value);
void        g_key_file_set_integer          (GKeyFile *key_file,
                                             const gchar *group_name,
                                             const gchar *key,
                                             gint value);
void        g_key_file_set_string_list      (GKeyFile *key_file,
                                             const gchar *group_name,
```

```
                                        const gchar *key,
                                        const gchar *const   list[],
                                        gsize length);
void        g_key_file_set_locale_string_list
                                       (GKeyFile *key_file,
                                        const gchar *group_name,
                                        const gchar *key,
                                        const gchar *locale,
                                        const gchar *const   list[],
                                        gsize length);
void        g_key_file_set_boolean_list     (GKeyFile *key_file,
                                        const gchar *group_name,
                                        const gchar *key,
                                        gboolean list[],
                                        gsize length);
void        g_key_file_set_integer_list     (GKeyFile *key_file,
                                        const gchar *group_name,
                                        const gchar *key,
                                        gint list[],
                                        gsize length);
void        g_key_file_set_comment          (GKeyFile *key_file,
                                        const gchar *group_name,
                                        const gchar *key,
                                        const gchar *comment,
                                        GError **error);
void        g_key_file_remove_group         (GKeyFile *key_file,
                                        const gchar *group_name,
                                        GError **error);
void        g_key_file_remove_key           (GKeyFile *key_file,
                                        const gchar *group_name,
                                        const gchar *key,
                                        GError **error);
void        g_key_file_remove_comment       (GKeyFile *key_file,
                                        const gchar *group_name,
                                        const gchar *key,
                                        GError **error);
```

Description

GKeyFile lets you parse, edit or create files containing groups of key-value pairs, which we call *key files* for lack of a better name. Several freedesktop.org specifications use key files now, e.g the Desktop Entry Specification and the Icon Theme Specification.

The syntax of key files is described in detail in the Desktop Entry Specification, here is a quick summary: Key files consists of groups of key-value pairs, interspersed with comments.

```
# this is just an example
# there can be comments before the first group

[First Group]

Name=Key File Example\tthis value shows\nescaping

# localized strings are stored in multiple key-value pairs
Welcome=Hello
Welcome[de]=Hallo
Welcome[fr]=Bonjour
Welcome[it]=Ciao

[Another Group]
```



```
Numbers=2;20;-200;0

Booleans=true;false;true;true
```

Lines beginning with a '#' and blank lines are considered comments.

Groups are started by a header line containing the group name enclosed in '[' and ']', and ended implicitly by the start of the next group or the end of the file. Each key-value pair must be contained in a group.

Key-value pairs generally have the form `key=value`, with the exception of localized strings, which have the form `key[locale]=value`. Space before and after the '=' character are ignored. Newline, tab, carriage return and backslash characters are escaped as \n, \t, \r, and \\, respectively. To preserve initial and final spaces, these can also be escaped as \s.

Key files can store strings (possibly with localized variants), integers, booleans and lists of these. Lists are separated by a separator character, typically ';' or ','. To use the list separator character in a value in a list, it has to be escaped by prefixing it with a backslash.

This syntax is obviously inspired by the `.ini` files commonly met on Windows, but there are some important differences:

- `.ini` files use the ';' character to begin comments, key files use the '#' character.
- Key files allow only comments before the first group.
- Key files are always encoded in UTF-8.

Details

GKeyFile

```
typedef struct _GKeyFile GKeyFile;
```

The GKeyFile struct contains only private fields and should not be used directly.

G_KEY_FILE_ERROR

```
#define G_KEY_FILE_ERROR g_key_file_error_quark()
```

Error domain for key file parsing. Errors in this domain will be from the GKeyFileError enumeration. See GError for information on error domains.

---

enum GKeyFileError

```
typedef enum
{
  G_KEY_FILE_ERROR_UNKNOWN_ENCODING,
  G_KEY_FILE_ERROR_PARSE,
  G_KEY_FILE_ERROR_NOT_FOUND,
  G_KEY_FILE_ERROR_KEY_NOT_FOUND,
  G_KEY_FILE_ERROR_GROUP_NOT_FOUND,
  G_KEY_FILE_ERROR_INVALID_VALUE
} GKeyFileError;
```

Error codes returned by key file parsing.

`G_KEY_FILE_ERROR_UNKNOWN_ENCODING`	the text being parsed was in an unknown encoding
`G_KEY_FILE_ERROR_PARSE`	document was ill-formed
`G_KEY_FILE_ERROR_NOT_FOUND`	the file was not found
`G_KEY_FILE_ERROR_KEY_NOT_FOUND`	a requested key was not found
`G_KEY_FILE_ERROR_GROUP_NOT_FOUND`	a requested group was not found
`G_KEY_FILE_ERROR_INVALID_VALUE`	a value could not be parsed

---

enum GKeyFileFlags

```
typedef enum
{
  G_KEY_FILE_NONE              = 0,
  G_KEY_FILE_KEEP_COMMENTS     = 1 << 0,
  G_KEY_FILE_KEEP_TRANSLATIONS = 1 << 1
} GKeyFileFlags;
```

Flags which influence the parsing.

`G_KEY_FILE_NONE`	No flags, default behaviour
`G_KEY_FILE_KEEP_COMMENTS`	Use this flag if you plan to write the (possibly modified) contents of the key file back to a file; otherwise all comments will be lost when the key file is written back.
`G_KEY_FILE_KEEP_TRANSLATIONS`	Use this flag if you plan to write the (possibly modified) contents of the key file back to a file; otherwise only the translations for the current language will be written back.

---

g_key_file_new ()

```
GKeyFile*   g_key_file_new                  (void);
```

Creates a new empty GKeyFile object. Use `g_key_file_load_from_file()`, `g_key_file_load_from_data()` or `g_key_file_load_from_data_dirs()` to read an existing key file.

Returns : an empty GKeyFile.

Since 2.6

---

g_key_file_free ()

```
void        g_key_file_free                 (GKeyFile *key_file);
```

Frees a GKeyFile.

`key_file` : a GKeyFile

Since 2.6

---

g_key_file_set_list_separator ()

```
void        g_key_file_set_list_separator   (GKeyFile *key_file,
                                             gchar separator);
```

Sets the character which is used to separate values in lists. Typically ';' or ',' are used as separators. The default list separator is ';'.

`key_file` : a GKeyFile
`separator` : the separator

Since 2.6

---

g_key_file_load_from_file ()

```
gboolean    g_key_file_load_from_file       (GKeyFile *key_file,
                                             const gchar *file,
                                             GKeyFileFlags flags,
                                             GError **error);
```

Loads a key file into an empty GKeyFile structure. If the file could not be loaded then `error` is set to either a GFileError or GKeyFileError.

`key_file` : an empty GKeyFile struct
`file` : the path of a filename to load, in the GLib file name encoding
`flags` : flags from GKeyFileFlags
`error` : return location for a GError, or `NULL`
Returns :

TRUE if a key file could be loaded, FALSE othewise

Since 2.6

---

g_key_file_load_from_data ()

```
gboolean    g_key_file_load_from_data       (GKeyFile *key_file,
                                             const gchar *data,
                                             gsize length,
                                             GKeyFileFlags flags,
                                             GError **error);
```

Loads a key file from memory into an empty GKeyFile structure. If the object cannot be created then error is set to a GKeyFileError.

key_file : an empty GKeyFile struct
data : key file loaded in memory.
length : the length of *data* in bytes
flags : flags from GKeyFileFlags
error : return location for a GError, or NULL
Returns : TRUE if a key file could be loaded, FALSE othewise

Since 2.6

---

g_key_file_load_from_data_dirs ()

```
gboolean    g_key_file_load_from_data_dirs  (GKeyFile *key_file,
                                             const gchar *file,
                                             gchar **full_path,
                                             GKeyFileFlags flags,
                                             GError **error);
```

This function looks for a key file named *file* in the paths returned from g_get_user_data_dir() and g_get_system_data_dirs(), loads the file into *key_file* and returns the file's full path in *full_path*. If the file could not be loaded then an error is set to either a GFileError or GKeyFileError.

key_file : an empty GKeyFile struct
file : a relative path to a filename to open and parse
full_path : return location for a string containing the full path of the file, or NULL
flags : flags from GKeyFileFlags
error : return location for a GError, or NULL
Returns : TRUE if a key file could be loaded, FALSE othewise

Since 2.6

---

g_key_file_to_data ()

```
gchar*      g_key_file_to_data              (GKeyFile *key_file,
                                             gsize *length,
                                             GError **error);
```

This function outputs *key_file* as a string.

key_file : a GKeyFile
length : return location for the length of the returned string, or NULL
error : return location for a GError, or NULL
Returns : a newly allocated string holding the contents of the GKeyFile

Since 2.6

---

g_key_file_get_start_group ()

```
gchar*      g_key_file_get_start_group      (GKeyFile *key_file);
```

Returns the name of the start group of the file.

key_file : a GKeyFile
Returns : The start group of the key file.

Since 2.6

---

g_key_file_get_groups ()

```
gchar**     g_key_file_get_groups           (GKeyFile *key_file,
                                             gsize *length);
```

Returns all groups in the key file loaded with *key_file*. The array of returned groups will be NULL-terminated, so *length* may optionally be NULL.

key_file : a GKeyFile
length : return location for the number of returned groups, or NULL
Returns : a newly-allocated NULL-terminated array of strings. Use g_strfreev() to free it.

Since 2.6

---

g_key_file_get_keys ()

```
gchar**     g_key_file_get_keys             (GKeyFile *key_file,
                                             const gchar *group_name,
                                             gsize *length,
                                             GError **error);
```

Returns all keys for the group name *group_name*. The array of returned keys will be NULL-terminated, so *length* may optionally be NULL. In the event that the *group_name* cannot be found, NULL is returned and *error* is set to G_KEY_FILE_ERROR_GROUP_NOT_FOUND.

key_file : a GKeyFile
group_name : a group name
length : return location for the number of keys returned, or NULL
error : return location for a GError, or NULL
Returns : a newly-allocated NULL-terminated array of strings. Use g_strfreev() to free it.

Since 2.6

---

g_key_file_has_group ()

```
gboolean    g_key_file_has_group            (GKeyFile *key_file,
                                             const gchar *group_name);
```

Looks whether the key file has the group *group_name*.

key_file : a GKeyFile
group_name : a group name
Returns : TRUE if *group_name* is a part of *key_file*, FALSE otherwise.

Since 2.6

---

g_key_file_has_key ()

```
gboolean    g_key_file_has_key              (GKeyFile *key_file,
                                             const gchar *group_name,
                                             const gchar *key,
                                             GError **error);
```

Looks whether the key file has the key *key* in the group *group_name*. If *group_name* is NULL, the

start group is used.

key_file : a GKeyFile
group_name : a group name
key : a key name
error : return location for a GError
Returns : TRUE if *key* is a part of *group_name*, FALSE otherwise.

Since 2.6

---

g_key_file_get_value ()

```
gchar*      g_key_file_get_value            (GKeyFile *key_file,
                                             const gchar *group_name,
                                             const gchar *key,
                                             GError **error);
```

Returns the value associated with *key* under *group_name*.

In the event the key cannot be found, NULL is returned and *error* is set to G_KEY_FILE_ERROR_KEY_NOT_FOUND. In the event that the *group_name* cannot be found, NULL is returned and *error* is set to G_KEY_FILE_ERROR_GROUP_NOT_FOUND.

key_file : a GKeyFile
group_name : a group name
key : a key
error : return location for a GError, or NULL
Returns : a string or NULL if the specified key cannot be found.

Since 2.6

---

g_key_file_get_string ()

```
gchar*      g_key_file_get_string           (GKeyFile *key_file,
                                             const gchar *group_name,
                                             const gchar *key,
                                             GError **error);
```

Returns the value associated with *key* under *group_name*.

In the event the key cannot be found, NULL is returned and *error* is set to G_KEY_FILE_ERROR_KEY_NOT_FOUND. In the event that the *group_name* cannot be found, NULL is returned and *error* is set to G_KEY_FILE_ERROR_GROUP_NOT_FOUND.

key_file : a GKeyFile

group_name : a group name
key : a key
error : return location for a GError, or NULL
Returns : a string or NULL if the specified key cannot be found.

Since 2.6

---

g_key_file_get_locale_string ()

```
gchar*      g_key_file_get_locale_string    (GKeyFile *key_file,
                                             const gchar *group_name,
                                             const gchar *key,
                                             const gchar *locale,
                                             GError **error);
```

Returns the value associated with *key* under *group_name* translated in the given *locale* if available. If *locale* is NULL then the current locale is assumed.

If *key* cannot be found then NULL is returned and *error* is set to G_KEY_FILE_ERROR_KEY_NOT_FOUND. If the value associated with *key* cannot be interpreted or no suitable translation can be found then the untranslated value is returned.

key_file : a GKeyFile
group_name : a group name
key : a key
locale : a locale or NULL
error : return location for a GError, or NULL
Returns : a string or NULL if the specified key cannot be found.

Since 2.6

---

g_key_file_get_boolean ()

```
gboolean    g_key_file_get_boolean          (GKeyFile *key_file,
                                             const gchar *group_name,
                                             const gchar *key,
                                             GError **error);
```

Returns the value associated with *key* under *group_name* as a boolean. If *group_name* is NULL, the start group is used.

If *key* cannot be found then the return value is undefined and *error* is set to G_KEY_FILE_ERROR_KEY_NOT_FOUND. Likewise, if the value associated with *key* cannot be interpreted as a boolean then the return value is also undefined and *error* is set to G_KEY_FILE_ERROR_INVALID_VALUE.

key_file : a GKeyFile
group_name : a group name
key : a key
error : return location for a GError
Returns : the value associated with the key as a boolean

Since 2.6

---

g_key_file_get_integer ()

```
gint        g_key_file_get_integer          (GKeyFile *key_file,
                                             const gchar *group_name,
                                             const gchar *key,
                                             GError **error);
```

Returns the value associated with *key* under *group_name* as an integer. If *group_name* is NULL, the start_group is used.

If *key* cannot be found then the return value is undefined and *error* is set to G_KEY_FILE_ERROR_KEY_NOT_FOUND. Likewise, if the value associated with *key* cannot be interpreted as an integer then the return value is also undefined and *error* is set to G_KEY_FILE_ERROR_INVALID_VALUE.

key_file : a GKeyFile
group_name : a group name
key : a key
error : return location for a GError
Returns : the value associated with the key as an integer.

Since 2.6

---

g_key_file_get_string_list ()

```
gchar**     g_key_file_get_string_list      (GKeyFile *key_file,
                                             const gchar *group_name,
                                             const gchar *key,
                                             gsize *length,
                                             GError **error);
```

Returns the values associated with *key* under *group_name*.

In the event the key cannot be found, NULL is returned and *error* is set to G_KEY_FILE_ERROR_KEY_NOT_FOUND. In the event that the *group_name* cannot be found, NULL is returned and *error* is set to G_KEY_FILE_ERROR_GROUP_NOT_FOUND.

key_file : a GKeyFile
group_name : a group name
key : a key
length : return location for the number of returned strings, or NULL
error : return location for a GError, or NULL
Returns : a NULL-terminated string array or NULL if the specified key cannot be found. The array should be freed with g_strfreev().

Since 2.6

---

g_key_file_get_locale_string_list ()

```
gchar**     g_key_file_get_locale_string_list
                                        (GKeyFile *key_file,
                                         const gchar *group_name,
                                         const gchar *key,
                                         const gchar *locale,
                                         gsize *length,
                                         GError **error);
```

Returns the values associated with *key* under *group_name* translated in the given *locale* if available. If *locale* is NULL then the current locale is assumed. If *group_name* is NULL, then the start group is used.

If *key* cannot be found then NULL is returned and *error* is set to G_KEY_FILE_ERROR_KEY_NOT_FOUND. If the values associated with *key* cannot be interpreted or no suitable translations can be found then the untranslated values are returned. The returned array is NULL-terminated, so *length* may optionally be NULL.

key_file : a GKeyFile
group_name : a group name
key : a key
locale : a locale
length : return location for the number of returned strings or NULL
error : return location for a GError or NULL
Returns : a newly allocated NULL-terminated string array or NULL if the key isn't found. The string array should be freed with g_strfreev().

Since 2.6

---

g_key_file_get_boolean_list ()

```
gboolean*   g_key_file_get_boolean_list     (GKeyFile *key_file,
                                         const gchar *group_name,
                                         const gchar *key,
```



```
                                         gsize *length,
                                         GError **error);
```

Returns the values associated with *key* under *group_name* as booleans. If *group_name* is NULL, the start_group is used.

If *key* cannot be found then the return value is undefined and *error* is set to G_KEY_FILE_ERROR_KEY_NOT_FOUND. Likewise, if the values associated with *key* cannot be interpreted as booleans then the return value is also undefined and *error* is set to G_KEY_FILE_ERROR_INVALID_VALUE.

key_file : a GKeyFile
group_name : a group name
key : a key
length : the number of booleans returned
error : return location for a GError
Returns : the values associated with the key as a boolean

Since 2.6

---

g_key_file_get_integer_list ()

```
gint*       g_key_file_get_integer_list     (GKeyFile *key_file,
                                         const gchar *group_name,
                                         const gchar *key,
                                         gsize *length,
                                         GError **error);
```

Returns the values associated with *key* under *group_name* as integers. If *group_name* is NULL, the start group is used.

If *key* cannot be found then the return value is undefined and *error* is set to G_KEY_FILE_ERROR_KEY_NOT_FOUND. Likewise, if the values associated with *key* cannot be interpreted as integers then the return value is also undefined and *error* is set to G_KEY_FILE_ERROR_INVALID_VALUE.

key_file : a GKeyFile
group_name : a group name
key : a key
length : the number of integers returned
error : return location for a GError
Returns : the values associated with the key as a integer

Since 2.6

---

g_key_file_get_comment ()

```
gchar*      g_key_file_get_comment          (GKeyFile *key_file,
                                             const gchar *group_name,
                                             const gchar *key,
                                             GError **error);
```

Retreives a comment above *key* from *group_name*. *group_name*. If *key* is NULL then *comment* will be read from above *group_name*. If both *key* and *group_name* are NULL, then *comment* will be read from above the first group in the file.

Return value:

key_file : a GKeyFile
group_name : a group name, or NULL
key : a key
error : return location for a GError
Returns : a comment that should be freed with g_free()

Since 2.6

---

g_key_file_set_value ()

```
void        g_key_file_set_value            (GKeyFile *key_file,
                                             const gchar *group_name,
                                             const gchar *key,
                                             const gchar *value);
```

Associates a new value with *key* under *group_name*. If *key* cannot be found then it is created. If *group_name* cannot be found then it is created.

key_file : a GKeyFile
group_name : a group name
key : a key
value : a string

Since 2.6

---

g_key_file_set_string ()

```
void        g_key_file_set_string           (GKeyFile *key_file,
                                             const gchar *group_name,
                                             const gchar *key,
                                             const gchar *string);
```


Associates a new string value with *key* under *group_name*. If *key* cannot be found then it is created. If *group_name* cannot be found then it is created.

key_file : a GKeyFile
group_name : a group name
key : a key
string : a string

Since 2.6

---

g_key_file_set_locale_string ()

```
void        g_key_file_set_locale_string    (GKeyFile *key_file,
                                             const gchar *group_name,
                                             const gchar *key,
                                             const gchar *locale,
                                             const gchar *string);
```

Associates a string value for *key* and *locale* under *group_name*. If the translation for *key* cannot be found then it is created.

key_file : a GKeyFile
group_name : a group name
key : a key
locale : a locale
string : a string

Since 2.6

---

g_key_file_set_boolean ()

```
void        g_key_file_set_boolean          (GKeyFile *key_file,
                                             const gchar *group_name,
                                             const gchar *key,
                                             gboolean value);
```

Associates a new boolean value with *key* under *group_name*. If *key* cannot be found then it is created. If *group_name* is NULL, the start group is used.

key_file : a GKeyFile
group_name : a group name
key : a key
value : TRUE or FALSE

Since 2.6

---

g_key_file_set_integer ()

```
void        g_key_file_set_integer          (GKeyFile *key_file,
                                             const gchar *group_name,
                                             const gchar *key,
                                             gint value);
```

Associates a new integer value with *key* under *group_name*. If *key* cannot be found then it is created. If *group_name* is NULL, the start group is used.

key_file : a GKeyFile
group_name : a group name
key : a key
value : an integer value

Since 2.6

---

g_key_file_set_string_list ()

```
void        g_key_file_set_string_list      (GKeyFile *key_file,
                                             const gchar *group_name,
                                             const gchar *key,
                                             const gchar *const   list[],
                                             gsize length);
```

Associates a list of string values for *key* under *group_name*. If *key* cannot be found then it is created. If *group_name* cannot be found then it is created.

key_file : a GKeyFile
group_name : a group name
key : a key
list : an array of locale string values
length : number of locale string values in *list*

Since 2.6

---

g_key_file_set_locale_string_list ()

```
void        g_key_file_set_locale_string_list
                                            (GKeyFile *key_file,
                                             const gchar *group_name,
                                             const gchar *key,
```



```
                                             const gchar *locale,
                                             const gchar *const   list[],
                                             gsize length);
```

Associates a list of string values for *key* and *locale* under *group_name*. If the translation for *key* cannot be found then it is created. If *group_name* is NULL, the start group is used.

key_file : a GKeyFile
group_name : a group name
key : a key
locale : a locale
list : a NULL-terminated array of locale string values
length : the length of *list*

Since 2.6

---

g_key_file_set_boolean_list ()

```
void        g_key_file_set_boolean_list     (GKeyFile *key_file,
                                             const gchar *group_name,
                                             const gchar *key,
                                             gboolean list[],
                                             gsize length);
```

Associates a list of boolean values with *key* under *group_name*. If *key* cannot be found then it is created. If *group_name* is NULL, the start_group is used.

key_file : a GKeyFile
group_name : a group name
key : a key
list : an array of boolean values
length : length of *list*

Since 2.6

---

g_key_file_set_integer_list ()

```
void        g_key_file_set_integer_list     (GKeyFile *key_file,
                                             const gchar *group_name,
                                             const gchar *key,
                                             gint list[],
                                             gsize length);
```

Associates a list of integer values with *key* under *group_name*. If *key* cannot be found then it is created. If *group_name* is NULL the start group is used.

key_file : a GKeyFile
group_name : a group name
key : a key
list : an array of integer values
length : number of integer values in *list*

Since 2.6

---

g_key_file_set_comment ()

```
void        g_key_file_set_comment          (GKeyFile *key_file,
                                             const gchar *group_name,
                                             const gchar *key,
                                             const gchar *comment,
                                             GError **error);
```

Places a comment above *key* from *group_name*. *group_name*. If *key* is NULL then *comment* will be written above *group_name*. If both *key* and *group_name* are NULL, then *comment* will be written above the first group in the file.

key_file : a GKeyFile
group_name : a group name, or NULL
key : a key
comment : a comment
error : return location for a GError

Since 2.6

---

g_key_file_remove_group ()

```
void        g_key_file_remove_group         (GKeyFile *key_file,
                                             const gchar *group_name,
                                             GError **error);
```

Removes the specified group, *group_name*, from the key file.

key_file : a GKeyFile
group_name : a group name
error : return location for a GError or NULL

Since 2.6

---

g_key_file_remove_key ()

```
void        g_key_file_remove_key           (GKeyFile *key_file,
                                             const gchar *group_name,
                                             const gchar *key,
                                             GError **error);
```

Removes *key* in *group_name* from the key file.

key_file : a GKeyFile
group_name : a group name
key : a key name to remove
error : return location for a GError or NULL

Since 2.6

---

g_key_file_remove_comment ()

```
void        g_key_file_remove_comment       (GKeyFile *key_file,
                                             const gchar *group_name,
                                             const gchar *key,
                                             GError **error);
```

Removes a comment above *key* from *group_name*. *group_name*. If *key* is NULL then *comment* will be written above *group_name*. If both *key* and *group_name* are NULL, then *comment* will be written above the first group in the file.

key_file : a GKeyFile
group_name : a group name, or NULL
key : a key
error : return location for a GError

Since 2.6

<< Simple XML Subset Parser **Windows Compatibility Functions >>**

Windows Compatibility Functions

Windows Compatibility Functions — UNIX emulation on Windows.

Synopsis

```
#include <glib.h>


#define     MAXPATHLEN
typedef     pid_t;
#define     pipe                            (phandles)
gchar*      g_win32_error_message           (gint error);
gchar*      g_win32_getlocale               (void);
gchar*      g_win32_get_package_installation_directory
                                            (gchar *package,
                                             gchar *dll_name);
gchar*      g_win32_get_package_installation_subdirectory
                                            (gchar *package,
                                             gchar *dll_name,
                                             gchar *subdir);
guint       g_win32_get_windows_version     (void);
#define     G_WIN32_DLLMAIN_FOR_DLL_NAME    (static, dll_name)
#define     G_WIN32_HAVE_WIDECHAR_API       ()
#define     G_WIN32_IS_NT_BASED             ()
```

Description

Details

MAXPATHLEN

```
#define MAXPATHLEN 1024
```

Provided for UNIX emulation on Windows; equivalent to UNIX macro `MAXPATHLEN`, which is the maximum length of a filename (including full path).

---

pid_t

```
typedef int pid_t;
```

Provided for UNIX emulation on Windows; process ID type.

---

pipe()

```
#define pipe(phandles)  _pipe (phandles, 4096, _O_BINARY)
```

Provided for UNIX emulation on Windows; see documentation for `pipe()` in any UNIX manual.

phandles :

---

g_win32_error_message ()

```
gchar*      g_win32_error_message           (gint error);
```

Translate a Win32 error code (as returned by `GetLastError()`) into the corresponding message. The message is either language neutral, or in the thread's language, or the user's language, the system's language, or US English (see docs for `FormatMessage()`). The returned string is in UTF-8. It should be deallocated with `g_free()`.

error : error code.

Returns : newly-allocated error message

---

g_win32_getlocale ()

```
gchar*      g_win32_getlocale               (void);
```

The `setlocale()` function in the Microsoft C library uses locale names of the form "English_United States.1252" etc. We want the UNIXish standard form "en_US", "zh_TW" etc. This function gets the current thread locale from Windows - without any encoding info - and returns it as a string of the above form for use in forming file names etc. The returned string should be deallocated with `g_free()`.

Returns : newly-allocated locale name.

---

g_win32_get_package_installation_directory ()

```
gchar*      g_win32_get_package_installation_directory
                                            (gchar *package,
                                             gchar *dll_name);
```

Try to determine the installation directory for a software package. Typically used by GNU software packages.

package should be a short identifier for the package. Typically it is the same identifier as used for `GETTEXT_PACKAGE` in software configured according to GNU standards. The function first looks in the Windows Registry for the value `#InstallationDirectory` in the key `#HKLM\Software\`*package*, and if that value exists and is a string, returns that.

If *package* is NULL, or the above value isn't found in the Registry, but *dll_name* is non-NULL, it should name a DLL loaded into the current process. Typically that would be the name of the DLL calling this function, looking for its installation directory. The function then asks Windows what directory that DLL was loaded from. If that directory's last component is "bin" or "lib", the parent directory is returned, otherwise the directory itself. If that DLL isn't loaded, the function proceeds as if *dll_name* was NULL.

If both *package* and *dll_name* are NULL, the directory from where the main executable of the process was loaded is uses instead in the same way as above.

package : An identifier for a software package, or NULL

dll_name : The name of a DLL that a package provides, or NULL.

Returns : a string containing the installation directory for *package*. The return value should be freed with g_free() when not needed any longer.

---

g_win32_get_package_installation_subdirectory ()

```
gchar*      g_win32_get_package_installation_subdirectory
                                        (gchar *package,
                                         gchar *dll_name,
                                         gchar *subdir);
```

Returns a newly-allocated string containing the path of the subdirectory *subdir* in the return value from calling g_win32_get_package_installation_directory() with the *package* and *dll_name* parameters.

package : An identifier for a software package, or NULL.

dll_name : The name of a DLL that a package provides, or NULL.

subdir : A subdirectory of the package installation directory.

Returns : a string containing the complete path to *subdir* inside the installation directory of *package*. The return value should be freed with g_free() when no longer needed.

---

g_win32_get_windows_version ()

```
guint       g_win32_get_windows_version     (void);
```

Returns version information for the Windows operating system the code is running on. See MSDN documentation for the GetVersion() function. To summarize, the most significant bit is one on Win9x, and zero on NT-based systems. The least significant byte is 4 on Windows NT 4, 5 on Windows XP. Software that needs really detailled version and feature information should use Win32 API like GetVersionEx() and VerifyVersionInfo().

If there is an environment variable G_WIN32_PRETEND_WIN9X defined (with any value), this function always returns a version code for Windows 9x. This is mainly an internal debugging aid for GTK+ and GLib developers, to be able to check the code paths for Windows 9x.

Returns : The version information.

Since 2.6

G_WIN32_DLLMAIN_FOR_DLL_NAME()

```
#define     G_WIN32_DLLMAIN_FOR_DLL_NAME(static, dll_name)
```

On Windows, this macro defines a DllMain() function that stores the actual DLL name that the code being compiled will be included in.

On non-Windows platforms, expands to nothing.

static : empty or "static".

dll_name : the name of the (pointer to the) char array where the DLL name will be stored. If this is used, you must also include windows.h. If you need a more complex DLL entry point function, you cannot use this.

G_WIN32_HAVE_WIDECHAR_API()

```
#define G_WIN32_HAVE_WIDECHAR_API() (G_WIN32_IS_NT_BASED ())
```

On Windows, this macro defines an expression which evaluates to TRUE if the code is running on a version of Windows where the wide character versions of the Win32 API functions, and the wide chaacter versions of the C library functions work. (They are always present in the DLLs, but don't work on Windows 9x and Me.)

On non-Windows platforms, it is not defined.

Since 2.6

G_WIN32_IS_NT_BASED()

```
#define G_WIN32_IS_NT_BASED() (g_win32_get_windows_version () < 0x80000000)
```

On Windows, this macro defines an expression which evaluates to TRUE if the code is running on an NT-based Windows operating system.

On non-Windows platforms, it is not defined.

Since 2.6

<< Key-value file parser **GLib Data Types >>**

GLib Data Types

Memory Chunks - efficient way to allocate groups of equal-sized chunks of memory.
Doubly-Linked Lists - linked lists containing integer values or pointers to data, with the ability to iterate over the list in both directions.
Singly-Linked Lists - linked lists containing integer values or pointers to data, limited to iterating over the list in one direction.
Double-ended Queues - double-ended queue data structure.
Trash Stacks - maintain a stack of unused allocated memory chunks.
Hash Tables - associations between keys and values so that given a key the value can be found quickly.
Strings - text buffers which grow automatically as text is added.
String Chunks - efficient storage of groups of strings.
Arrays - arrays of arbitrary elements which grow automatically as elements are added.
Pointer Arrays - arrays of pointers to any type of data, which grow automatically as new elements are added.
Byte Arrays - arrays of bytes, which grow automatically as elements are added.
Balanced Binary Trees - a sorted collection of key/value pairs optimized for searching and traversing in order.
N-ary Trees - trees of data with any number of branches.
Quarks - a 2-way association between a string and a unique integer identifier.
Keyed Data Lists - lists of data elements which are accessible by a string or GQuark identifier.
Datasets - associate groups of data elements with particular memory locations.
Relations and Tuples - tables of data which can be indexed on any number of fields.
Caches - caches allow sharing of complex data structures to save resources.
Memory Allocators - allocates chunks of memory for GList, GSList and GNode.

<< Windows Compatibility Functions **Memory Chunks >>**

Memory Chunks

Memory Chunks — efficient way to allocate groups of equal-sized chunks of memory.

Synopsis

```
#include <glib.h>


            GMemChunk;
#define     G_ALLOC_AND_FREE
#define     G_ALLOC_ONLY

GMemChunk*  g_mem_chunk_new                 (const gchar *name,
                                             gint atom_size,
                                             gulong area_size,
                                             gint type);
gpointer    g_mem_chunk_alloc               (GMemChunk *mem_chunk);
gpointer    g_mem_chunk_alloc0              (GMemChunk *mem_chunk);
void        g_mem_chunk_free                (GMemChunk *mem_chunk,
                                             gpointer mem);
void        g_mem_chunk_destroy             (GMemChunk *mem_chunk);

#define     g_mem_chunk_create              (type, pre_alloc, alloc_type)
#define     g_chunk_new                     (type, chunk)
#define     g_chunk_new0                    (type, chunk)
#define     g_chunk_free                    (mem, mem_chunk)

void        g_mem_chunk_reset               (GMemChunk *mem_chunk);
void        g_mem_chunk_clean               (GMemChunk *mem_chunk);
void        g_blow_chunks                   (void);

void        g_mem_chunk_info                (void);
void        g_mem_chunk_print               (GMemChunk *mem_chunk);
```

Description

Memory chunks provide an efficient way to allocate equal-sized pieces of memory, called atoms. They are used extensively within GLib itself. For example, the Doubly Linked Lists use memory chunks to allocate space for elements of the lists.

There are two types of memory chunks, G_ALLOC_ONLY, and G_ALLOC_AND_FREE.

- G_ALLOC_ONLY chunks only allow allocation of atoms. The atoms can never be freed individually. The memory chunk can only be free in its entirety.
- G_ALLOC_AND_FREE chunks do allow atoms to be freed individually. The disadvantage of this is that the memory chunk has to keep track of which atoms have been freed. This results in more memory being used and a slight degradation in performance.

To create a memory chunk use g_mem_chunk_new() or the convenience macro g_mem_chunk_create().

To allocate a new atom use g_mem_chunk_alloc(), g_mem_chunk_alloc0(), or the convenience macros g_chunk_new() or g_chunk_new0().

To free an atom use g_mem_chunk_free(), or the convenience macro g_chunk_free(). (Atoms can only be freed if the memory chunk is created with the type set to G_ALLOC_AND_FREE.)

To free any blocks of memory which are no longer being used, use g_mem_chunk_clean(). To clean all memory chunks, use g_blow_chunks().

To reset the memory chunk, freeing all of the atoms, use g_mem_chunk_reset().

To destroy a memory chunk, use g_mem_chunk_destroy().

To help debug memory chunks, use g_mem_chunk_info() and g_mem_chunk_print().

Example 1. Using a GMemChunk

```
GMemChunk *mem_chunk;
gchar *mem[10000];
gint i;

/* Create a GMemChunk with atoms 50 bytes long, and memory blocks holding
   100 bytes. Note that this means that only 2 atoms fit into each memory
   block and so isn't very efficient. */
mem_chunk = g_mem_chunk_new ("test mem chunk", 50, 100, G_ALLOC_AND_FREE);

/* Now allocate 10000 atoms. */
for (i = 0; i < 10000; i++)
  {
    mem[i] = g_chunk_new (gchar, mem_chunk);

    /* Fill in the atom memory with some junk. */
    for (j = 0; j < 50; j++)
      mem[i][j] = i * j;
  }

/* Now free all of the atoms. Note that since we are going to destroy the
   GMemChunk, this wouldn't normally be used. */
for (i = 0; i < 10000; i++)
  {
    g_mem_chunk_free (mem_chunk, mem[i]);
  }

/* We are finished with the GMemChunk, so we destroy it. */
g_mem_chunk_destroy (mem_chunk);
```

Example 2. Using a GMemChunk with data structures

```
GMemChunk *array_mem_chunk;
GRealArray *array;

/* Create a GMemChunk to hold GRealArray structures, using the
   g_mem_chunk_create() convenience macro. We want 1024 atoms in each
   memory block, and we want to be able to free individual atoms. */
```

```
array_mem_chunk = g_mem_chunk_create (GRealArray, 1024, G_ALLOC_AND_FREE);

/* Allocate one atom, using the g_chunk_new() convenience macro. */
array = g_chunk_new (GRealArray, array_mem_chunk);

/* We can now use array just like a normal pointer to a structure. */
array->data            = NULL;
array->len             = 0;
array->alloc           = 0;
array->zero_terminated = (zero_terminated ? 1 : 0);
array->clear           = (clear ? 1 : 0);
array->elt_size        = elt_size;

/* We can free the element, so it can be reused. */
g_chunk_free (array, array_mem_chunk);

/* We destroy the GMemChunk when we are finished with it. */
g_mem_chunk_destroy (array_mem_chunk);
```

Details

GMemChunk

```
typedef struct _GMemChunk GMemChunk;
```

The GMemChunk struct is an opaque data structure representing a memory chunk. It should be accessed only through the use of the following functions.

G_ALLOC_AND_FREE

```
#define G_ALLOC_AND_FREE  2
```

Specifies the type of a GMemChunk. Used in g_mem_chunk_new() and g_mem_chunk_create() to specify that atoms will be freed individually.

G_ALLOC_ONLY

```
#define G_ALLOC_ONLY      1
```

Specifies the type of a GMemChunk. Used in g_mem_chunk_new() and g_mem_chunk_create() to specify that atoms will never be freed individually.

---

g_mem_chunk_new ()

```
GMemChunk*  g_mem_chunk_new                (const gchar *name,
                                            gint atom_size,
                                            gulong area_size,
                                            gint type);
```

Creates a new GMemChunk.

name :	a string to identify the GMemChunk. It is not copied so it should be valid for the lifetime of the GMemChunk. It is only used in g_mem_chunk_print(), which is used for debugging.
atom_size :	the size, in bytes, of each element in the GMemChunk.
area_size :	the size, in bytes, of each block of memory allocated to contain the atoms.
type :	the type of the GMemChunk. G_ALLOC_AND_FREE is used if the atoms will be freed individually. G_ALLOC_ONLY should be used if atoms will never be freed individually. G_ALLOC_ONLY is quicker, since it does not need to track free atoms, but it obviously wastes memory if you no longer need many of the atoms.
Returns :	the new GMemChunk.

---

g_mem_chunk_alloc ()

```
gpointer    g_mem_chunk_alloc              (GMemChunk *mem_chunk);
```

Allocates an atom of memory from a GMemChunk.

mem_chunk :	a GMemChunk.
Returns :	a pointer to the allocated atom.

---

g_mem_chunk_alloc0 ()

```
gpointer    g_mem_chunk_alloc0             (GMemChunk *mem_chunk);
```

Allocates an atom of memory from a GMemChunk, setting the memory to 0.

mem_chunk :	a GMemChunk.
Returns :	a pointer to the allocated atom.

---

g_mem_chunk_free ()

```
void        g_mem_chunk_free               (GMemChunk *mem_chunk,
                                            gpointer mem);
```

Frees an atom in a GMemChunk. This should only be called if the GMemChunk was created with G_ALLOC_AND_FREE. Otherwise it will simply return.

mem_chunk : a GMemChunk.
mem : a pointer to the atom to free.

---

g_mem_chunk_destroy ()

```
void        g_mem_chunk_destroy             (GMemChunk *mem_chunk);
```

Frees all of the memory allocated for a GMemChunk.

mem_chunk : a GMemChunk.

---

g_mem_chunk_create()

```
#define     g_mem_chunk_create(type, pre_alloc, alloc_type)
```

A convenience macro for creating a new GMemChunk. It calls `g_mem_chunk_new()`, using the given type to create the GMemChunk name. The atom size is determined using `sizeof()`, and the area size is calculated by multiplying the *pre_alloc* parameter with the atom size.

type : the type of the atoms, typically a structure name.
pre_alloc : the number of atoms to store in each block of memory.
alloc_type : the type of the GMemChunk. G_ALLOC_AND_FREE is used if the atoms will be freed individually. G_ALLOC_ONLY should be used if atoms will never be freed individually. G_ALLOC_ONLY is quicker, since it does not need to track free atoms, but it obviously wastes memory if you no longer need many of the atoms.
Returns : the new GMemChunk.

---

g_chunk_new()

```
#define     g_chunk_new(type, chunk)
```

A convenience macro to allocate an atom of memory from a GMemChunk. It calls `g_mem_chunk_alloc()` and casts the returned atom to a pointer to the given type, avoiding a type cast in the source code.

type : the type of the GMemChunk atoms, typically a structure name.
chunk : a GMemChunk.
Returns : a pointer to the allocated atom, cast to a pointer to *type*.

---

g_chunk_new0()


```
#define     g_chunk_new0(type, chunk)
```

A convenience macro to allocate an atom of memory from a GMemChunk. It calls `g_mem_chunk_alloc0()` and casts the returned atom to a pointer to the given type, avoiding a type cast in the source code.

type : the type of the GMemChunk atoms, typically a structure name.
chunk : a GMemChunk.
Returns : a pointer to the allocated atom, cast to a pointer to *type*.

---

g_chunk_free()

```
#define     g_chunk_free(mem, mem_chunk)
```

A convenience macro to free an atom of memory from a GMemChunk. It simply switches the arguments and calls `g_mem_chunk_free()` It is included simply to complement the other convenience macros, `g_chunk_new()` and `g_chunk_new0()`.

mem : a pointer to the atom to be freed.
mem_chunk : a GMemChunk.

---

g_mem_chunk_reset ()

```
void        g_mem_chunk_reset               (GMemChunk *mem_chunk);
```

Resets a GMemChunk to its initial state. It frees all of the currently allocated blocks of memory.

mem_chunk : a GMemChunk.

---

g_mem_chunk_clean ()

```
void        g_mem_chunk_clean               (GMemChunk *mem_chunk);
```

Frees any blocks in a GMemChunk which are no longer being used.

mem_chunk : a GMemChunk.

---

g_blow_chunks ()

```
void        g_blow_chunks                   (void);
```

Calls `g_mem_chunk_clean()` on all GMemChunk objects.

---

g_mem_chunk_info ()

```
void        g_mem_chunk_info                (void);
```

Outputs debugging information for all GMemChunk objects currently in use. It outputs the number of GMemChunk objects currently allocated, and calls `g_mem_chunk_print()` to output information on each one.

---

g_mem_chunk_print ()

```
void        g_mem_chunk_print               (GMemChunk *mem_chunk);
```

Outputs debugging information for a GMemChunk. It outputs the name of the GMemChunk (set with `g_mem_chunk_new()`), the number of bytes used, and the number of blocks of memory allocated.

mem_chunk : a GMemChunk.

<< GLib Data Types **Doubly-Linked Lists >>**

Doubly-Linked Lists

Doubly-Linked Lists — linked lists containing integer values or pointers to data, with the ability to iterate over the list in both directions.

Synopsis

```
#include <glib.h>


            GList;

GList*      g_list_append                   (GList *list,
                                             gpointer data);
GList*      g_list_prepend                  (GList *list,
                                             gpointer data);
GList*      g_list_insert                   (GList *list,
                                             gpointer data,
                                             gint position);
GList*      g_list_insert_before            (GList *list,
                                             GList *sibling,
                                             gpointer data);
GList*      g_list_insert_sorted            (GList *list,
                                             gpointer data,
                                             GCompareFunc func);
GList*      g_list_remove                   (GList *list,
                                             gconstpointer data);
GList*      g_list_remove_link              (GList *list,
                                             GList *llink);
GList*      g_list_delete_link              (GList *list,
                                             GList *link_);
GList*      g_list_remove_all               (GList *list,
                                             gconstpointer data);
void        g_list_free                     (GList *list);

GList*      g_list_alloc                    (void);
void        g_list_free_1                   (GList *list);

guint       g_list_length                   (GList *list);
GList*      g_list_copy                     (GList *list);
GList*      g_list_reverse                  (GList *list);
GList*      g_list_sort                     (GList *list,
                                             GCompareFunc compare_func);
gint        (*GCompareFunc)                 (gconstpointer a,
                                             gconstpointer b);
GList*      g_list_sort_with_data           (GList *list,
                                             GCompareDataFunc compare_func,
                                             gpointer user_data);
gint        (*GCompareDataFunc)             (gconstpointer a,
                                             gconstpointer b,
                                             gpointer user_data);
GList*      g_list_concat                   (GList *list1,
                                             GList *list2);
void        g_list_foreach                  (GList *list,
                                             GFunc func,
```



```
                                             gpointer user_data);
void        (*GFunc)                        (gpointer data,
                                             gpointer user_data);

GList*      g_list_first                    (GList *list);
GList*      g_list_last                     (GList *list);
#define     g_list_previous                 (list)
#define     g_list_next                     (list)
GList*      g_list_nth                      (GList *list,
                                             guint n);
gpointer    g_list_nth_data                 (GList *list,
                                             guint n);
GList*      g_list_nth_prev                 (GList *list,
                                             guint n);

GList*      g_list_find                     (GList *list,
                                             gconstpointer data);
GList*      g_list_find_custom              (GList *list,
                                             gconstpointer data,
                                             GCompareFunc func);
gint        g_list_position                 (GList *list,
                                             GList *llink);
gint        g_list_index                    (GList *list,
                                             gconstpointer data);

void        g_list_push_allocator           (GAllocator *allocator);
void        g_list_pop_allocator            (void);
```

Description

The GList structure and its associated functions provide a standard doubly-linked list data structure.

Each element in the list contains a piece of data, together with pointers which link to the previous and next elements in the list. Using these pointers it is possible to move through the list in both directions (unlike the Singly-Linked Lists which only allows movement through the list in the forward direction).

The data contained in each element can be either integer values, by using one of the Type Conversion Macros, or simply pointers to any type of data.

List elements are allocated in blocks using a GAllocator, which is more efficient than allocating elements individually.

Note that most of the GList functions expect to be passed a pointer to the first element in the list. The functions which insert elements return the new start of the list, which may have changed.

There is no function to create a GList. `NULL` is considered to be the empty list so you simply set a GList* to `NULL`.

To add elements, use `g_list_append()`, `g_list_prepend()`, `g_list_insert()` and `g_list_insert_sorted()`.

To remove elements, use `g_list_remove()`.

To find elements in the list use `g_list_first()`, `g_list_last()`, `g_list_next()`, `g_list_previous()`, `g_list_nth()`, `g_list_nth_data()`, `g_list_find()` and `g_list_find_custom()`.

To find the index of an element use `g_list_position()` and `g_list_index()`.

To call a function for each element in the list use `g_list_foreach()`.

To free the entire list, use `g_list_free()`.

Details

GList

```
typedef struct {
  gpointer data;
  GList *next;
  GList *prev;
} GList;
```

The GList struct is used for each element in a doubly-linked list. The *data* field holds the element's data, which can be a pointer to any kind of data, or any integer value using the Type Conversion Macros. The *next* and *prev* pointers are the links to the next and previous elements in the list.

---

g_list_append ()

```
GList*      g_list_append                 (GList *list,
                                           gpointer data);
```

Adds a new element on to the end of the list.

Note

The return value is the new start of the list, which may have changed, so make sure you store the new value.

```
  /* Notice that these are initialized to the empty list. */
  GList *list = NULL, *number_list = NULL;

  /* This is a list of strings. */
  list = g_list_append (list, "first");
  list = g_list_append (list, "second");

  /* This is a list of integers. */
  number_list = g_list_append (number_list, GINT_TO_POINTER (27));
  number_list = g_list_append (number_list, GINT_TO_POINTER (14));
```

list : a pointer to a GList.
data : the data for the new element.
Returns : the new start of the GList.

---

g_list_prepend ()


```
GList*      g_list_prepend                (GList *list,
                                           gpointer data);
```

Adds a new element on to the start of the list.

Note

The return value is the new start of the list, which may have changed, so make sure you store the new value.

```
  /* Notice that it is initialized to the empty list. */
  GList *list = NULL;
  list = g_list_prepend (list, "last");
  list = g_list_prepend (list, "first");
```

list : a pointer to a GList.
data : the data for the new element.
Returns : the new start of the GList.

---

g_list_insert ()

```
GList*      g_list_insert                 (GList *list,
                                           gpointer data,
                                           gint position);
```

Inserts a new element into the list at the given position.

list : a pointer to a GList.
data : the data for the new element.
position : the position to insert the element. If this is negative, or is larger than the number of elements in the list, the new element is added on to the end of the list.
Returns : the new start of the GList.

---

g_list_insert_before ()

```
GList*      g_list_insert_before          (GList *list,
                                           GList *sibling,
                                           gpointer data);
```

Inserts a new element into the list before the given position.

list : a pointer to a GList.
sibling : the list element before which the new element is inserted or `NULL` to insert at the end of the list.
data : the data for the new element.

Returns : the new start of the GList.

---

g_list_insert_sorted ()

```
GList*      g_list_insert_sorted            (GList *list,
                                             gpointer data,
                                             GCompareFunc func);
```

Inserts a new element into the list, using the given comparison function to determine its position.

list : a pointer to a GList.
data : the data for the new element.
func : the function to compare elements in the list. It should return a number > 0 if the first parameter comes after the second parameter in the sort order.
Returns : the new start of the GList.

---

g_list_remove ()

```
GList*      g_list_remove                   (GList *list,
                                             gconstpointer data);
```

Removes an element from a GList. If two elements contain the same data, only the first is removed. If none of the elements contain the data, the GList is unchanged.

list : a GList.
data : the data of the element to remove.
Returns : the new start of the GList.

---

g_list_remove_link ()

```
GList*      g_list_remove_link              (GList *list,
                                             GList *llink);
```

Removes an element from a GList, without freeing the element. The removed element's prev and next links are set to NULL, so that it becomes a self-contained list with one element.

list : a GList.
llink : an element in the GList.
Returns : the new start of the GList, without the element.

---

g_list_delete_link ()

```
GList*      g_list_delete_link              (GList *list,
```



```
                                             GList *link_);
```

Deletes the node *link_* from *list*.

list : a GList.
*link_* : node to delete from *list*.
Returns : the new head of *list*.

---

g_list_remove_all ()

```
GList*      g_list_remove_all               (GList *list,
                                             gconstpointer data);
```

Removes all list nodes with data equal to *data*. Returns the new head of the list. Contrast with g_list_remove() which removes only the first node matching the given data.

list : a GList.
data : data to remove.
Returns : new head of *list*.

---

g_list_free ()

```
void        g_list_free                     (GList *list);
```

Frees all of the memory used by a GList. The freed elements are added to the GAllocator free list.

Note

If list elements contain dynamically-allocated memory, they should be freed first.

list : a GList.

---

g_list_alloc ()

```
GList*      g_list_alloc                    (void);
```

Allocates space for one GList element. It is called by g_list_append(), g_list_prepend(), g_list_insert() and g_list_insert_sorted() and so is rarely used on its own.

Returns : a pointer to the newly-allocated GList element.

---

g_list_free_1 ()

```
void        g_list_free_1                   (GList *list);
```

Frees one GList element. It is usually used after `g_list_remove_link()`.

list : a GList element.

---

g_list_length ()

```
guint       g_list_length                   (GList *list);
```

Gets the number of elements in a GList.

list : a GList.
Returns : the number of elements in the GList.

---

g_list_copy ()

```
GList*      g_list_copy                     (GList *list);
```

Copies a GList.

Note that this is a "shallow" copy. If the list elements consist of pointers to data, the pointers are copied but the actual data isn't.

list : a GList.
Returns : a copy of *list*.

---

g_list_reverse ()

```
GList*      g_list_reverse                  (GList *list);
```

Reverses a GList. It simply switches the next and prev pointers of each element.

list : a GList.
Returns : the start of the reversed GList.

---

g_list_sort ()

```
GList*      g_list_sort                     (GList *list,
                                             GCompareFunc compare_func);
```

Sorts a GList using the given comparison function.

list : a GList.
compare_func : the comparison function used to sort the GList. This function is passed 2 elements of the GList and should return 0 if they are equal, a negative value if the first element comes before the second, or a positive value if the first element comes after the second.
Returns : the start of the sorted GList.

GCompareFunc ()

```
gint        (*GCompareFunc)                 (gconstpointer a,
                                             gconstpointer b);
```

Specifies the type of a comparison function used to compare two values. The function should return a negative integer if the first value comes before the second, 0 if they are equal, or a positive integer if the first value comes after the second.

a : a value.
b : a value to compare with.
Returns : negative value if *a* < *b*; zero if *a* = *b*; positive value if *a* > *b*.

---

g_list_sort_with_data ()

```
GList*      g_list_sort_with_data           (GList *list,
                                             GCompareDataFunc compare_func,
                                             gpointer user_data);
```

Like `g_list_sort()`, but the comparison function accepts a user data argument.

list : a GList.
compare_func : comparison function.
user_data : user data to pass to comparison function.
Returns : the new head of *list*.

GCompareDataFunc ()

```
gint        (*GCompareDataFunc)             (gconstpointer a,
                                             gconstpointer b,
                                             gpointer user_data);
```

Specifies the type of a comparison function used to compare two values. The function should return a negative integer if the first value comes before the second, 0 if they are equal, or a positive integer if the first value comes after the second.

a : a value.

b : a value to compare with.
user_data : user data to pass to comparison function.
Returns : negative value if *a* < *b*; zero if *a* = *b*; positive value if *a* > *b*.

---

g_list_concat ()

```
GList*      g_list_concat                   (GList *list1,
                                             GList *list2);
```

Adds the second GList onto the end of the first GList. Note that the elements of the second GList are not copied. They are used directly.

list1 : a GList.
list2 : the GList to add to the end of the first GList.
Returns : the start of the new GList.

---

g_list_foreach ()

```
void        g_list_foreach                  (GList *list,
                                             GFunc func,
                                             gpointer user_data);
```

Calls a function for each element of a GList.

list : a GList.
func : the function to call with each element's data.
user_data : user data to pass to the function.

GFunc ()

```
void        (*GFunc)                        (gpointer data,
                                             gpointer user_data);
```

Specifies the type of functions passed to `g_list_foreach()` and `g_slist_foreach()`.

data : the element's data.
user_data : user data passed to `g_list_foreach()` or `g_slist_foreach()`.

---

g_list_first ()

```
GList*      g_list_first                    (GList *list);
```


Gets the first element in a GList.

list : a GList.
Returns : the first element in a GList, or `NULL` if the GList has no elements.

---

g_list_last ()

```
GList*      g_list_last                     (GList *list);
```

Gets the last element in a GList.

list : a GList.
Returns : the last element in the GList, or `NULL` if the GList has no elements.

---

g_list_previous()

```
#define     g_list_previous(list)
```

A convenience macro to gets the previous element in a GList.

list : an element in a GList.
Returns : the previous element, or `NULL` if there are no previous elements.

---

g_list_next()

```
#define     g_list_next(list)
```

A convenience macro to gets the next element in a GList.

list : an element in a GList.
Returns : the next element, or `NULL` if there are no more elements.

---

g_list_nth ()

```
GList*      g_list_nth                      (GList *list,
                                             guint n);
```

Gets the element at the given position in a GList.

list : a GList.
n : the position of the element, counting from 0.

Returns : the element, or `NULL` if the position is off the end of the GList.

---

g_list_nth_data ()

```
gpointer    g_list_nth_data                 (GList *list,
                                             guint n);
```

Gets the data of the element at the given position.

list : a GList.
n : the position of the element.
Returns : the element's data, or `NULL` if the position is off the end of the GList.

---

g_list_nth_prev ()

```
GList*      g_list_nth_prev                 (GList *list,
                                             guint n);
```

Gets the element *n* places before *list*.

list : a GList.
n : the position of the element, counting from 0.
Returns : the element, or `NULL` if the position is off the end of the GList.

---

g_list_find ()

```
GList*      g_list_find                     (GList *list,
                                             gconstpointer data);
```

Finds the element in a GList which contains the given data.

list : a GList.
data : the element data to find.
Returns : the found GList element, or `NULL` if it is not found.

---

g_list_find_custom ()

```
GList*      g_list_find_custom              (GList *list,
                                             gconstpointer data,
                                             GCompareFunc func);
```

Finds an element in a GList, using a supplied function to find the desired element. It iterates over the

list, calling the given function which should return 0 when the desired element is found. The function takes two gconstpointer arguments, the GList element's data and the given user data.

list : a GList.
data : user data passed to the function.
func : the function to call for each element. It should return 0 when the desired element is found.
Returns : the found GList element, or `NULL` if it is not found.

---

g_list_position ()

```
gint        g_list_position                 (GList *list,
                                             GList *llink);
```

Gets the position of the given element in the GList (starting from 0).

list : a GList.
llink : an element in the GList.
Returns : the position of the element in the GList, or -1 if the element is not found.

---

g_list_index ()

```
gint        g_list_index                    (GList *list,
                                             gconstpointer data);
```

Gets the position of the element containing the given data (starting from 0).

list : a GList.
data : the data to find.
Returns : the index of the element containing the data, or -1 if the data is not found.

---

g_list_push_allocator ()

```
void        g_list_push_allocator           (GAllocator *allocator);
```

Sets the allocator to use to allocate GList elements. Use `g_list_pop_allocator()` to restore the previous allocator.

allocator : the GAllocator to use when allocating GList elements.

---

g_list_pop_allocator ()

```
void        g_list_pop_allocator            (void);
```

Restores the previous GAllocator, used when allocating GList elements.

<< Memory Chunks **Singly-Linked Lists >>**

Singly-Linked Lists

Singly-Linked Lists — linked lists containing integer values or pointers to data, limited to iterating over the list in one direction.

Synopsis

```
#include <glib.h>


            GSList;

GSList*     g_slist_alloc                   (void);
GSList*     g_slist_append                  (GSList *list,
                                             gpointer data);
GSList*     g_slist_prepend                 (GSList *list,
                                             gpointer data);
GSList*     g_slist_insert                  (GSList *list,
                                             gpointer data,
                                             gint position);
GSList*     g_slist_insert_before           (GSList *slist,
                                             GSList *sibling,
                                             gpointer data);
GSList*     g_slist_insert_sorted           (GSList *list,
                                             gpointer data,
                                             GCompareFunc func);
GSList*     g_slist_remove                  (GSList *list,
                                             gconstpointer data);
GSList*     g_slist_remove_link             (GSList *list,
                                             GSList *link_);
GSList*     g_slist_delete_link             (GSList *list,
                                             GSList *link_);
GSList*     g_slist_remove_all              (GSList *list,
                                             gconstpointer data);
void        g_slist_free                    (GSList *list);
void        g_slist_free_1                  (GSList *list);

guint       g_slist_length                  (GSList *list);
GSList*     g_slist_copy                    (GSList *list);
GSList*     g_slist_reverse                 (GSList *list);
GSList*     g_slist_sort                    (GSList *list,
                                             GCompareFunc compare_func);
GSList*     g_slist_sort_with_data          (GSList *list,
                                             GCompareDataFunc compare_func,
                                             gpointer user_data);
GSList*     g_slist_concat                  (GSList *list1,
                                             GSList *list2);
void        g_slist_foreach                 (GSList *list,
                                             GFunc func,
                                             gpointer user_data);

GSList*     g_slist_last                    (GSList *list);
#define     g_slist_next                    (slist)
GSList*     g_slist_nth                     (GSList *list,
                                             guint n);
```



```
gpointer    g_slist_nth_data                (GSList *list,
                                             guint n);

GSList*     g_slist_find                    (GSList *list,
                                             gconstpointer data);
GSList*     g_slist_find_custom             (GSList *list,
                                             gconstpointer data,
                                             GCompareFunc func);
gint        g_slist_position                (GSList *list,
                                             GSList *llink);
gint        g_slist_index                   (GSList *list,
                                             gconstpointer data);

void        g_slist_push_allocator          (GAllocator *allocator);
void        g_slist_pop_allocator           (void);
```

Description

The GSList structure and its associated functions provide a standard singly-linked list data structure.

Each element in the list contains a piece of data, together with a pointer which links to the next element in the list. Using this pointer it is possible to move through the list in one direction only (unlike the Doubly-Linked Lists which allow movement in both directions).

The data contained in each element can be either integer values, by using one of the Type Conversion Macros, or simply pointers to any type of data.

List elements are allocated in blocks using a GAllocator, which is more efficient than allocating elements individually.

Note that most of the GSList functions expect to be passed a pointer to the first element in the list. The functions which insert elements return the new start of the list, which may have changed.

There is no function to create a GSList. `NULL` is considered to be the empty list so you simply set a GSList* to `NULL`.

To add elements, use `g_slist_append()`, `g_slist_prepend()`, `g_slist_insert()` and `g_slist_insert_sorted()`.

To remove elements, use `g_slist_remove()`.

To find elements in the list use `g_slist_last()`, `g_slist_next()`, `g_slist_nth()`, `g_slist_nth_data()`, `g_slist_find()` and `g_slist_find_custom()`.

To find the index of an element use `g_slist_position()` and `g_slist_index()`.

To call a function for each element in the list use `g_slist_foreach()`.

To free the entire list, use `g_slist_free()`.

Details

GSList

```
typedef struct {
  gpointer data;
  GSList *next;
} GSList;
```

The GSList struct is used for each element in the singly-linked list. The *data* field holds the element's data, which can be a pointer to any kind of data, or any integer value using the Type Conversion Macros. The *next* field contains the link to the next element in the list.

---

g_slist_alloc ()

```
GSList*     g_slist_alloc                   (void);
```

Allocates space for one GSList element. It is called by the g_slist_append(), g_slist_prepend (), g_slist_insert() and g_slist_insert_sorted() functions and so is rarely used on its own.

Returns : a pointer to the newly-allocated GSList element.

---

g_slist_append ()

```
GSList*     g_slist_append                  (GSList *list,
                                             gpointer data);
```

Adds a new element on to the end of the list.

Note

The return value is the new start of the list, which may have changed, so make sure you store the new value.

```
/* Notice that these are initialized to the empty list. */
GSList *list = NULL, *number_list = NULL;

/* This is a list of strings. */
list = g_slist_append (list, "first");
list = g_slist_append (list, "second");

/* This is a list of integers. */
number_list = g_slist_append (number_list, GINT_TO_POINTER (27));
number_list = g_slist_append (number_list, GINT_TO_POINTER (14));
```

list : a GSList.
data : the data for the new element.
Returns : the new start of the GSList.

---

g_slist_prepend ()


```
GSList*     g_slist_prepend                 (GSList *list,
                                             gpointer data);
```

Adds a new element on to the start of the list.

Note

The return value is the new start of the list, which may have changed, so make sure you store the new value.

```
/* Notice that it is initialized to the empty list. */
GSList *list = NULL;
list = g_slist_prepend (list, "last");
list = g_slist_prepend (list, "first");
```

list : a GSList.
data : the data for the new element.
Returns : the new start of the GSList.

---

g_slist_insert ()

```
GSList*     g_slist_insert                  (GSList *list,
                                             gpointer data,
                                             gint position);
```

Inserts a new element into the list at the given position.

list : a GSList.
data : the data for the new element.
position : the position to insert the element. If this is negative, or is larger than the number of elements in the list, the new element is added on to the end of the list.
Returns : the new start of the GSList.

---

g_slist_insert_before ()

```
GSList*     g_slist_insert_before           (GSList *slist,
                                             GSList *sibling,
                                             gpointer data);
```

Inserts a node before *sibling* containing *data*. Returns the new head of the list.

slist : a GSList.
sibling : node to insert *data* before.
data : data to put in the newly-inserted node.
Returns : new head of the list.

---

g_slist_insert_sorted ()

```
GSList*     g_slist_insert_sorted           (GSList *list,
                                             gpointer data,
                                             GCompareFunc func);
```

Inserts a new element into the list, using the given comparison function to determine its position.

list : a GSList.
data : the data for the new element.
func : the function to compare elements in the list. It should return a number > 0 if the first parameter comes after the second parameter in the sort order.
Returns : the new start of the GSList.

---

g_slist_remove ()

```
GSList*     g_slist_remove                  (GSList *list,
                                             gconstpointer data);
```

Removes an element from a GSList. If two elements contain the same data, only the first is removed. If none of the elements contain the data, the GSList is unchanged.

list : a GSList.
data : the data of the element to remove.
Returns : the new start of the GSList.

---

g_slist_remove_link ()

```
GSList*     g_slist_remove_link             (GSList *list,
                                             GSList *link_);
```

Removes an element from a GSList, without freeing the element. The removed element's next link is set to NULL, so that it becomes a self-contained list with one element.

list : a GSList.
*link_* : an element in the GSList.
Returns : the new start of the GSList, without the element.

---

g_slist_delete_link ()

```
GSList*     g_slist_delete_link             (GSList *list,
                                             GSList *link_);
```

Deletes a node of *list*. Returns the new list head.

list : a GSList.
*link_* : node to delete.
Returns : new head of *list*.

---

g_slist_remove_all ()

```
GSList*     g_slist_remove_all              (GSList *list,
                                             gconstpointer data);
```

Removes all list nodes with data equal to *data*. Returns the new head of the list. Contrast with g_slist_remove() which removes only the first node matching the given data.

list : a GSList.
data : data to remove.
Returns : new head of *list*.

---

g_slist_free ()

```
void        g_slist_free                    (GSList *list);
```

Frees all of the memory used by a GSList. The freed elements are added to the GAllocator free list.

list : a GSList.

---

g_slist_free_1 ()

```
void        g_slist_free_1                  (GSList *list);
```

Frees one GSList element. It is usually used after g_slist_remove_link().

list : a GSList element.

---

g_slist_length ()

```
guint       g_slist_length                  (GSList *list);
```

Gets the number of elements in a GSList.

list : a GSList.
Returns : the number of elements in the GSList.

---

g_slist_copy ()

```
GSList*     g_slist_copy                    (GSList *list);
```

Copies a GSList.

Note that this is a "shallow" copy. If the list elements consist of pointers to data, the pointers are copied but the actual data isn't.

list : a GSList.
Returns : a copy of *list*.

---

g_slist_reverse ()

```
GSList*     g_slist_reverse                 (GSList *list);
```

Reverses a GSList.

list : a GSList.
Returns : the start of the reversed GSList.

---

g_slist_sort ()

```
GSList*     g_slist_sort                    (GSList *list,
                                             GCompareFunc compare_func);
```

Sorts a GSList using the given comparison function.

list : a GSList.
compare_func : qsort()-style comparison function.
Returns : the start of the sorted GList.

---

g_slist_sort_with_data ()

```
GSList*     g_slist_sort_with_data          (GSList *list,
```



```
                                             GCompareDataFunc compare_func,
                                             gpointer user_data);
```

Like g_slist_sort(), but the sort function accepts a user data argument.

list : a GSList
compare_func : comparison function.
user_data : data to pass to comparison function.
Returns : new head of the list.

---

g_slist_concat ()

```
GSList*     g_slist_concat                  (GSList *list1,
                                             GSList *list2);
```

Adds the second GSList onto the end of the first GSList. Note that the elements of the second GSList are not copied. They are used directly.

list1 : a GSList.
list2 : the GSList to add to the end of the first GSList.
Returns : the start of the new GSList.

---

g_slist_foreach ()

```
void        g_slist_foreach                 (GSList *list,
                                             GFunc func,
                                             gpointer user_data);
```

Calls a function for each element of a GSList.

list : a GSList.
func : the function to call with each element's data.
user_data : user data to pass to the function.

---

g_slist_last ()

```
GSList*     g_slist_last                    (GSList *list);
```

Gets the last element in a GSList.

list : a GSList.
Returns : the last element in the GSList, or NULL if the GSList has no elements.

---

g_slist_next()

```
#define     g_slist_next(slist)
```

A convenience macro to gets the next element in a GSList.

slist : an element in a GSList.
Returns : the next element, or NULL if there are no more elements.

---

g_slist_nth ()

```
GSList*     g_slist_nth                     (GSList *list,
                                             guint n);
```

Gets the element at the given position in a GSList.

list : a GSList.
n : the position of the element, counting from 0.
Returns : the element, or NULL if the position is off the end of the GSList.

---

g_slist_nth_data ()

```
gpointer    g_slist_nth_data                (GSList *list,
                                             guint n);
```

Gets the data of the element at the given position.

list : a GSList.
n : the position of the element.
Returns : the element's data, or NULL if the position is off the end of the GSList.

---

g_slist_find ()

```
GSList*     g_slist_find                    (GSList *list,
                                             gconstpointer data);
```

Finds the element in a GSList which contains the given data.

list : a GSList.
data : the element data to find.
Returns : the found GSList element, or NULL if it is not found.

---

g_slist_find_custom ()

```
GSList*     g_slist_find_custom             (GSList *list,
                                             gconstpointer data,
                                             GCompareFunc func);
```

Finds an element in a GSList, using a supplied function to find the desired element. It iterates over the list, calling the given function which should return 0 when the desired element is found. The function takes two gconstpointer arguments, the GSList element's data and the given user data.

list : a GSList.
data : user data passed to the function.
func : the function to call for each element. It should return 0 when the desired element is found.
Returns : the found GSList element, or NULL if it is not found.

---

g_slist_position ()

```
gint        g_slist_position                (GSList *list,
                                             GSList *llink);
```

Gets the position of the given element in the GSList (starting from 0).

list : a GSList.
llink : an element in the GSList.
Returns : the position of the element in the GSList, or -1 if the element is not found.

---

g_slist_index ()

```
gint        g_slist_index                   (GSList *list,
                                             gconstpointer data);
```

Gets the position of the element containing the given data (starting from 0).

list : a GSList.
data : the data to find.
Returns : the index of the element containing the data, or -1 if the data is not found.

---

g_slist_push_allocator ()

```
void        g_slist_push_allocator          (GAllocator *allocator);
```

Sets the allocator to use to allocate GSList elements. Use g_slist_pop_allocator() to restore the previous allocator.

allocator : the GAllocator to use when allocating GSList elements.

---

g_slist_pop_allocator ()

```
void        g_slist_pop_allocator           (void);
```

Restores the previous GAllocator, used when allocating GSList elements.

<< Doubly-Linked Lists **Double-ended Queues >>**

Double-ended Queues

Double-ended Queues — double-ended queue data structure.

Synopsis

```
#include <glib.h>


            GQueue;
GQueue*     g_queue_new                     (void);
void        g_queue_free                    (GQueue *queue);
gboolean    g_queue_is_empty                (GQueue *queue);
guint       g_queue_get_length              (GQueue *queue);
void        g_queue_reverse                 (GQueue *queue);
GQueue*     g_queue_copy                    (GQueue *queue);
void        g_queue_foreach                 (GQueue *queue,
                                             GFunc func,
                                             gpointer user_data);
GList*      g_queue_find                    (GQueue *queue,
                                             gconstpointer data);
GList*      g_queue_find_custom             (GQueue *queue,
                                             gconstpointer data,
                                             GCompareFunc func);
void        g_queue_sort                    (GQueue *queue,
                                             GCompareDataFunc compare_func,
                                             gpointer user_data);
void        g_queue_push_head               (GQueue *queue,
                                             gpointer data);
void        g_queue_push_tail               (GQueue *queue,
                                             gpointer data);
void        g_queue_push_nth                (GQueue *queue,
                                             gpointer data,
                                             gint n);
gpointer    g_queue_pop_head                (GQueue *queue);
gpointer    g_queue_pop_tail                (GQueue *queue);
gpointer    g_queue_pop_nth                 (GQueue *queue,
                                             guint n);
gpointer    g_queue_peek_head               (GQueue *queue);
gpointer    g_queue_peek_tail               (GQueue *queue);
gpointer    g_queue_peek_nth                (GQueue *queue,
                                             guint n);
gint        g_queue_index                   (GQueue *queue,
                                             gconstpointer data);
void        g_queue_remove                  (GQueue *queue,
                                             gconstpointer data);
void        g_queue_remove_all              (GQueue *queue,
                                             gconstpointer data);
void        g_queue_insert_before           (GQueue *queue,
                                             GList *sibling,
                                             gpointer data);
void        g_queue_insert_after            (GQueue *queue,
                                             GList *sibling,
                                             gpointer data);
void        g_queue_insert_sorted           (GQueue *queue,
                                             gpointer data,
                                             GCompareDataFunc func,
                                             gpointer user_data);
void        g_queue_push_head_link          (GQueue *queue,
                                             GList *link_);
void        g_queue_push_tail_link          (GQueue *queue,
                                             GList *link_);
void        g_queue_push_nth_link           (GQueue *queue,
                                             gint n,
                                             GList *link_);
GList*      g_queue_pop_head_link           (GQueue *queue);
GList*      g_queue_pop_tail_link           (GQueue *queue);
GList*      g_queue_pop_nth_link            (GQueue *queue,
                                             guint n);
GList*      g_queue_peek_head_link          (GQueue *queue);
GList*      g_queue_peek_tail_link          (GQueue *queue);
GList*      g_queue_peek_nth_link           (GQueue *queue,
                                             guint n);
gint        g_queue_link_index              (GQueue *queue,
                                             GList *link_);
void        g_queue_unlink                  (GQueue *queue,
                                             GList *link_);
void        g_queue_delete_link             (GQueue *queue,
                                             GList *link_);
```

Description

The GQueue structure and its associated functions provide a standard queue data structure. Internally, GQueue uses the same data structure as GList to store elements.

The data contained in each element can be either integer values, by using one of the Type Conversion Macros, or simply pointers to any type of data.

To create a new GQueue, use `g_queue_new()`.

To add elements, use `g_queue_push_head()`, `g_queue_push_head_link()`, `g_queue_push_tail()` and `g_queue_push_tail_link()`.

To remove elements, use `g_queue_pop_head()` and `g_queue_pop_tail()`.

To free the entire queue, use `g_queue_free()`.

Details

GQueue

```
typedef struct {
  GList *head;
  GList *tail;
  guint  length;
} GQueue;
```

Contains the public fields of a Queue.

GList *head; a pointer to the first element of the queue.

GList *tail; a pointer to the last element of the queue.
guint length; the number of elements in the queue.

---

g_queue_new ()

```
GQueue*     g_queue_new                     (void);
```

Creates a new GQueue.

Returns : a new GQueue.

---

g_queue_free ()

```
void        g_queue_free                    (GQueue *queue);
```

Frees the memory allocated for the GQueue.

queue : a GQueue.

---

g_queue_is_empty ()

```
gboolean    g_queue_is_empty                (GQueue *queue);
```

Returns TRUE if the queue is empty.

queue : a GQueue.
Returns : TRUE if the queue is empty.

---

g_queue_get_length ()

```
guint       g_queue_get_length              (GQueue *queue);
```

Returns the number of items in *queue*.

queue : a GQueue
Returns : The number of items in *queue*.

Since 2.4

---

g_queue_reverse ()

```
void        g_queue_reverse                 (GQueue *queue);
```

Reverses the order of the items in *queue*.

queue : a GQueue

Since 2.4

---

g_queue_copy ()

```
GQueue*     g_queue_copy                    (GQueue *queue);
```

Copies a *queue*. Note that is a shallow copy. If the elements in the queue consist of pointers to data, the pointers are copied, but the actual data is not.

queue : a GQueue
Returns : A copy of *queue*

Since 2.4

---

g_queue_foreach ()

```
void        g_queue_foreach                 (GQueue *queue,
                                             GFunc func,
                                             gpointer user_data);
```

Calls *func* for each element in the queue passing *user_data* to the function.

queue : a GQueue
func : the function to call for each element's data
user_data : user data to pass to *func*

Since 2.4

---

g_queue_find ()

```
GList*      g_queue_find                    (GQueue *queue,
                                             gconstpointer data);
```

Finds the first link in *queue* which contains *data*.

queue : a GQueue
data : data to find
Returns : The first link in *queue* which contains *data*.

Since 2.4

---

g_queue_find_custom ()

```
GList*      g_queue_find_custom             (GQueue *queue,
                                             gconstpointer data,
                                             GCompareFunc func);
```

Finds an element in a GQueue, using a supplied function to find the desired element. It iterates over the queue, calling the given function which should return 0 when the desired element is found. The function takes two gconstpointer arguments, the GQueue element's data and the given user data.

queue : a GQueue
data : user data passed to *func*
func : a GCompareFunc to call for each element. It should return 0 when the desired element is found
Returns : The found link, or `NULL` if it wasn't found

Since 2.4

---

g_queue_sort ()

```
void        g_queue_sort                    (GQueue *queue,
                                             GCompareDataFunc compare_func,
                                             gpointer user_data);
```

Sorts *queue* using *compare_func*.

queue : a GQueue
compare_func : the GCompareDataFunc used to sort *queue*. This function is passed two elements of the queue and should return 0 if they are equal, a negative value if the first comes before the second, and a positive value if the second comes before the first.
user_data : user data passed to *compare_func*

Since 2.4

---

g_queue_push_head ()

```
void        g_queue_push_head               (GQueue *queue,
                                             gpointer data);
```

Adds a new element at the head of the queue.

queue : a GQueue.
data : the data for the new element.

---

g_queue_push_tail ()

```
void        g_queue_push_tail               (GQueue *queue,
                                             gpointer data);
```

Adds a new element at the tail of the queue.

queue : a GQueue.
data : the data for the new element.

---

g_queue_push_nth ()

```
void        g_queue_push_nth                (GQueue *queue,
                                             gpointer data,
                                             gint n);
```

Inserts a new element into *queue* at the given position

queue : a GQueue
data : the data for the new element
n : the position to insert the new element. If *n* is negative or larger than the number of elements in the *queue*, the element is added to the end of the queue.

Since 2.4

---

g_queue_pop_head ()

```
gpointer    g_queue_pop_head                (GQueue *queue);
```

Removes the first element of the queue.

queue : a GQueue.
Returns : the data of the first element in the queue, or `NULL` if the queue is empty.

---

g_queue_pop_tail ()

```
gpointer    g_queue_pop_tail                (GQueue *queue);
```

Removes the last element of the queue.

queue : a GQueue.
Returns : the data of the last element in the queue, or `NULL` if the queue is empty.

---

g_queue_pop_nth ()

```
gpointer    g_queue_pop_nth                 (GQueue *queue,
                                             guint n);
```

Removes the *n*'th element of *queue*.

queue : a GQueue
n : the position of the element.
Returns : the element's data, or `NULL` if *n* is off the end of *queue*.

Since 2.4

---

g_queue_peek_head ()

```
gpointer    g_queue_peek_head               (GQueue *queue);
```

Returns the first element of the queue.

queue : a GQueue.
Returns : the data of the first element in the queue, or `NULL` if the queue is empty.

---

g_queue_peek_tail ()

```
gpointer    g_queue_peek_tail               (GQueue *queue);
```

Returns the last element of the queue.

queue : a GQueue.
Returns : the data of the last element in the queue, or `NULL` if the queue is empty.

---

g_queue_peek_nth ()

```
gpointer    g_queue_peek_nth                (GQueue *queue,
                                             guint n);
```

Returns the *n*'th element of *queue*.

queue : a GQueue
n : the position of the element.
Returns : The data for the *n*'th element of *queue*, or `NULL` if *n* is off the end of *queue*.

Since 2.4

---

g_queue_index ()

```
gint        g_queue_index                   (GQueue *queue,
                                             gconstpointer data);
```

Returns the position of the first element in *queue* which contains *data*.

queue : a GQueue
data : the data to find.
Returns : The position of the first element in *queue* which contains *data*, or -1 if no element in *queue* contains *data*.

Since 2.4

---

g_queue_remove ()

```
void        g_queue_remove                  (GQueue *queue,
                                             gconstpointer data);
```

Removes the first element in *queue* that contains *data*.

queue : a GQueue
data : data to remove.

Since 2.4

g_queue_remove_all ()

```
void        g_queue_remove_all              (GQueue *queue,
                                             gconstpointer data);
```

Remove all elemeents in *queue* which contains *data*.

queue : a GQueue
data : data to remove

Since 2.4

g_queue_insert_before ()

```
void        g_queue_insert_before           (GQueue *queue,
                                             GList *sibling,
                                             gpointer data);
```

Inserts *data* into *queue* before *sibling*.

sibling must be part of *queue*.

queue : a GQueue
sibling : a GList link that *must* be part of *queue*
data : the data to insert

Since 2.4

g_queue_insert_after ()

```
void        g_queue_insert_after            (GQueue *queue,
                                             GList *sibling,
                                             gpointer data);
```

Inserts *data* into *queue* after *sibling*

sibling must be part of *queue*

queue : a GQueue
sibling : a GList link that *must* be part of *queue*
data : the data to insert

Since 2.4

g_queue_insert_sorted ()

```
void        g_queue_insert_sorted           (GQueue *queue,
                                             gpointer data,
                                             GCompareDataFunc func,
                                             gpointer user_data);
```

Inserts *data* into *queue* using *func* to determine the new position.

queue : a GQueue
data : the data to insert
func : the GCompareDataFunc used to compare elements in the queue. It is called with two elements of the *queue* and *user_data*. It should return 0 if the elements are equal, a negative value if the first element comes before the second, and a positive value if the second element comes after the first.
user_data : user data passed to *func*.

Since 2.4

g_queue_push_head_link ()

```
void        g_queue_push_head_link          (GQueue *queue,
                                             GList *link_);
```

Adds a new element at the head of the queue.

queue : a GQueue.
*link_* : a single GList element, *not* a list with more than one element.

g_queue_push_tail_link ()

```
void        g_queue_push_tail_link          (GQueue *queue,
                                             GList *link_);
```

Adds a new element at the tail of the queue.

queue : a GQueue.
*link_* : a single GList element, *not* a list with more than one element.

g_queue_push_nth_link ()

```
void        g_queue_push_nth_link           (GQueue *queue,
                                             gint n,
                                             GList *link_);
```

Inserts *link* into *queue* at the given position.

queue : a GQueue
n : the position to insert the link. If this is negative or larger than the number of elements in *queue*, the link is added to the end of *queue*.
*link_* : the link to add to *queue*

Since 2.4

---

g_queue_pop_head_link ()

```
GList*      g_queue_pop_head_link           (GQueue *queue);
```

Removes the first element of the queue.

queue : a GQueue.
Returns : the GList element at the head of the queue, or `NULL` if the queue is empty.

---

g_queue_pop_tail_link ()

```
GList*      g_queue_pop_tail_link           (GQueue *queue);
```

Removes the last element of the queue.

queue : a GQueue.
Returns : the GList element at the tail of the queue, or `NULL` if the queue is empty.

---

g_queue_pop_nth_link ()

```
GList*      g_queue_pop_nth_link            (GQueue *queue,
                                             guint n);
```

Removes and returns the link at the given position.

queue : a GQueue
n : the link's position
Returns : The *n*'th link, or `NULL` if *n* is off the end of *queue*.

Since 2.4

---

g_queue_peek_head_link ()

```
GList*      g_queue_peek_head_link          (GQueue *queue);
```

Returns the first link in *queue*

queue : a GQueue
Returns : the first link in *queue*, or `NULL` if *queue* is empty

Since 2.4

---

g_queue_peek_tail_link ()

```
GList*      g_queue_peek_tail_link          (GQueue *queue);
```

Returns the last link *queue*.

queue : a GQueue
Returns : the last link in *queue*, or `NULL` if *queue* is empty

Since 2.4

---

g_queue_peek_nth_link ()

```
GList*      g_queue_peek_nth_link           (GQueue *queue,
                                             guint n);
```

Returns the link at the given position

queue : a GQueue
n : the position of the link
Returns : The link at the *n*'th position, or `NULL` if *n* is off the end of the list

Since 2.4

---

g_queue_link_index ()

```
gint        g_queue_link_index              (GQueue *queue,
                                             GList *link_);
```

Returns the position of *link_* in *queue*.

queue : a Gqueue
*link_* : A GList link
Returns : The position of *link_*, or -1 if the link is not part of *queue*

Since 2.4

---

g_queue_unlink ()

```
void        g_queue_unlink                  (GQueue *queue,
                                             GList *link_);
```

Unlinks *link_* so that it will no longer be part of *queue*. The link is not freed.

*link_* must be part of *queue*,

queue : a GQueue
*link_* : a GList link that *must* be part of *queue*

Since 2.4

---

g_queue_delete_link ()

```
void        g_queue_delete_link             (GQueue *queue,
                                             GList *link_);
```

Removes *link_* from *queue* and frees it.

*link_* must be part of *queue*.

queue : a GQueue
*link_* : a GList link that *must* be part of *queue*

Since 2.4

<< Singly-Linked Lists **Trash Stacks >>**

GLib Reference Manual

Trash Stacks

Trash Stacks — maintain a stack of unused allocated memory chunks.

Synopsis

```
#include <glib.h>


            GTrashStack;
void        g_trash_stack_push              (GTrashStack **stack_p,
                                             gpointer data_p);
gpointer    g_trash_stack_pop               (GTrashStack **stack_p);
gpointer    g_trash_stack_peek              (GTrashStack **stack_p);
guint       g_trash_stack_height            (GTrashStack **stack_p);
```

Description

A GTrashStack is an efficient way to keep a stack of unused allocated memory chunks. Each memory chunk is required to be large enough to hold a gpointer. This allows the stack to be maintained without any space overhead, since the stack pointers can be stored inside the memory chunks.

There is no function to create a GTrashStack. A `NULL` GTrashStack* is a perfectly valid empty stack.

Details

GTrashStack

```
typedef struct {
  GTrashStack *next;
} GTrashStack;
```

Each piece of memory that is pushed onto the stack is cast to a GTrashStack*.

GTrashStack *`next`; pointer to the previous element of the stack, gets stored in the first `sizeof (gpointer)` bytes of the element.

---

g_trash_stack_push ()

```
void        g_trash_stack_push              (GTrashStack **stack_p,
                                             gpointer data_p);
```

Pushes a piece of memory onto a GTrashStack.

`stack_p` : a pointer to a GTrashStack.
`data_p` : the piece of memory to push on the stack.

---

g_trash_stack_pop ()

```
gpointer    g_trash_stack_pop               (GTrashStack **stack_p);
```

Pops a piece of memory off a GTrashStack.

`stack_p` : a pointer to a GTrashStack.
Returns : the element at the top of the stack.

---

g_trash_stack_peek ()

```
gpointer    g_trash_stack_peek              (GTrashStack **stack_p);
```

Returns the element at the top of a GTrashStack.

`stack_p` : a pointer to a GTrashStack.
Returns : the element at the top of the stack.

---

g_trash_stack_height ()

```
guint       g_trash_stack_height            (GTrashStack **stack_p);
```

Returns the height of a GTrashStack.

`stack_p` : a pointer to a GTrashStack.
Returns : the height of the stack.

<< Double-ended Queues **Hash Tables >>**

Hash Tables

Hash Tables — associations between keys and values so that given a key the value can be found quickly.

Synopsis

```
#include <glib.h>


            GHashTable;
GHashTable* g_hash_table_new                (GHashFunc hash_func,
                                             GEqualFunc key_equal_func);
GHashTable* g_hash_table_new_full           (GHashFunc hash_func,
                                             GEqualFunc key_equal_func,
                                             GDestroyNotify key_destroy_func,
                                             GDestroyNotify value_destroy_func);
guint       (*GHashFunc)                    (gconstpointer key);
gboolean    (*GEqualFunc)                   (gconstpointer a,
                                             gconstpointer b);
void        g_hash_table_insert             (GHashTable *hash_table,
                                             gpointer key,
                                             gpointer value);
void        g_hash_table_replace            (GHashTable *hash_table,
                                             gpointer key,
                                             gpointer value);
guint       g_hash_table_size               (GHashTable *hash_table);
gpointer    g_hash_table_lookup             (GHashTable *hash_table,
                                             gconstpointer key);
gboolean    g_hash_table_lookup_extended    (GHashTable *hash_table,
                                             gconstpointer lookup_key,
                                             gpointer *orig_key,
                                             gpointer *value);
void        g_hash_table_foreach            (GHashTable *hash_table,
                                             GHFunc func,
                                             gpointer user_data);
gpointer    g_hash_table_find               (GHashTable *hash_table,
                                             GHRFunc predicate,
                                             gpointer user_data);
void        (*GHFunc)                       (gpointer key,
                                             gpointer value,
                                             gpointer user_data);
gboolean    g_hash_table_remove             (GHashTable *hash_table,
                                             gconstpointer key);
gboolean    g_hash_table_steal              (GHashTable *hash_table,
                                             gconstpointer key);
guint       g_hash_table_foreach_remove     (GHashTable *hash_table,
                                             GHRFunc func,
                                             gpointer user_data);
guint       g_hash_table_foreach_steal      (GHashTable *hash_table,
                                             GHRFunc func,
                                             gpointer user_data);
gboolean    (*GHRFunc)                      (gpointer key,
                                             gpointer value,
                                             gpointer user_data);
```



```
#define     g_hash_table_freeze             (hash_table)
#define     g_hash_table_thaw               (hash_table)
void        g_hash_table_destroy            (GHashTable *hash_table);

gboolean    g_direct_equal                  (gconstpointer v,
                                             gconstpointer v2);
guint       g_direct_hash                   (gconstpointer v);
gboolean    g_int_equal                     (gconstpointer v,
                                             gconstpointer v2);
guint       g_int_hash                      (gconstpointer v);
gboolean    g_str_equal                     (gconstpointer v,
                                             gconstpointer v2);
guint       g_str_hash                      (gconstpointer v);
```

Description

A GHashTable provides associations between keys and values which is optimized so that given a key, the associated value can be found very quickly.

Note that neither keys nor values are copied when inserted into the GHashTable, so they must exist for the lifetime of the GHashTable. This means that the use of static strings is OK, but temporary strings (i.e. those created in buffers and those returned by GTK+ widgets) should be copied with `g_strdup()` before being inserted.

If keys or values are dynamically allocated, you must be careful to ensure that they are freed when they are removed from the GHashTable, and also when they are overwritten by new insertions into the GHashTable. It is also not advisable to mix static strings and dynamically-allocated strings in a GHashTable, because it then becomes difficult to determine whether the string should be freed.

To create a GHashTable, use `g_hash_table_new()`.

To insert a key and value into a GHashTable, use `g_hash_table_insert()`.

To lookup a value corresponding to a given key, use `g_hash_table_lookup()` and `g_hash_table_lookup_extended()`.

To remove a key and value, use `g_hash_table_remove()`.

To call a function for each key and value pair use `g_hash_table_foreach()`.

To destroy a GHashTable use `g_hash_table_destroy()`.

Details

GHashTable

```
typedef struct _GHashTable GHashTable;
```

The GHashTable struct is an opaque data structure to represent a Hash Table. It should only be accessed via the following functions.

g_hash_table_new ()

```
GHashTable* g_hash_table_new                (GHashFunc hash_func,
                                             GEqualFunc key_equal_func);
```

Creates a new GHashTable.

hash_func : a function to create a hash value from a key. Hash values are used to determine where keys are stored within the GHashTable data structure. The `g_direct_hash()`, `g_int_hash()` and `g_str_hash()` functions are provided for some common types of keys. If hash_func is `NULL`, `g_direct_hash()` is used.

key_equal_func : a function to check two keys for equality. This is used when looking up keys in the GHashTable. The `g_direct_equal()`, `g_int_equal()` and `g_str_equal()` functions are provided for the most common types of keys. If *key_equal_func* is `NULL`, keys are compared directly in a similar fashion to `g_direct_equal()`, but without the overhead of a function call.

Returns : a new GHashTable.

---

g_hash_table_new_full ()

```
GHashTable* g_hash_table_new_full           (GHashFunc hash_func,
                                             GEqualFunc key_equal_func,
                                             GDestroyNotify key_destroy_func,
                                             GDestroyNotify value_destroy_func);
```

Creates a new GHashTable like `g_hash_table_new()` and allows to specify functions to free the memory allocated for the key and value that get called when removing the entry from the GHashTable.

hash_func : a function to create a hash value from a key.

key_equal_func : a function to check two keys for equality.

key_destroy_func : a function to free the memory allocated for the key used when removing the entry from the GHashTable or `NULL` if you don't want to supply such a function.

value_destroy_func : a function to free the memory allocated for the value used when removing the entry from the GHashTable or `NULL` if you don't want to supply such a function.

Returns : a new GHashTable.

GHashFunc ()

```
guint       (*GHashFunc)                    (gconstpointer key);
```

Specifies the type of the hash function which is passed to `g_hash_table_new()` when a GHashTable is created.

The function is passed a key and should return a guint hash value. The functions `g_direct_hash()`, `g_int_hash()` and `g_str_hash()` provide hash functions which can be used when the key is a gpointer, gint, and gchar* respectively.

FIXME: Need more here. The hash values should be evenly distributed over a fairly large range? The modulus is taken with the hash table size (a prime number) to find the 'bucket' to place each key into. The function should also be very fast, since it is called for each key lookup.

key : a key.

Returns : the hash value corresponding to the key.

GEqualFunc ()

```
gboolean    (*GEqualFunc)                   (gconstpointer a,
                                             gconstpointer b);
```

Specifies the type of a function used to test two values for equality. The function should return `TRUE` if both values are equal and `FALSE` otherwise.

a : a value.

b : a value to compare with.

Returns : `TRUE` if *a* = *b*; `FALSE` otherwise.

---

g_hash_table_insert ()

```
void        g_hash_table_insert             (GHashTable *hash_table,
                                             gpointer key,
                                             gpointer value);
```

Inserts a new key and value into a GHashTable.

If the key already exists in the GHashTable its current value is replaced with the new value. If you supplied a *value_destroy_func* when creating the GHashTable, the old value is freed using that function. If you supplied a *key_destroy_func* when creating the GHashTable, the passed key is freed using that function.

hash_table : a GHashTable.

key : a key to insert.

value : the value to associate with the key.

---

g_hash_table_replace ()

```
void        g_hash_table_replace            (GHashTable *hash_table,
                                             gpointer key,
                                             gpointer value);
```

Inserts a new key and value into a GHashTable similar to g_hash_table_insert(). The difference is that if the key already exists in the GHashTable, it gets replaced by the new key. If you supplied a *value_destroy_func* when creating the GHashTable, the old value is freed using that function. If you supplied a *key_destroy_func* when creating the GHashTable, the old key is freed using that function.

hash_table : a GHashTable.
key : a key to insert.
value : the value to associate with the key.

---

g_hash_table_size ()

```
guint       g_hash_table_size               (GHashTable *hash_table);
```

Returns the number of elements contained in the GHashTable.

hash_table : a GHashTable.
Returns : the number of key/value pairs in the GHashTable.

---

g_hash_table_lookup ()

```
gpointer    g_hash_table_lookup             (GHashTable *hash_table,
                                             gconstpointer key);
```

Looks up a key in a GHashTable. Note that this function cannot distinguish between a key that is not present and one which is present and has the value NULL. If you need this distinction, use g_hash_table_lookup_extended().

hash_table : a GHashTable.
key : the key to look up.
Returns : the associated value, or NULL if the key is not found.

---

g_hash_table_lookup_extended ()

```
gboolean    g_hash_table_lookup_extended    (GHashTable *hash_table,
                                             gconstpointer lookup_key,
                                             gpointer *orig_key,
                                             gpointer *value);
```

Looks up a key in the GHashTable, returning the original key and the associated value and a gboolean which is TRUE if the key was found. This is useful if you need to free the memory allocated for the original key, for example before calling g_hash_table_remove().

hash_table : a GHashTable.

lookup_key : the key to look up.
orig_key : returns the original key.
value : returns the value associated with the key.
Returns : TRUE if the key was found in the GHashTable.

---

g_hash_table_foreach ()

```
void        g_hash_table_foreach            (GHashTable *hash_table,
                                             GHFunc func,
                                             gpointer user_data);
```

Calls the given function for each of the key/value pairs in the GHashTable. The function is passed the key and value of each pair, and the given *user_data* parameter. The hash table may not be modified while iterating over it (you can't add/remove items). To remove all items matching a predicate, use g_hash_table_remove().

hash_table : a GHashTable.
func : the function to call for each key/value pair.
user_data : user data to pass to the function.

---

g_hash_table_find ()

```
gpointer    g_hash_table_find               (GHashTable *hash_table,
                                             GHRFunc predicate,
                                             gpointer user_data);
```

Calls the given function for key/value pairs in the GHashTable until *predicate* returns TRUE. The function is passed the key and value of each pair, and the given *user_data* parameter. The hash table may not be modified while iterating over it (you can't add/remove items).

hash_table : a GHashTable.
predicate : function to test the key/value pairs for a certain property.
user_data : user data to pass to the function.
Returns : The value of the first key/value pair is returned, for which func evaluates to TRUE. If no pair with the requested property is found, NULL is returned.

Since 2.4

GHFunc ()

```
void        (*GHFunc)                       (gpointer key,
                                             gpointer value,
                                             gpointer user_data);
```

Specifies the type of the function passed to g_hash_table_foreach(). It is called with each key/value pair, together with the *user_data* parameter which is passed to g_hash_table_foreach().

key : a key.
value : the value corresponding to the key.
user_data : user data passed to g_hash_table_foreach().

---

g_hash_table_remove ()

```
gboolean    g_hash_table_remove             (GHashTable *hash_table,
                                             gconstpointer key);
```

Removes a key and its associated value from a GHashTable.

If the GHashTable was created using g_hash_table_new_full(), the key and value are freed using the supplied destroy functions, otherwise you have to make sure that any dynamically allocated values are freed yourself.

hash_table : a GHashTable.
key : the key to remove.
Returns : TRUE if the key was found and removed from the GHashTable.

---

g_hash_table_steal ()

```
gboolean    g_hash_table_steal              (GHashTable *hash_table,
                                             gconstpointer key);
```

Removes a key and its associated value from a GHashTable without calling the key and value destroy functions.

hash_table : a GHashTable.
key : the key to remove.
Returns : TRUE if the key was found and removed from the GHashTable.

---

g_hash_table_foreach_remove ()

```
guint       g_hash_table_foreach_remove     (GHashTable *hash_table,
                                             GHRFunc func,
                                             gpointer user_data);
```

Calls the given function for each key/value pair in the GHashTable. If the function returns TRUE, then the key/value pair is removed from the GHashTable. If you supplied key or value destroy functions when creating the GHashTable, they are used to free the memory allocated for the removed keys and values.

hash_table : a GHashTable.
func : the function to call for each key/value pair.
user_data : user data to pass to the function.
Returns : the number of key/value pairs removed.

---

g_hash_table_foreach_steal ()

```
guint       g_hash_table_foreach_steal      (GHashTable *hash_table,
                                             GHRFunc func,
                                             gpointer user_data);
```

Calls the given function for each key/value pair in the GHashTable. If the function returns TRUE, then the key/value pair is removed from the GHashTable, but no key or value destroy functions are called.

hash_table : a GHashTable.
func : the function to call for each key/value pair.
user_data : user data to pass to the function.
Returns : the number of key/value pairs removed.

GHRFunc ()

```
gboolean    (*GHRFunc)                      (gpointer key,
                                             gpointer value,
                                             gpointer user_data);
```

Specifies the type of the function passed to g_hash_table_foreach_remove(). It is called with each key/value pair, together with the *user_data* parameter passed to g_hash_table_foreach_remove(). It should return TRUE if the key/value pair should be removed from the GHashTable.

key : a key.
value : the value associated with the key.
user_data : user data passed to g_hash_table_remove().
Returns : TRUE if the key/value pair should be removed from the GHashTable.

---

g_hash_table_freeze()

```
#define     g_hash_table_freeze(hash_table)
```

Warning

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

This function is deprecated and will be removed in the next major release of GLib. It does nothing.

hash_table : a GHashTable

---

g_hash_table_thaw()

```
#define     g_hash_table_thaw(hash_table)
```

Warning

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

This function is deprecated and will be removed in the next major release of GLib. It does nothing.

hash_table : a GHashTable

---

g_hash_table_destroy ()

```
void        g_hash_table_destroy            (GHashTable *hash_table);
```

Destroys the GHashTable. If keys and/or values are dynamically allocated, you should either free them first or create the GHashTable using `g_hash_table_new_full()`. In the latter case the destroy functions you supplied will be called on all keys and values before destroying the GHashTable.

hash_table : a GHashTable.

---

g_direct_equal ()

```
gboolean    g_direct_equal                  (gconstpointer v,
                                             gconstpointer v2);
```

Compares two gpointer arguments and returns `TRUE` if they are equal. It can be passed to `g_hash_table_new()` as the *key_equal_func* parameter, when using pointers as keys in a GHashTable.

v : a key.
v2 : a key to compare with *v*.
Returns : `TRUE` if the two keys match.

---

g_direct_hash ()

```
guint       g_direct_hash                   (gconstpointer v);
```

Converts a gpointer to a hash value. It can be passed to `g_hash_table_new()` as the *hash_func*

parameter, when using pointers as keys in a GHashTable.

v : a gpointer key.
Returns : a hash value corresponding to the key.

---

g_int_equal ()

```
gboolean    g_int_equal                     (gconstpointer v,
                                             gconstpointer v2);
```

Compares the two gint values being pointed to and returns `TRUE` if they are equal. It can be passed to `g_hash_table_new()` as the *key_equal_func* parameter, when using pointers to integers as keys in a GHashTable.

v : a pointer to a gint key.
v2 : a pointer to a gint key to compare with *v*.
Returns : `TRUE` if the two keys match.

---

g_int_hash ()

```
guint       g_int_hash                      (gconstpointer v);
```

Converts a pointer to a gint to a hash value. It can be passed to `g_hash_table_new()` as the *hash_func* parameter, when using pointers to integers values as keys in a GHashTable.

v : a pointer to a gint key.
Returns : a hash value corresponding to the key.

---

g_str_equal ()

```
gboolean    g_str_equal                     (gconstpointer v,
                                             gconstpointer v2);
```

Compares two strings and returns `TRUE` if they are equal. It can be passed to `g_hash_table_new()` as the *key_equal_func* parameter, when using strings as keys in a GHashTable.

v : a key.
v2 : a key to compare with *v*.
Returns : `TRUE` if the two keys match.

---

g_str_hash ()

```
guint       g_str_hash                      (gconstpointer v);
```

Converts a string to a hash value. It can be passed to `g_hash_table_new()` as the *`hash_func`* parameter, when using strings as keys in a GHashTable.

v : a string key.

Returns : a hash value corresponding to the key.

<< Trash Stacks **Strings >>**

Strings

Strings — text buffers which grow automatically as text is added.

Synopsis

```
#include <glib.h>


            GString;
GString*    g_string_new                    (const gchar *init);
GString*    g_string_new_len                (const gchar *init,
                                             gssize len);
GString*    g_string_sized_new              (gsize dfl_size);
GString*    g_string_assign                 (GString *string,
                                             const gchar *rval);
#define     g_string_sprintf
#define     g_string_sprintfa
void        g_string_printf                 (GString *string,
                                             const gchar *format,
                                             ...);
void        g_string_append_printf          (GString *string,
                                             const gchar *format,
                                             ...);
GString*    g_string_append                 (GString *string,
                                             const gchar *val);
GString*    g_string_append_c               (GString *string,
                                             gchar c);
GString*    g_string_append_unichar         (GString *string,
                                             gunichar wc);
GString*    g_string_append_len             (GString *string,
                                             const gchar *val,
                                             gssize len);
GString*    g_string_prepend                (GString *string,
                                             const gchar *val);
GString*    g_string_prepend_c              (GString *string,
                                             gchar c);
GString*    g_string_prepend_unichar        (GString *string,
                                             gunichar wc);
GString*    g_string_prepend_len            (GString *string,
                                             const gchar *val,
                                             gssize len);
GString*    g_string_insert                 (GString *string,
                                             gssize pos,
                                             const gchar *val);
GString*    g_string_insert_c               (GString *string,
                                             gssize pos,
                                             gchar c);
GString*    g_string_insert_unichar         (GString *string,
                                             gssize pos,
                                             gunichar wc);
GString*    g_string_insert_len             (GString *string,
                                             gssize pos,
                                             const gchar *val,
                                             gssize len);
```



```
GString*    g_string_erase                  (GString *string,
                                             gssize pos,
                                             gssize len);
GString*    g_string_truncate               (GString *string,
                                             gsize len);
GString*    g_string_set_size               (GString *string,
                                             gsize len);
gchar*      g_string_free                   (GString *string,
                                             gboolean free_segment);

GString*    g_string_up                     (GString *string);
GString*    g_string_down                   (GString *string);

guint       g_string_hash                   (const GString *str);
gboolean    g_string_equal                  (const GString *v,
                                             const GString *v2);
```

Description

A GString is similar to a standard C string, except that it grows automatically as text is appended or inserted. Also, it stores the length of the string, so can be used for binary data with embedded nul bytes.

Details

GString

```
typedef struct {
  gchar  *str;
  gsize len;
  gsize allocated_len;
} GString;
```

The GString struct contains the public fields of a GString. The *str* field points to the character data. It may move as text is added. The *len* field contains the length of the string, not including the terminating nul character.

The *str* field is nul-terminated and so can be used as an ordinary C string. But it may be moved when text is appended or inserted into the string.

---

g_string_new ()

```
GString*    g_string_new                    (const gchar *init);
```

Creates a new GString, initialized with the given string.

init : the initial text to copy into the string.
Returns : the new GString.

---

g_string_new_len ()

```
GString*    g_string_new_len                (const gchar *init,
                                             gssize len);
```

Creates a new GString with *len* bytes of the *init* buffer. Because a length is provided, *init* need not be nul-terminated, and can contain embedded nul bytes.

init : initial contents of string.
len : length of *init* to use.
Returns : a new GString.

---

g_string_sized_new ()

```
GString*    g_string_sized_new              (gsize dfl_size);
```

Creates a new GString, with enough space for *dfl_size* characters. This is useful if you are going to add a lot of text to the string and don't want it to be reallocated too often.

dfl_size : the default size of the space allocated to hold the string.
Returns : the new GString.

---

g_string_assign ()

```
GString*    g_string_assign                 (GString *string,
                                             const gchar *rval);
```

Copies the characters from a string into a GString, destroying any previous contents. It is rather like the standard `strcpy()` function, except that you do not have to worry about having enough space to copy the string.

string : the destination GString. Its current contents are destroyed.
rval :
Returns : the destination GString.

---

g_string_sprintf

```
#define     g_string_sprintf
```

Warning

`g_string_sprintf` is deprecated and should not be used in newly-written code. This function has been renamed to `g_string_printf()`.

Writes a formatted string into a GString. This is similar to the standard `sprintf()` function, except that the GString buffer automatically expands to contain the results. The previous contents of the GString are destroyed.

---

g_string_sprintfa

```
#define     g_string_sprintfa
```

Warning

`g_string_sprintfa` is deprecated and should not be used in newly-written code. This function has been renamed to `g_string_append_printf()`.

Appends a formatted string onto the end of a GString. This function is is similar to `g_string_sprintf()` except that the text is appended to the GString.

---

g_string_printf ()

```
void        g_string_printf                 (GString *string,
                                             const gchar *format,
                                             ...);
```

Writes a formatted string into a GString. This is similar to the standard `sprintf()` function, except that the GString buffer automatically expands to contain the results. The previous contents of the GString are destroyed.

string : a GString.
format : the string format. See the `printf()` documentation.
... : the parameters to insert into the format string.

---

g_string_append_printf ()

```
void        g_string_append_printf          (GString *string,
                                             const gchar *format,
                                             ...);
```

Appends a formatted string onto the end of a GString. This function is is similar to `g_string_printf()` except that the text is appended to the GString.

string : a GString.
format : the string format. See the `printf()` documentation.
... : the parameters to insert into the format string.

---

g_string_append ()

```
GString*    g_string_append                 (GString *string,
                                             const gchar *val);
```

Adds a string onto the end of a GString, expanding it if necessary.

string : a GString.
val : the string to append onto the end of the GString.
Returns : the GString.

---

g_string_append_c ()

```
GString*    g_string_append_c               (GString *string,
                                             gchar c);
```

Adds a character onto the end of a GString, expanding it if necessary.

string : a GString.
c : the character to append onto the end of the GString.
Returns : the GString.

---

g_string_append_unichar ()

```
GString*    g_string_append_unichar         (GString *string,
                                             gunichar wc);
```

Converts a Unicode character into UTF-8, and appends it to the string.

string : a GString
wc : a Unicode character
Returns : *string*

---

g_string_append_len ()

```
GString*    g_string_append_len             (GString *string,
                                             const gchar *val,
                                             gssize len);
```

Appends *len* bytes of *val* to *string*. Because *len* is provided, *val* may contain embedded nuls and need not be nul-terminated.

string : a GString.
bytes to append.

val :
len : number of bytes of *val* to use.
Returns : the GString.

---

g_string_prepend ()

```
GString*    g_string_prepend                (GString *string,
                                             const gchar *val);
```

Adds a string on to the start of a GString, expanding it if necessary.

string : a GString.
val : the string to prepend on the start of the GString.
Returns : the GString.

---

g_string_prepend_c ()

```
GString*    g_string_prepend_c              (GString *string,
                                             gchar c);
```

Adds a character onto the start of a GString, expanding it if necessary.

string : a GString.
c : the character to prepend on the start of the GString.
Returns : the GString.

---

g_string_prepend_unichar ()

```
GString*    g_string_prepend_unichar        (GString *string,
                                             gunichar wc);
```

Converts a Unicode character into UTF-8, and prepends it to the string.

string : a GString.
wc : a Unicode character.
Returns : *string*.

---

g_string_prepend_len ()

```
GString*    g_string_prepend_len            (GString *string,
                                             const gchar *val,
                                             gssize len);
```

Prepends *len* bytes of *val* to *string*. Because *len* is provided, *val* may contain embedded nuls and need not be nul-terminated.

string : a GString.
val : bytes to prepend.
len : number of bytes in *val* to prepend.
Returns : the GString passed in.

---

g_string_insert ()

```
GString*    g_string_insert                 (GString *string,
                                             gssize pos,
                                             const gchar *val);
```

Inserts a copy of a string into a GString, expanding it if necessary.

string : a GString.
pos : the position to insert the copy of the string.
val : the string to insert.
Returns : the GString.

---

g_string_insert_c ()

```
GString*    g_string_insert_c               (GString *string,
                                             gssize pos,
                                             gchar c);
```

Inserts a character into a GString, expanding it if necessary.

string : a GString.
pos : the position to insert the character.
c : the character to insert.
Returns : the GString.

---

g_string_insert_unichar ()

```
GString*    g_string_insert_unichar         (GString *string,
                                             gssize pos,
                                             gunichar wc);
```

Converts a Unicode character into UTF-8, and insert it into the string at the given position.

string : a GString
pos : the position at which to insert character, or -1 to append at the end of the string.

wc : a Unicode character
Returns : *string*

---

g_string_insert_len ()

```
GString*    g_string_insert_len             (GString *string,
                                             gssize pos,
                                             const gchar *val,
                                             gssize len);
```

Inserts *len* bytes of *val* into *string* at *pos*. Because *len* is provided, *val* may contain embedded nuls and need not be nul-terminated. If *pos* is -1, bytes are inserted at the end of the string.

string : a GString.
pos : position in *string* where insertion should happen, or -1 for at the end.
val : bytes to insert.
len : number of bytes of *val* to insert.
Returns : the GString.

---

g_string_erase ()

```
GString*    g_string_erase                  (GString *string,
                                             gssize pos,
                                             gssize len);
```

Removes *len* characters from a GString, starting at position *pos*. The rest of the GString is shifted down to fill the gap.

string : a GString.
pos : the position of the characters to remove.
len : the number of characters to remove, or -1 to remove all following characters.
Returns : the GString.

---

g_string_truncate ()

```
GString*    g_string_truncate               (GString *string,
                                             gsize len);
```

Cuts off the end of the GString, leaving the first *len* characters.

string : a GString.
len : the new size of the GString.
Returns : the GString.

---

g_string_set_size ()

```
GString*    g_string_set_size               (GString *string,
                                             gsize len);
```

Sets the length of a GString. If the length is less than the current length, the string will be truncated. If the length is greater than the current length, the contents of the newly added area are undefined. (However, as always, string->str[string->len] will be a nul byte.)

string : a GString
len : the new length
Returns : *string*

---

g_string_free ()

```
gchar*      g_string_free                   (GString *string,
                                             gboolean free_segment);
```

Frees the memory allocated for the GString. If *free_segment* is TRUE it also frees the character data.

string : a GString.
free_segment : if TRUE the actual character data is freed as well.
Returns : the character data of *string* (i.e. NULL if *free_segment* is TRUE)

---

g_string_up ()

```
GString*    g_string_up                     (GString *string);
```

Warning

g_string_up is deprecated and should not be used in newly-written code. This function uses the locale-specific toupper() function, which is almost never the right thing. Use g_string_ascii_up() or g_utf8_strup() instead.

Converts a GString to uppercase.

string : a GString
Returns : the GString

---

g_string_down ()

```
GString*    g_string_down                   (GString *string);
```


Warning

g_string_down is deprecated and should not be used in newly-written code. This function uses the locale-specific tolower() function, which is almost never the right thing. Use g_string_ascii_down() or g_utf8_strdown() instead.

Converts a GString to lowercase.

string : a GString
Returns : the GString.

---

g_string_hash ()

```
guint       g_string_hash                   (const GString *str);
```

Creates a hash code for *str*; for use with GHashTable.

str : a string to hash.
Returns : hash code for *str*.

---

g_string_equal ()

```
gboolean    g_string_equal                  (const GString *v,
                                             const GString *v2);
```

Compares two strings for equality, returning TRUE if they are equal. For use with GHashTable.

v : a GString.
v2 : another GString.
Returns : TRUE if they strings are the same length and contain the same bytes.

<< Hash Tables **String Chunks >>**

String Chunks

String Chunks — efficient storage of groups of strings.

Synopsis

```
#include <glib.h>


            GStringChunk;
GStringChunk* g_string_chunk_new            (gsize size);
gchar*      g_string_chunk_insert           (GStringChunk *chunk,
                                             const gchar *string);
gchar*      g_string_chunk_insert_const     (GStringChunk *chunk,
                                             const gchar *string);
gchar*      g_string_chunk_insert_len       (GStringChunk *chunk,
                                             const gchar *string,
                                             gssize len);
void        g_string_chunk_free             (GStringChunk *chunk);
```

Description

String chunks are used to store groups of strings. Memory is allocated in blocks, and as strings are added to the GStringChunk they are copied into the next free position in a block. When a block is full a new block is allocated.

When storing a large number of strings, string chunks are more efficient than using `g_strdup()` since fewer calls to `malloc()` are needed, and less memory is wasted in memory allocation overheads.

By adding strings with `g_string_chunk_insert_const()` it is also possible to remove duplicates.

To create a new GStringChunk use `g_string_chunk_new()`.

To add strings to a GStringChunk use `g_string_chunk_insert()`.

To add strings to a GStringChunk, but without duplicating strings which are already in the GStringChunk, use `g_string_chunk_insert_const()`.

To free the entire GStringChunk use `g_string_chunk_free()`. It is not possible to free individual strings.

Details

GStringChunk


```
typedef struct _GStringChunk GStringChunk;
```

An opaque data structure representing String Chunks. It should only be accessed by using the following functions.

---

g_string_chunk_new ()

```
GStringChunk* g_string_chunk_new            (gsize size);
```

Creates a new GStringChunk.

size : the default size of the blocks of memory which are allocated to store the strings. If a particular string is larger than this default size, a larger block of memory will be allocated for it.

Returns : a new GStringChunk.

---

g_string_chunk_insert ()

```
gchar*      g_string_chunk_insert           (GStringChunk *chunk,
                                             const gchar *string);
```

Adds a copy of *string* to the GStringChunk. It returns a pointer to the new copy of the string in the GStringChunk. The characters in the string can be changed, if necessary, though you should not change anything after the end of the string.

Unlike `g_string_chunk_insert_const()`, this function does not check for duplicates. Also strings added with `g_string_chunk_insert()` will not be searched by `g_string_chunk_insert_const()` when looking for duplicates.

chunk : a GStringChunk.

string : the string to add.

Returns : a pointer to the copy of *string* within the GStringChunk.

---

g_string_chunk_insert_const ()

```
gchar*      g_string_chunk_insert_const     (GStringChunk *chunk,
                                             const gchar *string);
```

Adds a copy of *string* to the GStringChunk, unless the same string has already been added to the GStringChunk with `g_string_chunk_insert_const()`.

This function is useful if you need to copy a large number of strings but do not want to waste space storing duplicates. But you must remember that there may be several pointers to the same string, and so any changes made to the strings should be done very carefully.

Note that `g_string_chunk_insert_const()` will not return a pointer to a string added with `g_string_chunk_insert()`, even if they do match.

chunk : a GStringChunk.
string : the string to add.
Returns : a pointer to the new or existing copy of *string* within the GStringChunk.

---

g_string_chunk_insert_len ()

```
gchar*      g_string_chunk_insert_len       (GStringChunk *chunk,
                                             const gchar *string,
                                             gssize len);
```

Adds a copy of the first *len* bytes of *string* to the GStringChunk. The copy is nul-terminated.

The characters in the string can be changed, if necessary, though you should not change anything after the end of the string.

chunk : a GStringChunk
string : bytes to insert
len : number of bytes of *string* to insert, or -1 to insert a nul-terminated string.
Returns : a pointer to the copy of *string* within the GStringChunk

Since 2.4

---

g_string_chunk_free ()

```
void        g_string_chunk_free             (GStringChunk *chunk);
```

Frees all memory allocated by the GStringChunk. After calling `g_string_chunk_free()` it is not safe to access any of the strings which were contained within it.

chunk : a GStringChunk.

<< Strings **Arrays >>**

Arrays

Arrays — arrays of arbitrary elements which grow automatically as elements are added.

Synopsis

```
#include <glib.h>


            GArray;
GArray*     g_array_new                     (gboolean zero_terminated,
                                             gboolean clear_,
                                             guint element_size);
GArray*     g_array_sized_new               (gboolean zero_terminated,
                                             gboolean clear_,
                                             guint element_size,
                                             guint reserved_size);
#define     g_array_append_val              (a,v)
GArray*     g_array_append_vals             (GArray *array,
                                             gconstpointer data,
                                             guint len);
#define     g_array_prepend_val             (a,v)
GArray*     g_array_prepend_vals            (GArray *array,
                                             gconstpointer data,
                                             guint len);
#define     g_array_insert_val              (a,i,v)
GArray*     g_array_insert_vals             (GArray *array,
                                             guint index_,
                                             gconstpointer data,
                                             guint len);
GArray*     g_array_remove_index            (GArray *array,
                                             guint index_);
GArray*     g_array_remove_index_fast       (GArray *array,
                                             guint index_);
GArray*     g_array_remove_range            (GArray *array,
                                             guint index_,
                                             guint length);
void        g_array_sort                    (GArray *array,
                                             GCompareFunc compare_func);
void        g_array_sort_with_data          (GArray *array,
                                             GCompareDataFunc compare_func,
                                             gpointer user_data);
#define     g_array_index                   (a,t,i)
GArray*     g_array_set_size                (GArray *array,
                                             guint length);
gchar*      g_array_free                    (GArray *array,
                                             gboolean free_segment);
```

Description

Arrays are similar to standard C arrays, except that they grow automatically as elements are added.

Array elements can be of any size (though all elements of one array are the same size), and the array

can be automatically cleared to '0's and zero-terminated.

To create a new array use `g_array_new()`.

To add elements to an array, use `g_array_append_val()`, `g_array_append_vals()`, `g_array_prepend_val()`, and `g_array_prepend_vals()`.

To access an element of an array, use `g_array_index()`.

To set the size of an array, use `g_array_set_size()`.

To free an array, use `g_array_free()`.

Example 3. Using a GArray to store gint values

```
  GArray *garray;
  gint i;

  /* We create a new array to store gint values.
     We don't want it zero-terminated or cleared to 0's. */
  garray = g_array_new (FALSE, FALSE, sizeof (gint));
  for (i = 0; i < 10000; i++)
    g_array_append_val (garray, i);

  for (i = 0; i < 10000; i++)
    if (g_array_index (garray, gint, i) != i)
      g_print ("ERROR: got %d instead of %d\n",
               g_array_index (garray, gint, i), i);

  g_array_free (garray, TRUE);
```

Details

GArray

```
typedef struct {
  gchar *data;
  guint len;
} GArray;
```

Contains the public fields of an Array.

gchar *data; a pointer to the element data. The data may be moved as elements are added to the GArray.

guint len; the number of elements in the GArray.

---

g_array_new ()

```
GArray*     g_array_new                     (gboolean zero_terminated,
                                             gboolean clear_,
                                             guint element_size);
```

Creates a new GArray.

zero_terminated : TRUE if the array should have an extra element at the end which is set to 0.
*clear_* : TRUE if GArray elements should be automatically cleared to 0 when they are allocated.
element_size : the size of each element in bytes.
Returns : the new GArray.

---

g_array_sized_new ()

```
GArray*     g_array_sized_new               (gboolean zero_terminated,
                                             gboolean clear_,
                                             guint element_size,
                                             guint reserved_size);
```

Creates a new GArray with *reserved_size* elements preallocated. This avoids frequent reallocation, if you are going to add many elements to the array. Note however that the size of the array is still 0.

zero_terminated : TRUE if the array should have an extra element at the end with all bits cleared.
*clear_* : TRUE if all bits in the array should be cleared to 0 on allocation.
element_size : size of each element in the array.
reserved_size : number of elements preallocated.
Returns : the new GArray.

---

g_array_append_val()

```
#define     g_array_append_val(a,v)
```

Adds the value on to the end of the array. The array will grow in size automatically if necessary.

Note

g_array_append_val() is a macro which uses a reference to the value parameter *v*. This means that you cannot use it with literal values such as "27". You must use variables.

a : a GArray.
v : the value to append to the GArray.
Returns : the GArray.

---

g_array_append_vals ()


```
GArray*     g_array_append_vals             (GArray *array,
                                             gconstpointer data,
                                             guint len);
```

Adds *len* elements onto the end of the array.

array : a GArray.
data : a pointer to the elements to append to the end of the array.
len : the number of elements to append.
Returns : the GArray.

---

g_array_prepend_val()

```
#define     g_array_prepend_val(a,v)
```

Adds the value on to the start of the array. The array will grow in size automatically if necessary.

This operation is slower than g_array_append_val() since the existing elements in the array have to be moved to make space for the new element.

Note

g_array_prepend_val() is a macro which uses a reference to the value parameter *v*. This means that you cannot use it with literal values such as "27". You must use variables.

a : a GArray.
v : the value to prepend to the GArray.
Returns : the GArray.

---

g_array_prepend_vals ()

```
GArray*     g_array_prepend_vals            (GArray *array,
                                             gconstpointer data,
                                             guint len);
```

Adds *len* elements onto the start of the array.

This operation is slower than g_array_append_vals() since the existing elements in the array have to be moved to make space for the new elements.

array : a GArray.
data : a pointer to the elements to prepend to the start of the array.
len : the number of elements to prepend.
Returns : the GArray.

g_array_insert_val()

```
#define     g_array_insert_val(a,i,v)
```

Inserts an element into an array at the given index.

Note

`g_array_insert_val()` is a macro which uses a reference to the value parameter *v*. This means that you cannot use it with literal values such as "27". You must use variables.

a : a GArray.
i : the index to place the element at.
v : the value to insert into the array.
Returns : the GArray.

---

g_array_insert_vals ()

```
GArray*     g_array_insert_vals             (GArray *array,
                                             guint index_,
                                             gconstpointer data,
                                             guint len);
```

Inserts *len* elements into a GArray at the given index.

array : a GArray.
*index_* : the index to place the elements at.
data : a pointer to the elements to insert.
len : the number of elements to insert.
Returns : the GArray.

---

g_array_remove_index ()

```
GArray*     g_array_remove_index            (GArray *array,
                                             guint index_);
```

Removes the element at the given index from a GArray. The following elements are moved down one place.

array : a GArray.
*index_* : the index of the element to remove.
Returns : the GArray.

g_array_remove_index_fast ()

```
GArray*     g_array_remove_index_fast       (GArray *array,
                                             guint index_);
```

Removes the element at the given index from a GArray. The last element in the array is used to fill in the space, so this function does not preserve the order of the GArray. But it is faster than `g_array_remove_index()`.

array : a *GArray*.
*index_* : the index of the element to remove.
Returns : the GArray.

---

g_array_remove_range ()

```
GArray*     g_array_remove_range            (GArray *array,
                                             guint index_,
                                             guint length);
```

Removes the given number of elements starting at the given index from a GArray. The following elements are moved to close the gap.

array : a *GArray*.
*index_* : the index of the first element to remove.
length : the number of elements to remove.
Returns : the GArray.

Since 2.4

---

g_array_sort ()

```
void        g_array_sort                    (GArray *array,
                                             GCompareFunc compare_func);
```

Sorts a GArray using *compare_func* which should be a `qsort()`-style comparison function (returns -1 for first arg is less than second arg, 0 for equal, 1 if first arg is greater than second arg).

array : a GArray.
compare_func : comparison function.

---

g_array_sort_with_data ()

```
void        g_array_sort_with_data          (GArray *array,
                                             GCompareDataFunc compare_func,
                                             gpointer user_data);
```

Like g_array_sort(), but the comparison function receives a user data argument.

array : a GArray.
compare_func : comparison function.
user_data : data to pass to *compare_func*.

---

g_array_index()

```
#define     g_array_index(a,t,i)
```

Returns the element of a GArray at the given index. The return value is cast to the given type.

Example 4. Getting a pointer to an element in a GArray

```
  EDayViewEvent *event;

  /* This gets a pointer to the 3rd element in the array of EDayViewEvent
     structs. */
  event = &g_array_index (events, EDayViewEvent, 3);
```

a : a GArray.
t : the type of the elements.
i : the index of the element to return.
Returns : the element of the GArray at the index given by *i*.

---

g_array_set_size ()

```
GArray*     g_array_set_size                (GArray *array,
                                             guint length);
```

Sets the size of the array, expanding it if necessary. If the array was created with *clear_* set to TRUE, the new elements are set to 0.

array : a GArray.
length : the new size of the GArray.
Returns : the GArray.

---

g_array_free ()

```
gchar*      g_array_free                    (GArray *array,
```



```
                                             gboolean free_segment);
```

Frees the memory allocated for the GArray. If *free_segment* is TRUE it frees the actual element data as well.

array : a GArray.
free_segment : if TRUE the actual element data is freed as well.
Returns : the element data if *free_segment* is FALSE, otherwise NULL

<< String Chunks **Pointer Arrays >>**

Pointer Arrays

Pointer Arrays — arrays of pointers to any type of data, which grow automatically as new elements are added.

Synopsis

```
#include <glib.h>


            GPtrArray;
GPtrArray*  g_ptr_array_new                 (void);
GPtrArray*  g_ptr_array_sized_new           (guint reserved_size);
void        g_ptr_array_add                 (GPtrArray *array,
                                             gpointer data);
gboolean    g_ptr_array_remove              (GPtrArray *array,
                                             gpointer data);
gpointer    g_ptr_array_remove_index        (GPtrArray *array,
                                             guint index_);
gboolean    g_ptr_array_remove_fast         (GPtrArray *array,
                                             gpointer data);
gpointer    g_ptr_array_remove_index_fast   (GPtrArray *array,
                                             guint index_);
void        g_ptr_array_remove_range        (GPtrArray *array,
                                             guint index_,
                                             guint length);
void        g_ptr_array_sort                (GPtrArray *array,
                                             GCompareFunc compare_func);
void        g_ptr_array_sort_with_data      (GPtrArray *array,
                                             GCompareDataFunc compare_func,
                                             gpointer user_data);
void        g_ptr_array_set_size            (GPtrArray *array,
                                             gint length);
#define     g_ptr_array_index               (array,index_)
gpointer*   g_ptr_array_free                (GPtrArray *array,
                                             gboolean free_seg);
void        g_ptr_array_foreach             (GPtrArray *array,
                                             GFunc func,
                                             gpointer user_data);
```

Description

Pointer Arrays are similar to Arrays but are used only for storing pointers.

> **Note**
>
> If you remove elements from the array, elements at the end of the array are moved into the space previously occupied by the removed element. This means that you should not rely on the index of particular elements remaining the same. You should also be careful when deleting elements while iterating over the array.

To create a pointer array, use `g_ptr_array_new()`.

To add elements to a pointer array, use `g_ptr_array_add()`.

To remove elements from a pointer array, use `g_ptr_array_remove()`, `g_ptr_array_remove_index()` or `g_ptr_array_remove_index_fast()`.

To access an element of a pointer array, use `g_ptr_array_index()`.

To set the size of a pointer array, use `g_ptr_array_set_size()`.

To free a pointer array, use `g_ptr_array_free()`.

Example 5. Using a GPtrArray

```
  GPtrArray *gparray;
  gchar *string1 = "one", *string2 = "two", *string3 = "three";

  gparray = g_ptr_array_new ();
  g_ptr_array_add (gparray, (gpointer) string1);
  g_ptr_array_add (gparray, (gpointer) string2);
  g_ptr_array_add (gparray, (gpointer) string3);

  if (g_ptr_array_index (gparray, 0) != (gpointer) string1)
    g_print ("ERROR: got %p instead of %p\n",
             g_ptr_array_index (gparray, 0), string1);

  g_ptr_array_free (gparray, TRUE);
```

Details

GPtrArray

```
typedef struct {
  gpointer *pdata;
  guint     len;
} GPtrArray;
```

Contains the public fields of a pointer array.

gpointer *pdata; points to the array of pointers, which may be moved when the array grows.

guint len; number of pointers in the array.

---

g_ptr_array_new ()

```
GPtrArray*  g_ptr_array_new                 (void);
```

Creates a new GPtrArray.

Returns : the new GPtrArray.

---

g_ptr_array_sized_new ()

```
GPtrArray*  g_ptr_array_sized_new           (guint reserved_size);
```

Creates a new GPtrArray with *reserved_size* pointers preallocated. This avoids frequent reallocation, if you are going to add many pointers to the array. Note however that the size of the array is still 0.

reserved_size : number of pointers preallocated.
Returns : the new GPtrArray.

---

g_ptr_array_add ()

```
void        g_ptr_array_add                 (GPtrArray *array,
                                             gpointer data);
```

Adds a pointer to the end of the pointer array. The array will grow in size automatically if necessary.

array : a GPtrArray.
data : the pointer to add.

---

g_ptr_array_remove ()

```
gboolean    g_ptr_array_remove              (GPtrArray *array,
                                             gpointer data);
```

Removes the first occurrence of the given pointer from the pointer array. The following elements are moved down one place.

It returns TRUE if the pointer was removed, or FALSE if the pointer was not found.

array : a GPtrArray.
data : the pointer to remove.
Returns : TRUE if the pointer is removed. FALSE if the pointer is not found in the array.

---

g_ptr_array_remove_index ()

```
gpointer    g_ptr_array_remove_index        (GPtrArray *array,
                                             guint index_);
```

Removes the pointer at the given index from the pointer array. The following elements are moved

down one place.

array : a GPtrArray.
*index_* : the index of the pointer to remove.
Returns : the pointer which was removed.

---

g_ptr_array_remove_fast ()

```
gboolean    g_ptr_array_remove_fast         (GPtrArray *array,
                                             gpointer data);
```

Removes the first occurrence of the given pointer from the pointer array. The last element in the array is used to fill in the space, so this function does not preserve the order of the array. But it is faster than g_ptr_array_remove().

It returns TRUE if the pointer was removed, or FALSE if the pointer was not found.

array : a GPtrArray.
data : the pointer to remove.
Returns : TRUE if the pointer was found in the array.

---

g_ptr_array_remove_index_fast ()

```
gpointer    g_ptr_array_remove_index_fast   (GPtrArray *array,
                                             guint index_);
```

Removes the pointer at the given index from the pointer array. The last element in the array is used to fill in the space, so this function does not preserve the order of the array. But it is faster than g_ptr_array_remove_index().

array : a GPtrArray.
*index_* : the index of the pointer to remove.
Returns : the pointer which was removed.

---

g_ptr_array_remove_range ()

```
void        g_ptr_array_remove_range        (GPtrArray *array,
                                             guint index_,
                                             guint length);
```

Removes the given number of bytes starting at the given index from a GPtrArray. The following elements are moved to close the gap.

array : a *GPtrArray*.

*index_* : the index of the first pointer to remove.
length : the number of pointers to remove.

Since 2.4

---

g_ptr_array_sort ()

```
void        g_ptr_array_sort                (GPtrArray *array,
                                             GCompareFunc compare_func);
```

Sorts the array, using *compare_func* which should be a `qsort()`-style comparison function (returns -1 for first arg is less than second arg, 0 for equal, 1 if first arg is greater than second arg).

Note

The comparison function for `g_ptr_array_sort()` doesn't take the pointers from the array as arguments, it takes pointers to the pointers in the array.

array : a GPtrArray.
compare_func : comparison function.

---

g_ptr_array_sort_with_data ()

```
void        g_ptr_array_sort_with_data      (GPtrArray *array,
                                             GCompareDataFunc compare_func,
                                             gpointer user_data);
```

Like `g_ptr_array_sort()`, but the comparison function has a user data argument.

Note

The comparison function for `g_ptr_array_sort_with_data()` doesn't take the pointers from the array as arguments, it takes pointers to the pointers in the array.

array : a GPtrArray.
compare_func : comparison function.
user_data : data to pass to *compare_func*.

---

g_ptr_array_set_size ()

```
void        g_ptr_array_set_size            (GPtrArray *array,
                                             gint length);
```

Sets the size of the array, expanding it if necessary. New elements are set to `NULL`.

array : a GPtrArray.
length : the new length of the pointer array.

---

g_ptr_array_index()

```
#define     g_ptr_array_index(array,index_)
```

Returns the pointer at the given index of the pointer array.

array : a GPtrArray.
*index_* : the index of the pointer to return.
Returns : the pointer at the given index.

---

g_ptr_array_free ()

```
gpointer*   g_ptr_array_free                (GPtrArray *array,
                                             gboolean free_seg);
```

Frees all of the memory allocated for the pointer array.

array : a GPtrArray.
free_seg : if `TRUE` the actual element data is freed as well.
Returns :

---

g_ptr_array_foreach ()

```
void        g_ptr_array_foreach             (GPtrArray *array,
                                             GFunc func,
                                             gpointer user_data);
```

Calls a function for each element of a GPtrArray.

array : a GPtrArray
func : the function to call for each array element
user_data : user data to pass to the function

Since 2.4

<< Arrays **Byte Arrays >>**

Byte Arrays

Byte Arrays — arrays of bytes, which grow automatically as elements are added.

Synopsis

```
#include <glib.h>


            GByteArray;
GByteArray* g_byte_array_new                (void);
GByteArray* g_byte_array_sized_new          (guint reserved_size);
GByteArray* g_byte_array_append             (GByteArray *array,
                                             const guint8 *data,
                                             guint len);
GByteArray* g_byte_array_prepend            (GByteArray *array,
                                             const guint8 *data,
                                             guint len);
GByteArray* g_byte_array_remove_index       (GByteArray *array,
                                             guint index_);
GByteArray* g_byte_array_remove_index_fast  (GByteArray *array,
                                             guint index_);
GByteArray* g_byte_array_remove_range       (GByteArray *array,
                                             guint index_,
                                             guint length);
void        g_byte_array_sort               (GByteArray *array,
                                             GCompareFunc compare_func);
void        g_byte_array_sort_with_data     (GByteArray *array,
                                             GCompareDataFunc compare_func,
                                             gpointer user_data);
GByteArray* g_byte_array_set_size           (GByteArray *array,
                                             guint length);
guint8*     g_byte_array_free               (GByteArray *array,
                                             gboolean free_segment);
```

Description

GByteArray is based on GArray, to provide arrays of bytes which grow automatically as elements are added.

To create a new GByteArray use `g_byte_array_new()`.

To add elements to a GByteArray, use `g_byte_array_append()`, and `g_byte_array_prepend()`.

To set the size of a GByteArray, use `g_byte_array_set_size()`.

To free a GByteArray, use `g_byte_array_free()`.

Example 6. Using a GByteArray


```
GByteArray *gbarray;
gint i;

gbarray = g_byte_array_new ();
for (i = 0; i < 10000; i++)
  g_byte_array_append (gbarray, (guint8*) "abcd", 4);

for (i = 0; i < 10000; i++)
  {
    g_assert (gbarray->data[4*i] == 'a');
    g_assert (gbarray->data[4*i+1] == 'b');
    g_assert (gbarray->data[4*i+2] == 'c');
    g_assert (gbarray->data[4*i+3] == 'd');
  }

g_byte_array_free (gbarray, TRUE);
```

Details

GByteArray

```
typedef struct {
  guint8 *data;
  guint   len;
} GByteArray;
```

The GByteArray struct allows access to the public fields of a GByteArray.

guint8 *data; a pointer to the element data. The data may be moved as elements are added to the GByteArray.

guint len; the number of elements in the GByteArray.

---

g_byte_array_new ()

```
GByteArray* g_byte_array_new                (void);
```

Creates a new GByteArray.

Returns : the new GByteArray.

---

g_byte_array_sized_new ()

```
GByteArray* g_byte_array_sized_new          (guint reserved_size);
```

Creates a new GByteArray with *reserved_size* bytes preallocated. This avoids frequent reallocation, if you are going to add many bytes to the array. Note however that the size of the array is still 0.

reserved_size : number of bytes preallocated.
Returns : the new GByteArray.

---

g_byte_array_append ()

```
GByteArray* g_byte_array_append             (GByteArray *array,
                                             const guint8 *data,
                                             guint len);
```

Adds the given bytes to the end of the GByteArray. The array will grow in size automatically if necessary.

array : a GByteArray.
data : the byte data to be added.
len : the number of bytes to add.
Returns : the GByteArray.

---

g_byte_array_prepend ()

```
GByteArray* g_byte_array_prepend            (GByteArray *array,
                                             const guint8 *data,
                                             guint len);
```

Adds the given data to the start of the GByteArray. The array will grow in size automatically if necessary.

array : a GByteArray.
data : the byte data to be added.
len : the number of bytes to add.
Returns : the GByteArray.

---

g_byte_array_remove_index ()

```
GByteArray* g_byte_array_remove_index       (GByteArray *array,
                                             guint index_);
```

Removes the byte at the given index from a GByteArray. The following bytes are moved down one place.

array : a GByteArray.
*index_* : the index of the byte to remove.
Returns : the GByteArray.

---

g_byte_array_remove_index_fast ()

```
GByteArray* g_byte_array_remove_index_fast  (GByteArray *array,
                                             guint index_);
```

Removes the byte at the given index from a GByteArray. The last element in the array is used to fill in the space, so this function does not preserve the order of the GByteArray. But it is faster than `g_byte_array_remove_index()`.

array : a GByteArray.
*index_* : the index of the byte to remove.
Returns : the GByteArray.

---

g_byte_array_remove_range ()

```
GByteArray* g_byte_array_remove_range       (GByteArray *array,
                                             guint index_,
                                             guint length);
```

Removes the given number of bytes starting at the given index from a GByteArray. The following elements are moved to close the gap.

array : a *GByteArray*.
*index_* : the index of the first byte to remove.
length : the number of bytes to remove.
Returns : the GByteArray.

Since 2.4

---

g_byte_array_sort ()

```
void        g_byte_array_sort               (GByteArray *array,
                                             GCompareFunc compare_func);
```

Sorts a byte array, using *compare_func* which should be a `qsort()`-style comparison function (returns -1 for first arg is less than second arg, 0 for equal, 1 if first arg is greater than second arg).

array : a GByteArray.
compare_func : comparison function.

---

g_byte_array_sort_with_data ()

```
void        g_byte_array_sort_with_data     (GByteArray *array,
                                             GCompareDataFunc compare_func,
```

```
                                        gpointer user_data);
```

Like `g_byte_array_sort()`, but the comparison function takes a user data argument.

array : a GByteArray.
compare_func : comparison function.
user_data : data to pass to *compare_func*.

---

g_byte_array_set_size ()

```
GByteArray* g_byte_array_set_size           (GByteArray *array,
                                             guint length);
```

Sets the size of the GByteArray, expanding it if necessary.

array : a GByteArray.
length : the new size of the GByteArray.
Returns : the GByteArray.

---

g_byte_array_free ()

```
guint8*     g_byte_array_free               (GByteArray *array,
                                             gboolean free_segment);
```

Frees the memory allocated by the GByteArray. If *free_segment* is TRUE it frees the actual byte data.

array : a GByteArray.
free_segment : if TRUE the actual byte data is freed as well.
Returns :

<< Pointer Arrays **Balanced Binary Trees >>**

Balanced Binary Trees

Balanced Binary Trees — a sorted collection of key/value pairs optimized for searching and traversing in order.

Synopsis

```
#include <glib.h>


            GTree;
GTree*      g_tree_new                      (GCompareFunc key_compare_func);
GTree*      g_tree_new_with_data            (GCompareDataFunc key_compare_func,
                                             gpointer key_compare_data);
GTree*      g_tree_new_full                 (GCompareDataFunc key_compare_func,
                                             gpointer key_compare_data,
                                             GDestroyNotify key_destroy_func,
                                             GDestroyNotify value_destroy_func);
void        g_tree_insert                   (GTree *tree,
                                             gpointer key,
                                             gpointer value);
void        g_tree_replace                  (GTree *tree,
                                             gpointer key,
                                             gpointer value);
gint        g_tree_nnodes                   (GTree *tree);
gint        g_tree_height                   (GTree *tree);
gpointer    g_tree_lookup                   (GTree *tree,
                                             gconstpointer key);
gboolean    g_tree_lookup_extended          (GTree *tree,
                                             gconstpointer lookup_key,
                                             gpointer *orig_key,
                                             gpointer *value);
void        g_tree_foreach                  (GTree *tree,
                                             GTraverseFunc func,
                                             gpointer user_data);
void        g_tree_traverse                 (GTree *tree,
                                             GTraverseFunc traverse_func,
                                             GTraverseType traverse_type,
                                             gpointer user_data);
gboolean    (*GTraverseFunc)                (gpointer key,
                                             gpointer value,
                                             gpointer data);
enum        GTraverseType;
gpointer    g_tree_search                   (GTree *tree,
                                             GCompareFunc search_func,
                                             gconstpointer user_data);
void        g_tree_remove                   (GTree *tree,
                                             gconstpointer key);
void        g_tree_steal                    (GTree *tree,
                                             gconstpointer key);
void        g_tree_destroy                  (GTree *tree);
```

Description

The GTree structure and its associated functions provide a sorted collection of key/value pairs optimized for searching and traversing in order.

To create a new GTree use `g_tree_new()`.

To insert a key/value pair into a GTree use `g_tree_insert()`.

To lookup the value corresponding to a given key, use `g_tree_lookup()` and `g_tree_lookup_extended()`.

To find out the number of nodes in a GTree, use `g_tree_nnodes()`. To get the height of a GTree, use `g_tree_height()`.

To traverse a GTree, calling a function for each node visited in the traversal, use `g_tree_foreach()`.

To remove a key/value pair use `g_tree_remove()`.

To destroy a GTree, use `g_tree_destroy()`.

Details

GTree

```
typedef struct _GTree GTree;
```

The GTree struct is an opaque data structure representing a Balanced Binary Tree. It should be accessed only by using the following functions.

---

g_tree_new ()

```
GTree*      g_tree_new                      (GCompareFunc key_compare_func);
```

Creates a new GTree.

key_compare_func : the function used to order the nodes in the GTree. It should return values similar to the standard `strcmp()` function - 0 if the two arguments are equal, a negative value if the first argument comes before the second, or a positive value if the first argument comes after the second.

Returns : a new GTree.

---

g_tree_new_with_data ()

```
GTree*      g_tree_new_with_data            (GCompareDataFunc key_compare_func,
                                             gpointer key_compare_data);
```

Creates a new GTree with a comparison function that accepts user data. See `g_tree_new()` for more details.

key_compare_func : qsort()-style comparison function.
key_compare_data : data to pass to comparison function.
Returns : a new GTree.

---

g_tree_new_full ()

```
GTree*      g_tree_new_full                 (GCompareDataFunc key_compare_func,
                                             gpointer key_compare_data,
                                             GDestroyNotify key_destroy_func,
                                             GDestroyNotify value_destroy_func);
```

Creates a new GTree like `g_tree_new()` and allows to specify functions to free the memory allocated for the key and value that get called when removing the entry from the GTree.

key_compare_func : qsort()-style comparison function.
key_compare_data : data to pass to comparison function.
key_destroy_func : a function to free the memory allocated for the key used when removing the entry from the GTree or NULL if you don't want to supply such a function.
value_destroy_func : a function to free the memory allocated for the value used when removing the entry from the GTree or NULL if you don't want to supply such a function.
Returns : a new GTree.

---

g_tree_insert ()

```
void        g_tree_insert                   (GTree *tree,
                                             gpointer key,
                                             gpointer value);
```

Inserts a key/value pair into a GTree. If the given key already exists in the GTree its corresponding value is set to the new value. If you supplied a value_destroy_func when creating the GTree, the old value is freed using that function. If you supplied a *key_destroy_func* when creating the GTree, the passed key is freed using that function.

The tree is automatically 'balanced' as new key/value pairs are added, so that the distance from the root to every leaf is as small as possible.

tree : a GTree.
key : the key to insert.
value : the value corresponding to the key.

---

g_tree_replace ()

```
void        g_tree_replace                  (GTree *tree,
                                             gpointer key,
                                             gpointer value);
```

Inserts a new key and value into a GTree similar to `g_tree_insert()`. The difference is that if the key already exists in the GTree, it gets replaced by the new key. If you supplied a *value_destroy_func* when creating the GTree, the old value is freed using that function. If you supplied a *key_destroy_func* when creating the GTree, the old key is freed using that function.

The tree is automatically 'balanced' as new key/value pairs are added, so that the distance from the root to every leaf is as small as possible.

tree : a GTree.
key : the key to insert.
value : the value corresponding to the key.

---

g_tree_nnodes ()

```
gint        g_tree_nnodes                   (GTree *tree);
```

Gets the number of nodes in a GTree.

tree : a GTree.
Returns : the number of nodes in the GTree.

---

g_tree_height ()

```
gint        g_tree_height                   (GTree *tree);
```

Gets the height of a GTree.

If the GTree contains no nodes, the height is 0. If the GTree contains only one root node the height is 1. If the root node has children the height is 2, etc.

tree : a GTree.
Returns : the height of the GTree.

---

g_tree_lookup ()

```
gpointer    g_tree_lookup                   (GTree *tree,
                                             gconstpointer key);
```

Gets the value corresponding to the given key. Since a GTree is automatically balanced as key/value pairs are added, key lookup is very fast.

tree : a GTree.
key : the key to look up.
Returns : the value corresponding to the key.

---

g_tree_lookup_extended ()

```
gboolean    g_tree_lookup_extended          (GTree *tree,
                                             gconstpointer lookup_key,
                                             gpointer *orig_key,
                                             gpointer *value);
```

Looks up a key in the GTree, returning the original key and the associated value and a gboolean which is TRUE if the key was found. This is useful if you need to free the memory allocated for the original key, for example before calling g_tree_remove().

tree : a GTree.
lookup_key : the key to look up.
orig_key : returns the original key.
value : returns the value associated with the key.
Returns : TRUE if the key was found in the GTree.

---

g_tree_foreach ()

```
void        g_tree_foreach                  (GTree *tree,
                                             GTraverseFunc func,
                                             gpointer user_data);
```

Calls the given function for each of the key/value pairs in the GTree. The function is passed the key and value of each pair, and the given *data* parameter. The tree is traversed in sorted order.

The tree may not be modified while iterating over it (you can't add/remove items). To remove all items matching a predicate, you need to add each item to a list in your GTraverseFunc as you walk over the tree, then walk the list and remove each item.

tree : a GTree.
func : the function to call for each node visited. If this function returns TRUE, the traversal is stopped.
user_data : user data to pass to the function.

---

g_tree_traverse ()

```
void        g_tree_traverse                 (GTree *tree,
```



```
                                             GTraverseFunc traverse_func,
                                             GTraverseType traverse_type,
                                             gpointer user_data);
```

Warning

g_tree_traverse is deprecated and should not be used in newly-written code. The order of a balanced tree is somewhat arbitrary. If you just want to visit all nodes in sorted order, use g_tree_foreach() instead. If you really need to visit nodes in a different order, consider using an N-ary Tree.

Calls the given function for each node in the GTree.

tree : a GTree.
traverse_func : the function to call for each node visited. If this function returns TRUE, the traversal is stopped.
traverse_type : the order in which nodes are visited, one of G_IN_ORDER, G_PRE_ORDER and G_POST_ORDER.
user_data : user data to pass to the function.

GTraverseFunc ()

```
gboolean    (*GTraverseFunc)                (gpointer key,
                                             gpointer value,
                                             gpointer data);
```

Specifies the type of function passed to g_tree_traverse(). It is passed the key and value of each node, together with the *user_data* parameter passed to g_tree_traverse(). If the function returns TRUE, the traversal is stopped.

key : a key of a GTree node.
value : the value corresponding to the key.
data : user data passed to g_tree_traverse().
Returns : TRUE to stop the traversal.

---

enum GTraverseType

```
typedef enum
{
  G_IN_ORDER,
  G_PRE_ORDER,
  G_POST_ORDER,
  G_LEVEL_ORDER
} GTraverseType;
```

Specifies the type of traveral performed by g_tree_traverse(), g_node_traverse() and g_node_find().

G_IN_ORDER	vists a node's left child first, then the node itself, then its right child. This is the one to use if you want the output sorted according to the compare function.
G_PRE_ORDER	visits a node, then its children.
G_POST_ORDER	visits the node's children, then the node itself.
G_LEVEL_ORDER	is not implemented for Balanced Binary Trees. For N-ary Trees, it vists the root node first, then its children, then its grandchildren, and so on. Note that this is less efficient than the other orders.

---

g_tree_search ()

```
gpointer    g_tree_search                   (GTree *tree,
                                             GCompareFunc search_func,
                                             gconstpointer user_data);
```

Searches a GTree using *search_func*.

The *search_func* is called with a pointer to the key of a key/value pair in the tree, and the passed in *user_data*. If *search_func* returns 0 for a key/value pair, then g_tree_search_func() will return the value of that pair. If *search_func* returns -1, searching will proceed among the key/value pairs that have a smaller key; if *search_func* returns 1, searching will proceed among the key/value pairs that have a larger key.

tree :	a GTree.
search_func :	a function used to search the GTree.
user_data :	the data passed as the second argument to the *search_func* function.
Returns :	the value corresponding to the found key, or NULL if the key was not found.

---

g_tree_remove ()

```
void        g_tree_remove                   (GTree *tree,
                                             gconstpointer key);
```

Removes a key/value pair from a GTree.

If the GTree was created using g_tree_new_full(), the key and value are freed using the supplied destroy functions, otherwise you have to make sure that any dynamically allocated values are freed yourself.

tree : a GTree.
key : the key to remove.

---

g_tree_steal ()

```
void        g_tree_steal                    (GTree *tree,
```



```
                                             gconstpointer key);
```

Removes a key and its associated value from a GTree without calling the key and value destroy functions.

tree : a GTree.
key : the key to remove.

---

g_tree_destroy ()

```
void        g_tree_destroy                  (GTree *tree);
```

Destroys the GTree. If keys and/or values are dynamically allocated, you should either free them first or create the GTree using g_tree_new_full(). In the latter case the destroy functions you supplied will be called on all keys and values before destroying the GTree.

tree : a GTree.

<< Byte Arrays **N-ary Trees >>**

N-ary Trees

N-ary Trees — trees of data with any number of branches.

Synopsis

```
#include <glib.h>


            GNode;
GNode*      g_node_new                  (gpointer data);
GNode*      g_node_copy                 (GNode *node);
gpointer    (*GCopyFunc)                (gconstpointer src,
                                         gpointer data);
GNode*      g_node_copy_deep            (GNode *node,
                                         GCopyFunc copy_func,
                                         gpointer data);

GNode*      g_node_insert               (GNode *parent,
                                         gint position,
                                         GNode *node);
GNode*      g_node_insert_before        (GNode *parent,
                                         GNode *sibling,
                                         GNode *node);
GNode*      g_node_insert_after         (GNode *parent,
                                         GNode *sibling,
                                         GNode *node);
#define     g_node_append               (parent, node)
GNode*      g_node_prepend              (GNode *parent,
                                         GNode *node);

#define     g_node_insert_data          (parent, position, data)
#define     g_node_insert_data_before   (parent, sibling, data)
#define     g_node_append_data          (parent, data)
#define     g_node_prepend_data         (parent, data)

void        g_node_reverse_children     (GNode *node);
void        g_node_traverse             (GNode *root,
                                         GTraverseType order,
                                         GTraverseFlags flags,
                                         gint max_depth,
                                         GNodeTraverseFunc func,
                                         gpointer data);
enum        GTraverseFlags;
gboolean    (*GNodeTraverseFunc)        (GNode *node,
                                         gpointer data);
void        g_node_children_foreach     (GNode *node,
                                         GTraverseFlags flags,
                                         GNodeForeachFunc func,
                                         gpointer data);
void        (*GNodeForeachFunc)         (GNode *node,
                                         gpointer data);

GNode*      g_node_get_root             (GNode *node);
GNode*      g_node_find                 (GNode *root,
```



```
                                         GTraverseType order,
                                         GTraverseFlags flags,
                                         gpointer data);
GNode*      g_node_find_child           (GNode *node,
                                         GTraverseFlags flags,
                                         gpointer data);
gint        g_node_child_index          (GNode *node,
                                         gpointer data);
gint        g_node_child_position       (GNode *node,
                                         GNode *child);
#define     g_node_first_child          (node)
GNode*      g_node_last_child           (GNode *node);
GNode*      g_node_nth_child            (GNode *node,
                                         guint n);
GNode*      g_node_first_sibling        (GNode *node);
#define     g_node_next_sibling         (node)
#define     g_node_prev_sibling         (node)
GNode*      g_node_last_sibling         (GNode *node);

#define     G_NODE_IS_LEAF              (node)
#define     G_NODE_IS_ROOT              (node)
guint       g_node_depth                (GNode *node);
guint       g_node_n_nodes              (GNode *root,
                                         GTraverseFlags flags);
guint       g_node_n_children           (GNode *node);
gboolean    g_node_is_ancestor          (GNode *node,
                                         GNode *descendant);
guint       g_node_max_height           (GNode *root);

void        g_node_unlink               (GNode *node);
void        g_node_destroy              (GNode *root);

void        g_node_push_allocator       (GAllocator *allocator);
void        g_node_pop_allocator        (void);
```

Description

The GNode struct and its associated functions provide a N-ary tree data structure, where nodes in the tree can contain arbitrary data.

To create a new tree use g_node_new().

To insert a node into a tree use g_node_insert(), g_node_insert_before(), g_node_append() and g_node_prepend().

To create a new node and insert it into a tree use g_node_insert_data(), g_node_insert_data_before(), g_node_append_data() and g_node_prepend_data().

To reverse the children of a node use g_node_reverse_children().

To find a node use g_node_get_root(), g_node_find(), g_node_find_child(), g_node_child_index(), g_node_child_position(), g_node_first_child(), g_node_last_child(), g_node_nth_child(), g_node_first_sibling(), g_node_prev_sibling(), g_node_next_sibling() or g_node_last_sibling().

To get information about a node or tree use G_NODE_IS_LEAF(), G_NODE_IS_ROOT(), g_node_depth(), g_node_n_nodes(), g_node_n_children(), g_node_is_ancestor() or g_node_max_height().

To traverse a tree, calling a function for each node visited in the traversal, use `g_node_traverse()` or `g_node_children_foreach()`.

To remove a node or subtree from a tree use `g_node_unlink()` or `g_node_destroy()`.

Details

GNode

```
typedef struct {
  gpointer data;
  GNode   *next;
  GNode   *prev;
  GNode   *parent;
  GNode   *children;
} GNode;
```

The GNode struct represents one node in a N-ary Tree. The *data* field contains the actual data of the node. The *next* and *prev* fields point to the node's siblings (a sibling is another GNode with the same parent). The *parent* field points to the parent of the GNode, or is NULL if the GNode is the root of the tree. The *children* field points to the first child of the GNode. The other children are accessed by using the *next* pointer of each child.

---

g_node_new ()

```
GNode*      g_node_new                      (gpointer data);
```

Creates a new GNode containing the given data. Used to create the first node in a tree.

data : the data of the new node.
Returns : a new GNode.

---

g_node_copy ()

```
GNode*      g_node_copy                     (GNode *node);
```

Recursively copies a GNode (but does not deep-copy the data inside the nodes, see `g_node_copy_deep()` if you need that).

node : a GNode.
Returns : a new GNode containing the same data pointers.

GCopyFunc ()

```
gpointer    (*GCopyFunc)                    (gconstpointer src,
                                             gpointer data);
```

A function of this signature is used to copy the node data when doing a deep-copy of a tree.

src : A pointer to the data which should be copied.
data : Additional data.
Returns : A pointer to the copy.

Since 2.4

---

g_node_copy_deep ()

```
GNode*      g_node_copy_deep                (GNode *node,
                                             GCopyFunc copy_func,
                                             gpointer data);
```

Recursively copies a GNode and its data.

node : a GNode
copy_func : the function which is called to copy the data inside each node, or NULL to use the original data.
data : data to pass to *copy_func*
Returns : a new GNode containing copies of the data in *node*.

Since 2.4

---

g_node_insert ()

```
GNode*      g_node_insert                   (GNode *parent,
                                             gint position,
                                             GNode *node);
```

Inserts a GNode beneath the parent at the given position.

parent : the GNode to place *node* under.
position : the position to place *node* at, with respect to its siblings. If position is -1, *node* is inserted as the last child of *parent*.
node : the GNode to insert.
Returns : the inserted GNode.

---

g_node_insert_before ()

http://developer.gnome.org/doc/API/2.0/glib/glib-N-ary-Trees.html 16/11/2004

```
GNode*      g_node_insert_before            (GNode *parent,
                                             GNode *sibling,
                                             GNode *node);
```

Inserts a GNode beneath the parent before the given sibling.

parent : the GNode to place *node* under.
sibling : the sibling GNode to place *node* before. If sibling is NULL, the node is inserted as the last child of *parent*.
node : the GNode to insert.
Returns : the inserted GNode.

---

g_node_insert_after ()

```
GNode*      g_node_insert_after             (GNode *parent,
                                             GNode *sibling,
                                             GNode *node);
```

Inserts a GNode beneath the parent after the given sibling.

parent : the GNode to place *node* under.
sibling : the sibling GNode to place *node* after. If sibling is NULL, the node is inserted as the first child of *parent*.
node : the GNode to insert.
Returns : the inserted GNode.

---

g_node_append()

```
#define     g_node_append(parent, node)
```

Inserts a GNode as the last child of the given parent.

parent : the GNode to place the new GNode under.
node : the GNode to insert.
Returns : the inserted GNode.

---

g_node_prepend ()

```
GNode*      g_node_prepend                  (GNode *parent,
                                             GNode *node);
```

Inserts a GNode as the first child of the given parent.

parent : the GNode to place the new GNode under.
node : the GNode to insert.
Returns : the inserted GNode.

---

g_node_insert_data()

```
#define     g_node_insert_data(parent, position, data)
```

Inserts a new GNode at the given position.

parent : the GNode to place the new GNode under.
position : the position to place the new GNode at. If position is -1, the new GNode is inserted as the last child of *parent*.
data : the data for the new GNode.
Returns : the new GNode.

---

g_node_insert_data_before()

```
#define     g_node_insert_data_before(parent, sibling, data)
```

Inserts a new GNode before the given sibling.

parent : the GNode to place the new GNode under.
sibling : the sibling GNode to place the new GNode before.
data : the data for the new GNode.
Returns : the new GNode.

---

g_node_append_data()

```
#define     g_node_append_data(parent, data)
```

Inserts a new GNode as the last child of the given parent.

parent : the GNode to place the new GNode under.
data : the data for the new GNode.
Returns : the new GNode.

---

g_node_prepend_data()

```
#define     g_node_prepend_data(parent, data)
```

Inserts a new GNode as the first child of the given parent.

parent : the GNode to place the new GNode under.
data : the data for the new GNode.
Returns : the new GNode.

---

g_node_reverse_children ()

```
void        g_node_reverse_children         (GNode *node);
```

Reverses the order of the children of a GNode. (It doesn't change the order of the grandchildren.)

node : a GNode.

---

g_node_traverse ()

```
void        g_node_traverse                 (GNode *root,
                                             GTraverseType order,
                                             GTraverseFlags flags,
                                             gint max_depth,
                                             GNodeTraverseFunc func,
                                             gpointer data);
```

Traverses a tree starting at the given root GNode. It calls the given function for each node visited. The traversal can be halted at any point by returning TRUE from *func*.

root : the root GNode of the tree to traverse.
order : the order in which nodes are visited - G_IN_ORDER, G_PRE_ORDER, G_POST_ORDER, or G_LEVEL_ORDER.
flags : which types of children are to be visited, one of G_TRAVERSE_ALL, G_TRAVERSE_LEAVES and G_TRAVERSE_NON_LEAVES.
max_depth : the maximum depth of the traversal. Nodes below this depth will not be visited. If max_depth is -1 all nodes in the tree are visited. If depth is 1, only the root is visited. If depth is 2, the root and its children are visited. And so on.
func : the function to call for each visited GNode.
data : user data to pass to the function.

---

enum GTraverseFlags

```
typedef enum
{
  G_TRAVERSE_LEAVES     = 1 << 0,
  G_TRAVERSE_NON_LEAVES = 1 << 1,
  G_TRAVERSE_ALL        = G_TRAVERSE_LEAVES | G_TRAVERSE_NON_LEAVES,
  G_TRAVERSE_MASK       = 0x03,
  G_TRAVERSE_LEAFS      = G_TRAVERSE_LEAVES,
  G_TRAVERSE_NON_LEAFS  = G_TRAVERSE_NON_LEAVES
} GTraverseFlags;
```


Specifies which nodes are visited during several of the tree functions, including g_node_traverse() and g_node_find().

G_TRAVERSE_LEAVES — only leaf nodes should be visited. This name has been introduced in 2.6, for older version use G_TRAVERSE_LEAFS.
G_TRAVERSE_NON_LEAVES — only non-leaf nodes should be visited. This name has been introduced in 2.6, for older version use G_TRAVERSE_NON_LEAFS.
G_TRAVERSE_ALL — all nodes should be visited.
G_TRAVERSE_MASK
G_TRAVERSE_LEAFS — identical to G_TRAVERSE_LEAVES
G_TRAVERSE_NON_LEAFS — identical to G_TRAVERSE_NON_LEAVES

GNodeTraverseFunc ()

```
gboolean    (*GNodeTraverseFunc)            (GNode *node,
                                             gpointer data);
```

Specifies the type of function passed to g_node_traverse(). The function is called with each of the nodes visited, together with the user data passed to g_node_traverse(). If the function returns TRUE, then the traversal is stopped.

node : a GNode.
data : user data passed to g_node_traverse().
Returns : TRUE to stop the traversal.

---

g_node_children_foreach ()

```
void        g_node_children_foreach         (GNode *node,
                                             GTraverseFlags flags,
                                             GNodeForeachFunc func,
                                             gpointer data);
```

Calls a function for each of the children of a GNode. Note that it doesn't descend beneath the child nodes.

node : a GNode.
flags : which types of children are to be visited, one of G_TRAVERSE_ALL, G_TRAVERSE_LEAVES and G_TRAVERSE_NON_LEAVES.
func : the function to call for each visited node.
data : user data to pass to the function.

GNodeForeachFunc ()

```
void        (*GNodeForeachFunc)             (GNode *node,
                                             gpointer data);
```

Specifies the type of function passed to g_node_children_foreach(). The function is called with each child node, together with the user data passed to g_node_children_foreach().

node : a GNode.
data : user data passed to g_node_children_foreach().

---

g_node_get_root ()

```
GNode*      g_node_get_root                 (GNode *node);
```

Gets the root of a tree.

node : a GNode.
Returns : the root of the tree.

---

g_node_find ()

```
GNode*      g_node_find                     (GNode *root,
                                             GTraverseType order,
                                             GTraverseFlags flags,
                                             gpointer data);
```

Finds a GNode in a tree.

root : the root GNode of the tree to search.
order : the order in which nodes are visited - G_IN_ORDER, G_PRE_ORDER, G_POST_ORDER, or G_LEVEL_ORDER.
flags : which types of children are to be searched, one of G_TRAVERSE_ALL, G_TRAVERSE_LEAVES and G_TRAVERSE_NON_LEAVES.
data : the data to find.
Returns : the found GNode, or NULL if the data is not found.

---

g_node_find_child ()

```
GNode*      g_node_find_child               (GNode *node,
                                             GTraverseFlags flags,
                                             gpointer data);
```

Finds the first child of a GNode with the given data.

node : a GNode.
flags : which types of children are to be searched, one of G_TRAVERSE_ALL, G_TRAVERSE_LEAVES and G_TRAVERSE_NON_LEAVES.
data : the data to find.
Returns : the found child GNode, or NULL if the data is not found.

---

g_node_child_index ()

```
gint        g_node_child_index              (GNode *node,
                                             gpointer data);
```

Gets the position of the first child of a GNode which contains the given data.

node : a GNode.
data : the data to find.
Returns : the index of the child of *node* which contains *data*, or -1 if the data is not found.

---

g_node_child_position ()

```
gint        g_node_child_position           (GNode *node,
                                             GNode *child);
```

Gets the position of a GNode with respect to its siblings. *child* must be a child of *node*. The first child is numbered 0, the second 1, and so on.

node : a GNode.
child : a child of *node*.
Returns : the position of *child* with respect to its siblings.

---

g_node_first_child()

```
#define     g_node_first_child(node)
```

Gets the first child of a GNode.

node : a GNode.
Returns : the last child of *node*, or NULL if *node* is NULL or has no children.

---

g_node_last_child ()

```
GNode*      g_node_last_child               (GNode *node);
```

Gets the last child of a GNode.

node : a GNode (must not be NULL).
Returns : the last child of *node*, or NULL if *node* has no children.

---

g_node_nth_child ()

```
GNode*      g_node_nth_child                (GNode *node,
                                             guint n);
```

Gets a child of a GNode, using the given index. The first child is at index 0. If the index is too big, NULL is returned.

node : a GNode.
n : the index of the desired child.
Returns : the child of *node* at index *n*.

---

g_node_first_sibling ()

```
GNode*      g_node_first_sibling            (GNode *node);
```

Gets the first sibling of a GNode. This could possibly be the node itself.

node : a GNode.
Returns : the first sibling of *node*.

---

g_node_next_sibling()

```
#define     g_node_next_sibling(node)
```

Gets the next sibling of a GNode.

node : a GNode.
Returns : the next sibling of *node*, or NULL if *node* is NULL.

---

g_node_prev_sibling()

```
#define     g_node_prev_sibling(node)
```

Gets the previous sibling of a GNode.

a GNode.

node :
Returns : the previous sibling of *node*, or NULL if *node* is NULL.

---

g_node_last_sibling ()

```
GNode*      g_node_last_sibling             (GNode *node);
```

Gets the last sibling of a GNode. This could possibly be the node itself.

node : a GNode.
Returns : the last sibling of *node*.

G_NODE_IS_LEAF()

```
#define  G_NODE_IS_LEAF(node)   (((GNode*) (node))->children == NULL)
```

Returns TRUE if a GNode is a leaf node.

node : a GNode.
Returns : TRUE if the GNode is a leaf node (i.e. it has no children).

G_NODE_IS_ROOT()

```
#define     G_NODE_IS_ROOT(node)
```

Returns TRUE if a GNode is the root of a tree.

node : a GNode.
Returns : TRUE if the GNode is the root of a tree (i.e. it has no parent or siblings).

---

g_node_depth ()

```
guint       g_node_depth                    (GNode *node);
```

Gets the depth of a GNode.

If *node* is NULL the depth is 0. The root node has a depth of 1. For the children of the root node the depth is 2. And so on.

node : a GNode.
Returns : the depth of the GNode.

g_node_n_nodes ()

```
guint       g_node_n_nodes                  (GNode *root,
                                             GTraverseFlags flags);
```

Gets the number of nodes in a tree.

root : a GNode.
flags : which types of children are to be counted, one of G_TRAVERSE_ALL, G_TRAVERSE_LEAVES and G_TRAVERSE_NON_LEAVES.
Returns : the number of nodes in the tree.

g_node_n_children ()

```
guint       g_node_n_children               (GNode *node);
```

Gets the number of children of a GNode.

node : a GNode.
Returns : the number of children of *node*.

g_node_is_ancestor ()

```
gboolean    g_node_is_ancestor              (GNode *node,
                                             GNode *descendant);
```

Returns TRUE if *node* is an ancestor of *descendant*. This is true if node is the parent of *descendant*, or if node is the grandparent of *descendant* etc.

node : a GNode.
descendant : a GNode.
Returns : TRUE if *node* is an ancestor of *descendant*.

g_node_max_height ()

```
guint       g_node_max_height               (GNode *root);
```

Gets the maximum height of all branches beneath a GNode. This is the maximum distance from the GNode to all leaf nodes.

If *root* is NULL, 0 is returned. If *root* has no children, 1 is returned. If *root* has children, 2 is returned. And so on.

root : a GNode.
Returns : the maximum height of the tree beneath *root*.

g_node_unlink ()

```
void        g_node_unlink                   (GNode *node);
```

Unlinks a GNode from a tree, resulting in two separate trees.

node : the GNode to unlink, which becomes the root of a new tree.

g_node_destroy ()

```
void        g_node_destroy                  (GNode *root);
```

Removes the GNode and its children from the tree, freeing any memory allocated.

root : the root of the tree/subtree to destroy.

g_node_push_allocator ()

```
void        g_node_push_allocator           (GAllocator *allocator);
```

Sets the allocator to use to allocate GNode elements. Use g_node_pop_allocator() to restore the previous allocator.

allocator : the GAllocator to use when allocating GNode elements.

g_node_pop_allocator ()

```
void        g_node_pop_allocator            (void);
```

Restores the previous GAllocator, used when allocating GNode elements.

<< Balanced Binary Trees **Quarks >>**

GLib Reference Manual

Quarks

Quarks — a 2-way association between a string and a unique integer identifier.

Synopsis

```
#include <glib.h>


typedef     GQuark;
GQuark      g_quark_from_string             (const gchar *string);
GQuark      g_quark_from_static_string      (const gchar *string);
G_CONST_RETURN gchar* g_quark_to_string     (GQuark quark);
GQuark      g_quark_try_string              (const gchar *string);
```

Description

Quarks are associations between strings and integer identifiers. Given either the string or the GQuark identifier it is possible to retrieve the other.

Quarks are used for both Datasets and Keyed Data Lists.

To create a new quark from a string, use `g_quark_from_string()` or `g_quark_from_static_string()`.

To find the string corresponding to a given GQuark, use `g_quark_to_string()`.

To find the GQuark corresponding to a given string, use `g_quark_try_string()`.

Details

GQuark

```
typedef guint32 GQuark;
```

A GQuark is an integer which uniquely identifies a particular string.

---

g_quark_from_string ()

```
GQuark      g_quark_from_string             (const gchar *string);
```

Gets the GQuark identifying the given string. If the string does not currently have an associated GQuark, a new GQuark is created, using a copy of the string.

string : a string.

Returns : the GQuark identifying the string.

---

g_quark_from_static_string ()

```
GQuark      g_quark_from_static_string      (const gchar *string);
```

Gets the GQuark identifying the given (static) string. If the string does not currently have an associated GQuark, a new GQuark is created, linked to the given string.

Note that this function is identical to `g_quark_from_string()` except that if a new GQuark is created the string itself is used rather than a copy. This saves memory, but can only be used if the string will always exist (if, for example, it is a statically-allocated string).

string : a string.

Returns : the GQuark identifying the string.

---

g_quark_to_string ()

```
G_CONST_RETURN gchar* g_quark_to_string     (GQuark quark);
```

Gets the string associated with the given GQuark.

quark : a GQuark.

Returns : the string associated with the GQuark.

---

g_quark_try_string ()

```
GQuark      g_quark_try_string              (const gchar *string);
```

Gets the GQuark associated with the given string, or 0 if the string has no associated GQuark.

If you want the GQuark to be created if it doesn't already exist, use `g_quark_from_string()` or `g_quark_from_static_string()`.

string : a string.

Returns : the GQuark associated with the string, or 0 if there is no GQuark associated with the string.

<< N-ary Trees **Keyed Data Lists >>**

Keyed Data Lists

Keyed Data Lists — lists of data elements which are accessible by a string or GQuark identifier.

Synopsis

```
#include <glib.h>


            GData;
void        g_datalist_init                 (GData **datalist);

#define     g_datalist_id_set_data          (dl, q, d)
void        g_datalist_id_set_data_full     (GData **datalist,
                                             GQuark key_id,
                                             gpointer data,
                                             GDestroyNotify destroy_func);
gpointer    g_datalist_id_get_data          (GData **datalist,
                                             GQuark key_id);
#define     g_datalist_id_remove_data       (dl, q)
gpointer    g_datalist_id_remove_no_notify  (GData **datalist,
                                             GQuark key_id);

#define     g_datalist_set_data             (dl, k, d)
#define     g_datalist_set_data_full        (dl, k, d, f)
#define     g_datalist_get_data             (dl, k)
#define     g_datalist_remove_data          (dl, k)
#define     g_datalist_remove_no_notify     (dl, k)

void        g_datalist_foreach              (GData **datalist,
                                             GDataForeachFunc func,
                                             gpointer user_data);
void        g_datalist_clear                (GData **datalist);
```

Description

Keyed data lists provide lists of arbitrary data elements which can be accessed either with a string or with a GQuark corresponding to the string.

The GQuark methods are quicker, since the strings have to be converted to GQuarks anyway.

Data lists are used for associating arbitrary data with GObjects, using `g_object_set_data()` and related functions.

To create a datalist, use `g_datalist_init()`.

To add data elements to a datalist use `g_datalist_id_set_data()`, `g_datalist_id_set_data_full()`, `g_datalist_set_data()` and `g_datalist_set_data_full()`.

To get data elements from a datalist use `g_datalist_id_get_data()` and `g_datalist_get_data()`.

To iterate over all data elements in a datalist use `g_datalist_foreach()`.

To remove data elements from a datalist use `g_datalist_id_remove_data()` and `g_datalist_remove_data()`.

To remove all data elements from a datalist, use `g_datalist_clear()`.

Details

GData

```
typedef struct _GData GData;
```

The GData struct is an opaque data structure to represent a Keyed Data List. It should only be accessed via the following functions.

---

g_datalist_init ()

```
void        g_datalist_init                 (GData **datalist);
```

Resets the datalist to `NULL`. It does not free any memory or call any destroy functions.

`datalist` : a pointer to a pointer to a datalist.

---

g_datalist_id_set_data()

```
#define     g_datalist_id_set_data(dl, q, d)
```

Sets the data corresponding to the given GQuark id. Any previous data with the same key is removed, and its destroy function is called.

`dl` : a datalist.
`q` : the GQuark to identify the data element.
`d` : the data element, or `NULL` to remove any previous element corresponding to *`q`*.

---

g_datalist_id_set_data_full ()

```
void        g_datalist_id_set_data_full     (GData **datalist,
                                             GQuark key_id,
                                             gpointer data,
                                             GDestroyNotify destroy_func);
```

Sets the data corresponding to the given GQuark id, and the function to be called when the element is removed from the datalist. Any previous data with the same key is removed, and its destroy function is called.

datalist : a datalist.
key_id : the GQuark to identify the data element.
data : the data element or NULL to remove any previous element corresponding to *key_id*.
destroy_func : the function to call when the data element is removed. This function will be called with the data element and can be used to free any memory allocated for it. If *data* is NULL, then *destroy_func* must also be NULL.

---

g_datalist_id_get_data ()

```
gpointer    g_datalist_id_get_data          (GData **datalist,
                                             GQuark key_id);
```

Retrieves the data element corresponding to *key_id*.

datalist : a datalist.
key_id : the GQuark identifying a data element.
Returns : the data element, or NULL if it is not found.

---

g_datalist_id_remove_data()

```
#define     g_datalist_id_remove_data(dl, q)
```

Removes an element, using its GQuark identifier.

dl : a datalist.
q : the GQuark identifying the data element.

---

g_datalist_id_remove_no_notify ()

```
gpointer    g_datalist_id_remove_no_notify  (GData **datalist,
                                             GQuark key_id);
```

Removes an element, without calling its destroy notification function.

datalist : a datalist.
key_id : the GQuark identifying a data element.
Returns : the data previously stored at *key_id*, or NULL if none.

---

g_datalist_set_data()

```
#define     g_datalist_set_data(dl, k, d)
```

Sets the data element corresponding to the given string identifier.

dl : a datalist.
k : the string to identify the data element.
d : the data element, or NULL to remove any previous element corresponding to *k*.

---

g_datalist_set_data_full()

```
#define     g_datalist_set_data_full(dl, k, d, f)
```

Sets the data element corresponding to the given string identifier, and the function to be called when the data element is removed.

dl : a datalist.
k : the string to identify the data element.
d : the data element, or NULL to remove any previous element corresponding to *k*.
f : the function to call when the data element is removed. This function will be called with the data element and can be used to free any memory allocated for it. If *d* is NULL, then *f* must also be NULL.

---

g_datalist_get_data()

```
#define     g_datalist_get_data(dl, k)
```

Gets a data element, using its string identifer. This is slower than g_datalist_id_get_data() because the string is first converted to a GQuark.

dl : a datalist.
k : the string identifying a data element.
Returns : the data element, or NULL if it is not found.

---

g_datalist_remove_data()

```
#define     g_datalist_remove_data(dl, k)
```

Removes an element using its string identifier. The data element's destroy function is called if it has been set.

dl : a datalist.

k : the string identifying the data element.

---

g_datalist_remove_no_notify()

```
#define     g_datalist_remove_no_notify(dl, k)
```

Removes an element, without calling its destroy notifier.

dl : a datalist.
k : the string identifying the data element.

---

g_datalist_foreach ()

```
void        g_datalist_foreach              (GData **datalist,
                                             GDataForeachFunc func,
                                             gpointer user_data);
```

Calls the given function for each data element of the datalist. The function is called with each data element's GQuark id and data, together with the given *user_data* parameter.

datalist : a datalist.
func : the function to call for each data element.
user_data : user data to pass to the function.

---

g_datalist_clear ()

```
void        g_datalist_clear                (GData **datalist);
```

Frees all the data elements of the datalist. The data elements' destroy functions are called if they have been set.

datalist : a datalist.

<< Quarks **Datasets >>**

Datasets

Datasets — associate groups of data elements with particular memory locations.

Synopsis

```
#include <glib.h>


#define     g_dataset_id_set_data           (l, k, d)
void        g_dataset_id_set_data_full      (gconstpointer dataset_location,
                                             GQuark key_id,
                                             gpointer data,
                                             GDestroyNotify destroy_func);
void        (*GDestroyNotify)               (gpointer data);
gpointer    g_dataset_id_get_data           (gconstpointer dataset_location,
                                             GQuark key_id);
#define     g_dataset_id_remove_data        (l, k)
gpointer    g_dataset_id_remove_no_notify   (gconstpointer dataset_location,
                                             GQuark key_id);

#define     g_dataset_set_data              (l, k, d)
#define     g_dataset_set_data_full         (l, k, d, f)
#define     g_dataset_get_data              (l, k)
#define     g_dataset_remove_data           (l, k)
#define     g_dataset_remove_no_notify      (l, k)

void        g_dataset_foreach               (gconstpointer dataset_location,
                                             GDataForeachFunc func,
                                             gpointer user_data);
void        (*GDataForeachFunc)             (GQuark key_id,
                                             gpointer data,
                                             gpointer user_data);
void        g_dataset_destroy               (gconstpointer dataset_location);
```

Description

Datasets associate groups of data elements with particular memory locations. These are useful if you need to associate data with a structure returned from an external library. Since you cannot modify the structure, you use its location in memory as the key into a dataset, where you can associate any number of data elements with it.

There are two forms of most of the dataset functions. The first form uses strings to identify the data elements associated with a location. The second form uses GQuark identifiers, which are created with a call to `g_quark_from_string()` or `g_quark_from_static_string()`. The second form is quicker, since it does not require looking up the string in the hash table of GQuark identifiers.

There is no function to create a dataset. It is automatically created as soon as you add elements to it.

To add data elements to a dataset use `g_dataset_id_set_data()`, `g_dataset_id_set_data_full`

`()`, `g_dataset_set_data()` and `g_dataset_set_data_full()`.

To get data elements from a dataset use `g_dataset_id_get_data()` and `g_dataset_get_data()`.

To iterate over all data elements in a dataset use `g_dataset_foreach()`.

To remove data elements from a dataset use `g_dataset_id_remove_data()` and `g_dataset_remove_data()`.

To destroy a dataset, use `g_dataset_destroy()`.

Details

g_dataset_id_set_data()

```
#define     g_dataset_id_set_data(l, k, d)
```

Sets the data element associated with the given GQuark id. Any previous data with the same key is removed, and its destroy function is called.

l : the location identifying the dataset.
k : the GQuark id to identify the data element.
d : the data element.

---

g_dataset_id_set_data_full ()

```
void        g_dataset_id_set_data_full      (gconstpointer dataset_location,
                                             GQuark key_id,
                                             gpointer data,
                                             GDestroyNotify destroy_func);
```

Sets the data element associated with the given GQuark id, and also the function to call when the data element is destroyed. Any previous data with the same key is removed, and its destroy function is called.

dataset_location : the location identifying the dataset.
key_id : the GQuark id to identify the data element.
data : the data element.
destroy_func : the function to call when the data element is removed. This function will be called with the data element and can be used to free any memory allocated for it.

GDestroyNotify ()

```
void        (*GDestroyNotify)               (gpointer data);
```

Specifies the type of function which is called when a data element is destroyed. It is passed the pointer to the data element and should free any memory and resources allocated for it.

data : the data element.

---

g_dataset_id_get_data ()

```
gpointer    g_dataset_id_get_data           (gconstpointer dataset_location,
                                             GQuark key_id);
```

Gets the data element corresponding to a GQuark.

dataset_location : the location identifying the dataset.
key_id : the GQuark id to identify the data element.
Returns : the data element corresponding to the GQuark, or NULL if it is not found.

---

g_dataset_id_remove_data()

```
#define     g_dataset_id_remove_data(l, k)
```

Removes a data element from a dataset. The data element's destroy function is called if it has been set.

l : the location identifying the dataset.
k : the GQuark id identifying the data element.

---

g_dataset_id_remove_no_notify ()

```
gpointer    g_dataset_id_remove_no_notify   (gconstpointer dataset_location,
                                             GQuark key_id);
```

Removes an element, without calling its destroy notification function.

dataset_location : the location identifying the dataset.
key_id : the GQuark ID identifying the data element.
Returns : the data previously stored at *key_id*, or NULL if none.

---

g_dataset_set_data()

```
#define     g_dataset_set_data(l, k, d)
```


Sets the data corresponding to the given string identifier.

l : the location identifying the dataset.
k : the string to identify the data element.
d : the data element.

---

g_dataset_set_data_full()

```
#define     g_dataset_set_data_full(l, k, d, f)
```

Sets the data corresponding to the given string identifier, and the function to call when the data element is destroyed.

l : the location identifying the dataset.
k : the string to identify the data element.
d : the data element.
f : the function to call when the data element is removed. This function will be called with the data element and can be used to free any memory allocated for it.

---

g_dataset_get_data()

```
#define     g_dataset_get_data(l, k)
```

Gets the data element corresponding to a string.

l : the location identifying the dataset.
k : the string identifying the data element.
Returns : the data element corresponding to the string, or NULL if it is not found.

---

g_dataset_remove_data()

```
#define     g_dataset_remove_data(l, k)
```

Removes a data element corresponding to a string. Its destroy function is called if it has been set.

l : the location identifying the dataset.
k : the string identifying the data element.

---

g_dataset_remove_no_notify()

```
#define     g_dataset_remove_no_notify(l, k)
```

Removes an element, without calling its destroy notifier.

l : the location identifying the dataset.
k : the string identifying the data element.

---

g_dataset_foreach ()

```
void        g_dataset_foreach               (gconstpointer dataset_location,
                                             GDataForeachFunc func,
                                             gpointer user_data);
```

Calls the given function for each data element which is associated with the given location.

dataset_location : the location identifying the dataset.
func : the function to call for each data element.
user_data : user data to pass to the function.

GDataForeachFunc ()

```
void        (*GDataForeachFunc)             (GQuark key_id,
                                             gpointer data,
                                             gpointer user_data);
```

Specifies the type of function passed to g_dataset_foreach(). It is called with each GQuark id and associated data element, together with the *user_data* parameter supplied to g_dataset_foreach().

key_id : the GQuark id to identifying the data element.
data : the data element.
user_data : user data passed to g_dataset_foreach().

---

g_dataset_destroy ()

```
void        g_dataset_destroy               (gconstpointer dataset_location);
```

Destroys the dataset, freeing all memory allocated, and calling any destroy functions set for data elements.

dataset_location : the location identifying the dataset.

<< Keyed Data Lists **Relations and Tuples >>**

Relations and Tuples

Relations and Tuples — tables of data which can be indexed on any number of fields.

Synopsis

```
#include <glib.h>


            GRelation;
GRelation*  g_relation_new                  (gint fields);
void        g_relation_index                (GRelation *relation,
                                             gint field,
                                             GHashFunc hash_func,
                                             GEqualFunc key_equal_func);
void        g_relation_insert               (GRelation *relation,
                                             ...);
gboolean    g_relation_exists               (GRelation *relation,
                                             ...);
gint        g_relation_count                (GRelation *relation,
                                             gconstpointer key,
                                             gint field);
GTuples*    g_relation_select               (GRelation *relation,
                                             gconstpointer key,
                                             gint field);
gint        g_relation_delete               (GRelation *relation,
                                             gconstpointer key,
                                             gint field);
void        g_relation_destroy              (GRelation *relation);

void        g_relation_print                (GRelation *relation);

            GTuples;
void        g_tuples_destroy                (GTuples *tuples);
gpointer    g_tuples_index                  (GTuples *tuples,
                                             gint index_,
                                             gint field);
```

Description

A GRelation is a table of data which can be indexed on any number of fields, rather like simple database tables. A GRelation contains a number of records, called tuples. Each record contains a number of fields. Records are not ordered, so it is not possible to find the record at a particular index.

Note that GRelation tables are currently limited to 2 fields.

To create a GRelation, use `g_relation_new()`.

To specify which fields should be indexed, use `g_relation_index()`. Note that this must be called before any tuples are added to the GRelation.

To add records to a GRelation use `g_relation_insert()`.

To determine if a given record appears in a GRelation, use `g_relation_exists()`. Note that fields are compared directly, so pointers must point to the exact same position (i.e. different copies of the same string will not match.)

To count the number of records which have a particular value in a given field, use `g_relation_count()`.

To get all the records which have a particular value in a given field, use `g_relation_select()`. To access fields of the resulting records, use `g_tuples_index()`. To free the resulting records use `g_tuples_destroy()`.

To delete all records which have a particular value in a given field, use `g_relation_delete()`.

To destroy the GRelation, use `g_relation_destroy()`.

To help debug GRelation objects, use `g_relation_print()`.

Details

GRelation

```
typedef struct _GRelation GRelation;
```

The GRelation struct is an opaque data structure to represent a Relation. It should only be accessed via the following functions.

---

g_relation_new ()

```
GRelation*  g_relation_new                  (gint fields);
```

Creates a new GRelation with the given number of fields. Note that currently the number of fields must be 2.

`fields` : the number of fields.

Returns : a new GRelation.

---

g_relation_index ()

```
void        g_relation_index                (GRelation *relation,
                                             gint field,
                                             GHashFunc hash_func,
                                             GEqualFunc key_equal_func);
```

Creates an index on the given field. Note that this must be called before any records are added to the GRelation.

relation : a GRelation.
field : the field to index, counting from 0.
hash_func : a function to produce a hash value from the field data.
key_equal_func : a function to compare two values of the given field.

---

g_relation_insert ()

```
void        g_relation_insert               (GRelation *relation,
                                             ...);
```

Inserts a record into a GRelation.

relation : a GRelation.
... : the fields of the record to add. This must match the number of fields in the GRelation.

---

g_relation_exists ()

```
gboolean    g_relation_exists               (GRelation *relation,
                                             ...);
```

Returns TRUE if a record with the given values exists in a GRelation. Note that the values are compared directly, so that, for example, two copies of the same string will not match.

relation : a GRelation.
... : the fields of the record to compare. The number must match the number of fields in the GRelation.
Returns : TRUE if a record matches.

---

g_relation_count ()

```
gint        g_relation_count                (GRelation *relation,
                                             gconstpointer key,
                                             gint field);
```

Returns the number of tuples in a GRelation that have the given value in the given field.

relation : a GRelation.
key : the value to compare with.
field : the field of each record to match.
Returns : the number of matches.

---

g_relation_select ()

```
GTuples*    g_relation_select               (GRelation *relation,
                                             gconstpointer key,
                                             gint field);
```

Returns all of the tuples which have the given key in the given field. Use g_tuples_index() to access the returned records. The returned records should be freed with g_tuples_destroy().

relation : a GRelation.
key : the value to compare with.
field : the field of each record to match.
Returns : the records (tuples) that matched.

---

g_relation_delete ()

```
gint        g_relation_delete               (GRelation *relation,
                                             gconstpointer key,
                                             gint field);
```

Deletes any records from a GRelation that have the given key value in the given field.

relation : a GRelation.
key : the value to compare with.
field : the field of each record to match.
Returns : the number of records deleted.

---

g_relation_destroy ()

```
void        g_relation_destroy              (GRelation *relation);
```

Destroys the GRelation, freeing all memory allocated. However, it does not free memory allocated for the tuple data, so you should free that first if appropriate.

relation : a GRelation.

---

g_relation_print ()

```
void        g_relation_print                (GRelation *relation);
```

Outputs information about all records in a GRelation, as well as the indexes. It is for debugging.

relation : a GRelation.

GTuples

```
typedef struct {
  guint len;
} GTuples;
```

The GTuples struct is used to return records (or tuples) from the GRelation by `g_relation_select()`. It only contains one public member - the number of records that matched. To access the matched records, you must use `g_tuples_index()`.

guint *len*; the number of records that matched.

g_tuples_destroy ()

```
void        g_tuples_destroy                (GTuples *tuples);
```

Frees the records which were returned by `g_relation_select()`. This should always be called after `g_relation_select()` when you are finished with the records. The records are not removed from the GRelation.

tuples : the tuple data to free.

g_tuples_index ()

```
gpointer    g_tuples_index                  (GTuples *tuples,
                                             gint index_,
                                             gint field);
```

Gets a field from the records returned by `g_relation_select()`. It returns the given field of the record at the given index. The returned value should not be changed.

tuples : the tuple data, returned by `g_relation_select()`.
*index_* : the index of the record.
field : the field to return.
Returns : the field of the record.

<< Datasets **Caches >>**

Caches

Caches — caches allow sharing of complex data structures to save resources.

Synopsis

```
#include <glib.h>


            GCache;
GCache*     g_cache_new                     (GCacheNewFunc value_new_func,
                                             GCacheDestroyFunc value_destroy_fun
                                             GCacheDupFunc key_dup_func,
                                             GCacheDestroyFunc key_destroy_func,
                                             GHashFunc hash_key_func,
                                             GHashFunc hash_value_func,
                                             GEqualFunc key_equal_func);
gpointer    g_cache_insert                  (GCache *cache,
                                             gpointer key);
void        g_cache_remove                  (GCache *cache,
                                             gconstpointer value);
void        g_cache_destroy                 (GCache *cache);

void        g_cache_key_foreach             (GCache *cache,
                                             GHFunc func,
                                             gpointer user_data);
void        g_cache_value_foreach           (GCache *cache,
                                             GHFunc func,
                                             gpointer user_data);

void        (*GCacheDestroyFunc)            (gpointer value);
gpointer    (*GCacheDupFunc)                (gpointer value);
gpointer    (*GCacheNewFunc)                (gpointer key);
```

Description

A GCache allows sharing of complex data structures, in order to save system resources.

GTK+ uses caches for GtkStyles and GdkGCs. These consume a lot of resources, so a GCache is used to see if a GtkStyle or GdkGC with the required properties already exists. If it does, then the existing object is used instead of creating a new one.

GCache uses keys and values. A GCache key describes the properties of a particular resource. A GCache value is the actual resource.

Details

GCache


```
typedef struct _GCache GCache;
```

The GCache struct is an opaque data structure containing information about a GCache. It should only be accessed via the following functions.

---

g_cache_new ()

```
GCache*     g_cache_new                     (GCacheNewFunc value_new_func,
                                             GCacheDestroyFunc value_destroy_fun
                                             GCacheDupFunc key_dup_func,
                                             GCacheDestroyFunc key_destroy_func,
                                             GHashFunc hash_key_func,
                                             GHashFunc hash_value_func,
                                             GEqualFunc key_equal_func);
```

Creates a new GCache.

value_new_func : a function to create a new object given a key. This is called by g_cache_insert() if an object with the given key does not already exist.

value_destroy_func : a function to destroy an object. It is called by g_cache_remove() when the object is no longer needed (i.e. its reference count drops to 0).

key_dup_func : a function to copy a key. It is called by g_cache_insert() if the key does not already exist in the GCache.

key_destroy_func : a function to destroy a key. It is called by g_cache_remove() when the object is no longer needed (i.e. its reference count drops to 0).

hash_key_func : a function to create a hash value from a key.

hash_value_func : a function to create a hash value from a value.

key_equal_func : a function to compare two keys. It should return TRUE if the two keys are equivalent.

Returns : a new GCache.

---

g_cache_insert ()

```
gpointer    g_cache_insert                  (GCache *cache,
                                             gpointer key);
```

Gets the value corresponding to the given key, creating it if necessary. It first checks if the value already exists in the GCache, by using the *key_equal_func* function passed to g_cache_new(). If it does already exist it is returned, and its reference count is increased by one. If the value does not currently exist, if is created by calling the *value_new_func*. The key is duplicated by calling *key_dup_func* and the duplicated key and value are inserted into the GCache.

cache : a GCache.

key : a key describing a GCache object.
Returns : a pointer to a GCache value.

---

g_cache_remove ()

```
void        g_cache_remove                  (GCache *cache,
                                             gconstpointer value);
```

Decreases the reference count of the given value. If it drops to 0 then the value and its corresponding key are destroyed, using the *value_destroy_func* and *key_destroy_func* passed to g_cache_new ().

cache : a GCache.
value : the value to remove.

---

g_cache_destroy ()

```
void        g_cache_destroy                 (GCache *cache);
```

Frees the memory allocated for the GCache.

Note that it does not destroy the keys and values which were contained in the GCache.

cache : a GCache.

---

g_cache_key_foreach ()

```
void        g_cache_key_foreach             (GCache *cache,
                                             GHFunc func,
                                             gpointer user_data);
```

Calls the given function for each of the keys in the GCache.

cache : a GCache.
func : the function to call with each GCache key.
user_data : user data to pass to the function.

---

g_cache_value_foreach ()

```
void        g_cache_value_foreach           (GCache *cache,
                                             GHFunc func,
                                             gpointer user_data);
```


Calls the given function for each of the values in the GCache.

cache : a GCache.
func : the function to call with each GCache value.
user_data : user data to pass to the function.

GCacheDestroyFunc ()

```
void        (*GCacheDestroyFunc)            (gpointer value);
```

Specifies the type of the *value_destroy_func* and *key_destroy_func* functions passed to g_cache_new(). The functions are passed a pointer to the GCache key or GCache value and should free any memory and other resources associated with it.

value : the GCache value to destroy.

GCacheDupFunc ()

```
gpointer    (*GCacheDupFunc)                (gpointer value);
```

Specifies the type of the *key_dup_func* function passed to g_cache_new(). The function is passed a key (*not* a value as the prototype implies) and should return a duplicate of the key.

value : the GCache key to destroy (*not* a GCache value as it seems).
Returns : a copy of the GCache key.

GCacheNewFunc ()

```
gpointer    (*GCacheNewFunc)                (gpointer key);
```

Specifies the type of the *value_new_func* function passed to g_cache_new(). It is passed a GCache key and should create the value corresponding to the key.

key : a GCache key.
Returns : a new GCache value corresponding to the key.

<< Relations and Tuples **Memory Allocators >>**

Memory Allocators

Memory Allocators — allocates chunks of memory for GList, GSList and GNode.

Synopsis

```
#include <glib.h>


            GAllocator;
GAllocator* g_allocator_new                 (const gchar *name,
                                             guint n_preallocs);
void        g_allocator_free                (GAllocator *allocator);
```

Description

The GAllocator is used as an efficient way to allocate small pieces of memory for use with the GList, GSList and GNode data structures. It uses a GMemChunk so elements are allocated in groups, rather than individually.

The GList, GSList and GNode implementations create default GAllocator objects, which are probably sufficient for most purposes. These default allocators use blocks of 128 elements.

To use your own GAllocator, create it with `g_allocator_new()`. Then use `g_list_push_allocator()`, `g_slist_push_allocator()` or `g_node_push_allocator()` before any code which allocates new GList, GSList or GNode elements respectively. After allocating the new elements, you must use `g_list_pop_allocator()`, `g_slist_pop_allocator()` or `g_node_pop_allocator()` to restore the previous allocators.

Note that you cannot use the same allocator for GList, GSList and GNode elements. Each must use separate allocators.

Details

GAllocator

```
typedef struct _GAllocator GAllocator;
```

The GAllocator struct contains private data. and should only be accessed using the following functions.

---

g_allocator_new ()


```
GAllocator* g_allocator_new                 (const gchar *name,
                                             guint n_preallocs);
```

Creates a new GAllocator.

name : the name of the GAllocator. This name is used to set the name of the GMemChunk used by the GAllocator, and is only used for debugging.

n_preallocs : the number of elements in each block of memory allocated. Larger blocks mean less calls to `g_malloc()`, but some memory may be wasted. (GLib uses 128 elements per block by default.) The value must be between 1 and 65535.

Returns : a new GAllocator.

---

g_allocator_free ()

```
void        g_allocator_free                (GAllocator *allocator);
```

Frees all of the memory allocated by the GAllocator.

allocator : a GAllocator.

<< Caches **GLib Tools >>**

GLib Tools

glib-gettextize - gettext internationalization utility

<< Memory Allocators **glib-gettextize >>**

glib-gettextize

glib-gettextize — gettext internationalization utility

Synopsis

`glib-gettextize` [option...] [directory]

Description

glib-gettextize helps to prepare a source package for being internationalized through gettext. It is a variant of the **gettextize** that ships with gettext.

glib-gettextize differs from **gettextize** in that it doesn't create an `intl/` subdirectory and doesn't modify `po/ChangeLog` (note that newer versions of **gettextize** behave like this when called with the `--no-changelog` option).

Options

`--help`	print help and exit
`--version`	print version information and exit
`-c`, `--copy`	copy files instead of making symlinks
`-f`, `--force`	force writing of new files even if old ones exist

See also

gettextize(1)

Index

Symbols

_, _()

A

ABS, ABS()

C

CLAMP, CLAMP()

F

FALSE, FALSE

G

GAllocator, GAllocator
GArray, GArray
GAsyncQueue, GAsyncQueue
gboolean, gboolean
GByteArray, GByteArray
GCache, GCache
GCacheDestroyFunc, GCacheDestroyFunc ()
GCacheDupFunc, GCacheDupFunc ()
GCacheNewFunc, GCacheNewFunc ()
gchar, gchar
GChildWatchFunc, GChildWatchFunc ()
GCompareDataFunc, GCompareDataFunc ()
GCompareFunc, GCompareFunc ()
GCompletion, GCompletion
GCompletionFunc, GCompletionFunc ()
GCompletionStrncmpFunc, GCompletionStrncmpFunc ()
GCond, GCond
gconstpointer, gconstpointer
GConvertError, enum GConvertError
GCopyFunc, GCopyFunc ()
GData, GData
GDataForeachFunc, GDataForeachFunc ()
GDate, GDate
GDateDay, GDateDay
GDateDMY, enum GDateDMY
GDateMonth, enum GDateMonth
GDateWeekday, enum GDateWeekday
GDateYear, GDateYear
GDebugKey, GDebugKey

GDestroyNotify, GDestroyNotify ()
GDir, GDir
gdouble, gdouble
GDoubleIEEE754, union GDoubleIEEE754
GEqualFunc, GEqualFunc ()
GError, GError
GErrorType, enum GErrorType
GFileError, enum GFileError
GFileTest, enum GFileTest
gfloat, gfloat
GFloatIEEE754, union GFloatIEEE754
GFreeFunc, GFreeFunc ()
GFunc, GFunc ()
GHashFunc, GHashFunc ()
GHashTable, GHashTable
GHFunc, GHFunc ()
GHook, GHook
GHookCheckFunc, GHookCheckFunc ()
GHookCheckMarshaller, GHookCheckMarshaller ()
GHookCompareFunc, GHookCompareFunc ()
GHookFinalizeFunc, GHookFinalizeFunc ()
GHookFindFunc, GHookFindFunc ()
GHookFlagMask, enum GHookFlagMask
GHookFunc, GHookFunc ()
GHookList, GHookList
GHookMarshaller, GHookMarshaller ()
GHRFunc, GHRFunc ()
GIConv, GIConv
gint, gint
gint16, gint16
GINT16_FROM_BE, GINT16_FROM_BE()
GINT16_FROM_LE, GINT16_FROM_LE()
GINT16_TO_BE, GINT16_TO_BE()
GINT16_TO_LE, GINT16_TO_LE()
gint32, gint32
GINT32_FROM_BE, GINT32_FROM_BE()
GINT32_FROM_LE, GINT32_FROM_LE()
GINT32_TO_BE, GINT32_TO_BE()
GINT32_TO_LE, GINT32_TO_LE()
gint64, gint64
GINT64_FROM_BE, GINT64_FROM_BE()
GINT64_FROM_LE, GINT64_FROM_LE()
GINT64_TO_BE, GINT64_TO_BE()
GINT64_TO_LE, GINT64_TO_LE()
gint8, gint8
GINT_FROM_BE, GINT_FROM_BE()
GINT_FROM_LE, GINT_FROM_LE()
GINT_TO_BE, GINT_TO_BE()
GINT_TO_LE, GINT_TO_LE()
GINT_TO_POINTER, GINT_TO_POINTER()
GIOChannel, GIOChannel
GIOChannelError, enum GIOChannelError
GIOCondition, enum GIOCondition
GIOError, enum GIOError
GIOFlags, enum GIOFlags

GIOFunc, GIOFunc ()
GIOFuncs, GIOFuncs
GIOStatus, enum GIOStatus
GKeyFile, GKeyFile
GKeyFileError, enum GKeyFileError
GKeyFileFlags, enum GKeyFileFlags
glib_binary_age, glib_binary_age
glib_check_version, glib_check_version ()
GLIB_CHECK_VERSION, GLIB_CHECK_VERSION()
glib_interface_age, glib_interface_age
glib_major_version, glib_major_version
GLIB_MAJOR_VERSION, GLIB_MAJOR_VERSION
glib_mem_profiler_table, glib_mem_profiler_table
glib_micro_version, glib_micro_version
GLIB_MICRO_VERSION, GLIB_MICRO_VERSION
glib_minor_version, glib_minor_version
GLIB_MINOR_VERSION, GLIB_MINOR_VERSION
GList, GList
GLogFunc, GLogFunc ()
GLogLevelFlags, enum GLogLevelFlags
glong, glong
GLONG_FROM_BE, GLONG_FROM_BE()
GLONG_FROM_LE, GLONG_FROM_LE()
GLONG_TO_BE, GLONG_TO_BE()
GLONG_TO_LE, GLONG_TO_LE()
GMainContext, GMainContext
GMainLoop, GMainLoop
GMarkupError, enum GMarkupError
GMarkupParseContext, GMarkupParseContext
GMarkupParseFlags, enum GMarkupParseFlags
GMarkupParser, GMarkupParser
GMemChunk, GMemChunk
GMemVTable, GMemVTable
GModule, GModule
GModuleCheckInit, GModuleCheckInit ()
GModuleFlags, enum GModuleFlags
GModuleUnload, GModuleUnload ()
GMutex, GMutex
GNode, GNode
GNodeForeachFunc, GNodeForeachFunc ()
GNodeTraverseFunc, GNodeTraverseFunc ()
GNormalizeMode, enum GNormalizeMode
GOnce, GOnce
GOnceStatus, enum GOnceStatus
GOptionArg, enum GOptionArg
GOptionArgFunc, GOptionArgFunc ()
GOptionContext, GOptionContext
GOptionEntry, GOptionEntry
GOptionError, enum GOptionError
GOptionErrorFunc, GOptionErrorFunc ()
GOptionFlags, enum GOptionFlags
GOptionGroup, GOptionGroup
GOptionParseFunc, GOptionParseFunc ()
GPatternSpec, GPatternSpec
GPid, GPid

gpointer, gpointer
GPOINTER_TO_INT, GPOINTER_TO_INT()
GPOINTER_TO_SIZE, GPOINTER_TO_SIZE()
GPOINTER_TO_UINT, GPOINTER_TO_UINT()
GPollFD, GPollFD
GPollFunc, GPollFunc ()
GPrintFunc, GPrintFunc ()
GPrivate, GPrivate
GPtrArray, GPtrArray
GQuark, GQuark
GQueue, GQueue
GRand, GRand
GRelation, GRelation
GScanner, GScanner
GScannerConfig, GScannerConfig
GScannerMsgFunc, GScannerMsgFunc ()
GSeekType, enum GSeekType
GShellError, enum GShellError
gshort, gshort
gsize, gsize
GSIZE_TO_POINTER, GSIZE_TO_POINTER()
GSList, GSList
GSource, GSource
GSourceCallbackFuncs, GSourceCallbackFuncs
GSourceDummyMarshal, GSourceDummyMarshal ()
GSourceFunc, GSourceFunc ()
GSourceFuncs, GSourceFuncs
GSpawnChildSetupFunc, GSpawnChildSetupFunc ()
GSpawnError, enum GSpawnError
GSpawnFlags, enum GSpawnFlags
gssize, gssize
GStaticMutex, GStaticMutex
GStaticPrivate, GStaticPrivate
GStaticRecMutex, GStaticRecMutex
GStaticRWLock, GStaticRWLock
GString, GString
GStringChunk, GStringChunk
GThread, GThread
GThreadError, enum GThreadError
GThreadFunc, GThreadFunc ()
GThreadFunctions, GThreadFunctions
GThreadPool, GThreadPool
GThreadPriority, enum GThreadPriority
GTime, GTime
GTimer, GTimer
GTimeVal, GTimeVal
GTokenType, enum GTokenType
GTokenValue, union GTokenValue
GTranslateFunc, GTranslateFunc ()
GTrashStack, GTrashStack
GTraverseFlags, enum GTraverseFlags
GTraverseFunc, GTraverseFunc ()
GTraverseType, enum GTraverseType
GTree, GTree
GTuples, GTuples

guchar, guchar
guint, guint
guint16, guint16
GUINT16_FROM_BE, GUINT16_FROM_BE()
GUINT16_FROM_LE, GUINT16_FROM_LE()
GUINT16_SWAP_BE_PDP, GUINT16_SWAP_BE_PDP()
GUINT16_SWAP_LE_BE, GUINT16_SWAP_LE_BE()
GUINT16_SWAP_LE_PDP, GUINT16_SWAP_LE_PDP()
GUINT16_TO_BE, GUINT16_TO_BE()
GUINT16_TO_LE, GUINT16_TO_LE()
guint32, guint32
GUINT32_FROM_BE, GUINT32_FROM_BE()
GUINT32_FROM_LE, GUINT32_FROM_LE()
GUINT32_SWAP_BE_PDP, GUINT32_SWAP_BE_PDP()
GUINT32_SWAP_LE_BE, GUINT32_SWAP_LE_BE()
GUINT32_SWAP_LE_PDP, GUINT32_SWAP_LE_PDP()
GUINT32_TO_BE, GUINT32_TO_BE()
GUINT32_TO_LE, GUINT32_TO_LE()
guint64, guint64
GUINT64_FROM_BE, GUINT64_FROM_BE()
GUINT64_FROM_LE, GUINT64_FROM_LE()
GUINT64_SWAP_LE_BE, GUINT64_SWAP_LE_BE()
GUINT64_TO_BE, GUINT64_TO_BE()
GUINT64_TO_LE, GUINT64_TO_LE()
guint8, guint8
GUINT_FROM_BE, GUINT_FROM_BE()
GUINT_FROM_LE, GUINT_FROM_LE()
GUINT_TO_BE, GUINT_TO_BE()
GUINT_TO_LE, GUINT_TO_LE()
GUINT_TO_POINTER, GUINT_TO_POINTER()
gulong, gulong
GULONG_FROM_BE, GULONG_FROM_BE()
GULONG_FROM_LE, GULONG_FROM_LE()
GULONG_TO_BE, GULONG_TO_BE()
GULONG_TO_LE, GULONG_TO_LE()
gunichar, gunichar
gunichar2, gunichar2
GUnicodeBreakType, enum GUnicodeBreakType
GUnicodeType, enum GUnicodeType
gushort, gushort
GVoidFunc, GVoidFunc ()
g_alloca, g_alloca()
g_allocator_free, g_allocator_free ()
g_allocator_new, g_allocator_new ()
G_ALLOC_AND_FREE, G_ALLOC_AND_FREE
G_ALLOC_ONLY, G_ALLOC_ONLY
g_array_append_val, g_array_append_val()
g_array_append_vals, g_array_append_vals ()
g_array_free, g_array_free ()
g_array_index, g_array_index()
g_array_insert_val, g_array_insert_val()
g_array_insert_vals, g_array_insert_vals ()
g_array_new, g_array_new ()
g_array_prepend_val, g_array_prepend_val()
g_array_prepend_vals, g_array_prepend_vals ()

g_array_remove_index, g_array_remove_index ()
g_array_remove_index_fast, g_array_remove_index_fast ()
g_array_remove_range, g_array_remove_range ()
g_array_set_size, g_array_set_size ()
g_array_sized_new, g_array_sized_new ()
g_array_sort, g_array_sort ()
g_array_sort_with_data, g_array_sort_with_data ()
g_ascii_digit_value, g_ascii_digit_value ()
g_ascii_dtostr, g_ascii_dtostr ()
G_ASCII_DTOSTR_BUF_SIZE, G_ASCII_DTOSTR_BUF_SIZE
g_ascii_formatd, g_ascii_formatd ()
g_ascii_isalnum, g_ascii_isalnum ()
g_ascii_isalpha, g_ascii_isalpha ()
g_ascii_iscntrl, g_ascii_iscntrl ()
g_ascii_isdigit, g_ascii_isdigit ()
g_ascii_isgraph, g_ascii_isgraph ()
g_ascii_islower, g_ascii_islower ()
g_ascii_isprint, g_ascii_isprint ()
g_ascii_ispunct, g_ascii_ispunct ()
g_ascii_isspace, g_ascii_isspace ()
g_ascii_isupper, g_ascii_isupper ()
g_ascii_isxdigit, g_ascii_isxdigit ()
g_ascii_strcasecmp, g_ascii_strcasecmp ()
g_ascii_strdown, g_ascii_strdown ()
g_ascii_strncasecmp, g_ascii_strncasecmp ()
g_ascii_strtod, g_ascii_strtod ()
g_ascii_strtoull, g_ascii_strtoull ()
g_ascii_strup, g_ascii_strup ()
g_ascii_tolower, g_ascii_tolower ()
g_ascii_toupper, g_ascii_toupper ()
g_ascii_xdigit_value, g_ascii_xdigit_value ()
g_assert, g_assert()
g_assert_not_reached, g_assert_not_reached()
g_async_queue_length, g_async_queue_length ()
g_async_queue_length_unlocked, g_async_queue_length_unlocked ()
g_async_queue_lock, g_async_queue_lock ()
g_async_queue_new, g_async_queue_new ()
g_async_queue_pop, g_async_queue_pop ()
g_async_queue_pop_unlocked, g_async_queue_pop_unlocked ()
g_async_queue_push, g_async_queue_push ()
g_async_queue_push_unlocked, g_async_queue_push_unlocked ()
g_async_queue_ref, g_async_queue_ref ()
g_async_queue_ref_unlocked, g_async_queue_ref_unlocked ()
g_async_queue_timed_pop, g_async_queue_timed_pop ()
g_async_queue_timed_pop_unlocked, g_async_queue_timed_pop_unlocked ()
g_async_queue_try_pop, g_async_queue_try_pop ()
g_async_queue_try_pop_unlocked, g_async_queue_try_pop_unlocked ()
g_async_queue_unlock, g_async_queue_unlock ()
g_async_queue_unref, g_async_queue_unref ()
g_async_queue_unref_and_unlock, g_async_queue_unref_and_unlock ()
g_atexit, g_atexit ()
g_atomic_int_add, g_atomic_int_add ()
g_atomic_int_compare_and_exchange, g_atomic_int_compare_and_exchange ()
g_atomic_int_dec_and_test, g_atomic_int_dec_and_test ()
g_atomic_int_exchange_and_add, g_atomic_int_exchange_and_add ()

g_atomic_int_get, g_atomic_int_get ()
g_atomic_int_inc, g_atomic_int_inc ()
g_atomic_pointer_compare_and_exchange, g_atomic_pointer_compare_and_exchange ()
g_atomic_pointer_get, g_atomic_pointer_get ()
g_basename, g_basename ()
G_BEGIN_DECLS, G_BEGIN_DECLS
G_BIG_ENDIAN, G_BIG_ENDIAN
g_bit_nth_lsf, g_bit_nth_lsf ()
g_bit_nth_msf, g_bit_nth_msf ()
g_bit_storage, g_bit_storage ()
g_blow_chunks, g_blow_chunks ()
G_BREAKPOINT, G_BREAKPOINT()
g_build_filename, g_build_filename ()
g_build_path, g_build_path ()
g_byte_array_append, g_byte_array_append ()
g_byte_array_free, g_byte_array_free ()
g_byte_array_new, g_byte_array_new ()
g_byte_array_prepend, g_byte_array_prepend ()
g_byte_array_remove_index, g_byte_array_remove_index ()
g_byte_array_remove_index_fast, g_byte_array_remove_index_fast ()
g_byte_array_remove_range, g_byte_array_remove_range ()
g_byte_array_set_size, g_byte_array_set_size ()
g_byte_array_sized_new, g_byte_array_sized_new ()
g_byte_array_sort, g_byte_array_sort ()
g_byte_array_sort_with_data, g_byte_array_sort_with_data ()
G_BYTE_ORDER, G_BYTE_ORDER
g_cache_destroy, g_cache_destroy ()
g_cache_insert, g_cache_insert ()
g_cache_key_foreach, g_cache_key_foreach ()
g_cache_new, g_cache_new ()
g_cache_remove, g_cache_remove ()
g_cache_value_foreach, g_cache_value_foreach ()
g_child_watch_add, g_child_watch_add ()
g_child_watch_add_full, g_child_watch_add_full ()
g_child_watch_source_new, g_child_watch_source_new ()
g_chunk_free, g_chunk_free()
g_chunk_new, g_chunk_new()
g_chunk_new0, g_chunk_new0()
g_clear_error, g_clear_error ()
g_completion_add_items, g_completion_add_items ()
g_completion_clear_items, g_completion_clear_items ()
g_completion_complete, g_completion_complete ()
g_completion_complete_utf8, g_completion_complete_utf8 ()
g_completion_free, g_completion_free ()
g_completion_new, g_completion_new ()
g_completion_remove_items, g_completion_remove_items ()
g_completion_set_compare, g_completion_set_compare ()
g_cond_broadcast, g_cond_broadcast ()
g_cond_free, g_cond_free ()
g_cond_new, g_cond_new ()
g_cond_signal, g_cond_signal ()
g_cond_timed_wait, g_cond_timed_wait ()
g_cond_wait, g_cond_wait ()
G_CONST_RETURN, G_CONST_RETURN
g_convert, g_convert ()

G_CONVERT_ERROR, G_CONVERT_ERROR
g_convert_with_fallback, g_convert_with_fallback ()
g_convert_with_iconv, g_convert_with_iconv ()
g_critical, g_critical()
G_CSET_a_2_z, G_CSET_a_2_z
G_CSET_A_2_Z, G_CSET_A_2_Z
G_CSET_DIGITS, G_CSET_DIGITS
G_CSET_LATINC, G_CSET_LATINC
G_CSET_LATINS, G_CSET_LATINS
g_datalist_clear, g_datalist_clear ()
g_datalist_foreach, g_datalist_foreach ()
g_datalist_get_data, g_datalist_get_data()
g_datalist_id_get_data, g_datalist_id_get_data ()
g_datalist_id_remove_data, g_datalist_id_remove_data()
g_datalist_id_remove_no_notify, g_datalist_id_remove_no_notify ()
g_datalist_id_set_data, g_datalist_id_set_data()
g_datalist_id_set_data_full, g_datalist_id_set_data_full ()
g_datalist_init, g_datalist_init ()
g_datalist_remove_data, g_datalist_remove_data()
g_datalist_remove_no_notify, g_datalist_remove_no_notify()
g_datalist_set_data, g_datalist_set_data()
g_datalist_set_data_full, g_datalist_set_data_full()
g_dataset_destroy, g_dataset_destroy ()
g_dataset_foreach, g_dataset_foreach ()
g_dataset_get_data, g_dataset_get_data()
g_dataset_id_get_data, g_dataset_id_get_data ()
g_dataset_id_remove_data, g_dataset_id_remove_data()
g_dataset_id_remove_no_notify, g_dataset_id_remove_no_notify ()
g_dataset_id_set_data, g_dataset_id_set_data()
g_dataset_id_set_data_full, g_dataset_id_set_data_full ()
g_dataset_remove_data, g_dataset_remove_data()
g_dataset_remove_no_notify, g_dataset_remove_no_notify()
g_dataset_set_data, g_dataset_set_data()
g_dataset_set_data_full, g_dataset_set_data_full()
g_date_add_days, g_date_add_days ()
g_date_add_months, g_date_add_months ()
g_date_add_years, g_date_add_years ()
G_DATE_BAD_DAY, G_DATE_BAD_DAY
G_DATE_BAD_JULIAN, G_DATE_BAD_JULIAN
G_DATE_BAD_YEAR, G_DATE_BAD_YEAR
g_date_clamp, g_date_clamp ()
g_date_clear, g_date_clear ()
g_date_compare, g_date_compare ()
g_date_days_between, g_date_days_between ()
g_date_free, g_date_free ()
g_date_get_day, g_date_get_day ()
g_date_get_days_in_month, g_date_get_days_in_month ()
g_date_get_day_of_year, g_date_get_day_of_year ()
g_date_get_iso8601_week_of_year, g_date_get_iso8601_week_of_year ()
g_date_get_julian, g_date_get_julian ()
g_date_get_monday_weeks_in_year, g_date_get_monday_weeks_in_year ()
g_date_get_monday_week_of_year, g_date_get_monday_week_of_year ()
g_date_get_month, g_date_get_month ()
g_date_get_sunday_weeks_in_year, g_date_get_sunday_weeks_in_year ()
g_date_get_sunday_week_of_year, g_date_get_sunday_week_of_year ()

g_date_get_weekday, g_date_get_weekday ()
g_date_get_year, g_date_get_year ()
g_date_is_first_of_month, g_date_is_first_of_month ()
g_date_is_last_of_month, g_date_is_last_of_month ()
g_date_is_leap_year, g_date_is_leap_year ()
g_date_new, g_date_new ()
g_date_new_dmy, g_date_new_dmy ()
g_date_new_julian, g_date_new_julian ()
g_date_order, g_date_order ()
g_date_set_day, g_date_set_day ()
g_date_set_dmy, g_date_set_dmy ()
g_date_set_julian, g_date_set_julian ()
g_date_set_month, g_date_set_month ()
g_date_set_parse, g_date_set_parse ()
g_date_set_time, g_date_set_time ()
g_date_set_year, g_date_set_year ()
g_date_strftime, g_date_strftime ()
g_date_subtract_days, g_date_subtract_days ()
g_date_subtract_months, g_date_subtract_months ()
g_date_subtract_years, g_date_subtract_years ()
g_date_to_struct_tm, g_date_to_struct_tm ()
g_date_valid, g_date_valid ()
g_date_valid_day, g_date_valid_day ()
g_date_valid_dmy, g_date_valid_dmy ()
g_date_valid_julian, g_date_valid_julian ()
g_date_valid_month, g_date_valid_month ()
g_date_valid_weekday, g_date_valid_weekday ()
g_date_valid_year, g_date_valid_year ()
g_debug, g_debug()
g_direct_equal, g_direct_equal ()
g_direct_hash, g_direct_hash ()
g_dirname, g_dirname
g_dir_close, g_dir_close ()
g_dir_open, g_dir_open ()
g_dir_read_name, g_dir_read_name ()
g_dir_rewind, g_dir_rewind ()
G_DIR_SEPARATOR, G_DIR_SEPARATOR
G_DIR_SEPARATOR_S, G_DIR_SEPARATOR_S
G_E, G_E
G_END_DECLS, G_END_DECLS
g_error, g_error()
g_error_copy, g_error_copy ()
g_error_free, g_error_free ()
g_error_matches, g_error_matches ()
g_error_new, g_error_new ()
g_error_new_literal, g_error_new_literal ()
g_filename_display_name, g_filename_display_name ()
g_filename_from_uri, g_filename_from_uri ()
g_filename_from_utf8, g_filename_from_utf8 ()
g_filename_to_uri, g_filename_to_uri ()
g_filename_to_utf8, g_filename_to_utf8 ()
G_FILE_ERROR, G_FILE_ERROR
g_file_error_from_errno, g_file_error_from_errno ()
g_file_get_contents, g_file_get_contents ()
g_file_open_tmp, g_file_open_tmp ()

g_file_read_link, g_file_read_link ()
g_file_test, g_file_test ()
g_find_program_in_path, g_find_program_in_path ()
g_fopen, g_fopen ()
g_fprintf, g_fprintf ()
g_free, g_free ()
g_freopen, g_freopen ()
g_getenv, g_getenv ()
g_get_application_name, g_get_application_name ()
g_get_charset, g_get_charset ()
g_get_current_dir, g_get_current_dir ()
g_get_current_time, g_get_current_time ()
g_get_filename_charsets, g_get_filename_charsets ()
g_get_home_dir, g_get_home_dir ()
g_get_language_names, g_get_language_names ()
g_get_prgname, g_get_prgname ()
g_get_real_name, g_get_real_name ()
g_get_system_config_dirs, g_get_system_config_dirs ()
g_get_system_data_dirs, g_get_system_data_dirs ()
g_get_tmp_dir, g_get_tmp_dir ()
g_get_user_cache_dir, g_get_user_cache_dir ()
g_get_user_config_dir, g_get_user_config_dir ()
g_get_user_data_dir, g_get_user_data_dir ()
g_get_user_name, g_get_user_name ()
G_GINT16_FORMAT, G_GINT16_FORMAT
G_GINT16_MODIFIER, G_GINT16_MODIFIER
G_GINT32_FORMAT, G_GINT32_FORMAT
G_GINT32_MODIFIER, G_GINT32_MODIFIER
G_GINT64_CONSTANT, G_GINT64_CONSTANT()
G_GINT64_FORMAT, G_GINT64_FORMAT
G_GINT64_MODIFIER, G_GINT64_MODIFIER
G_GNUC_CONST, G_GNUC_CONST
G_GNUC_DEPRECATED, G_GNUC_DEPRECATED
G_GNUC_EXTENSION, G_GNUC_EXTENSION
G_GNUC_FORMAT, G_GNUC_FORMAT()
G_GNUC_FUNCTION, G_GNUC_FUNCTION
G_GNUC_INTERNAL, G_GNUC_INTERNAL
G_GNUC_NORETURN, G_GNUC_NORETURN
G_GNUC_NO_INSTRUMENT, G_GNUC_NO_INSTRUMENT
G_GNUC_PRETTY_FUNCTION, G_GNUC_PRETTY_FUNCTION
G_GNUC_PRINTF, G_GNUC_PRINTF()
G_GNUC_PURE, G_GNUC_PURE
G_GNUC_SCANF, G_GNUC_SCANF()
G_GNUC_UNUSED, G_GNUC_UNUSED
G_GSIZE_FORMAT, G_GSIZE_FORMAT
G_GSIZE_MODIFIER, G_GSIZE_MODIFIER
G_GSSIZE_FORMAT, G_GSSIZE_FORMAT
G_GUINT16_FORMAT, G_GUINT16_FORMAT
G_GUINT32_FORMAT, G_GUINT32_FORMAT
G_GUINT64_FORMAT, G_GUINT64_FORMAT
g_hash_table_destroy, g_hash_table_destroy ()
g_hash_table_find, g_hash_table_find ()
g_hash_table_foreach, g_hash_table_foreach ()
g_hash_table_foreach_remove, g_hash_table_foreach_remove ()
g_hash_table_foreach_steal, g_hash_table_foreach_steal ()

g_hash_table_freeze, g_hash_table_freeze()
g_hash_table_insert, g_hash_table_insert ()
g_hash_table_lookup, g_hash_table_lookup ()
g_hash_table_lookup_extended, g_hash_table_lookup_extended ()
g_hash_table_new, g_hash_table_new ()
g_hash_table_new_full, g_hash_table_new_full ()
g_hash_table_remove, g_hash_table_remove ()
g_hash_table_replace, g_hash_table_replace ()
g_hash_table_size, g_hash_table_size ()
g_hash_table_steal, g_hash_table_steal ()
g_hash_table_thaw, g_hash_table_thaw()
G_HAVE_GINT64, G_HAVE_GINT64
G_HOOK, G_HOOK()
G_HOOK_ACTIVE, G_HOOK_ACTIVE()
g_hook_alloc, g_hook_alloc ()
g_hook_append, g_hook_append()
g_hook_compare_ids, g_hook_compare_ids ()
g_hook_destroy, g_hook_destroy ()
g_hook_destroy_link, g_hook_destroy_link ()
g_hook_find, g_hook_find ()
g_hook_find_data, g_hook_find_data ()
g_hook_find_func, g_hook_find_func ()
g_hook_find_func_data, g_hook_find_func_data ()
g_hook_first_valid, g_hook_first_valid ()
G_HOOK_FLAGS, G_HOOK_FLAGS()
G_HOOK_FLAG_USER_SHIFT, G_HOOK_FLAG_USER_SHIFT
g_hook_free, g_hook_free ()
g_hook_get, g_hook_get ()
g_hook_insert_before, g_hook_insert_before ()
g_hook_insert_sorted, g_hook_insert_sorted ()
G_HOOK_IN_CALL, G_HOOK_IN_CALL()
G_HOOK_IS_UNLINKED, G_HOOK_IS_UNLINKED()
G_HOOK_IS_VALID, G_HOOK_IS_VALID()
g_hook_list_clear, g_hook_list_clear ()
g_hook_list_init, g_hook_list_init ()
g_hook_list_invoke, g_hook_list_invoke ()
g_hook_list_invoke_check, g_hook_list_invoke_check ()
g_hook_list_marshal, g_hook_list_marshal ()
g_hook_list_marshal_check, g_hook_list_marshal_check ()
g_hook_next_valid, g_hook_next_valid ()
g_hook_prepend, g_hook_prepend ()
g_hook_ref, g_hook_ref ()
g_hook_unref, g_hook_unref ()
g_htonl, g_htonl()
g_htons, g_htons()
g_iconv, g_iconv ()
g_iconv_close, g_iconv_close ()
g_iconv_open, g_iconv_open ()
g_idle_add, g_idle_add ()
g_idle_add_full, g_idle_add_full ()
g_idle_remove_by_data, g_idle_remove_by_data ()
g_idle_source_new, g_idle_source_new ()
G_IEEE754_DOUBLE_BIAS, G_IEEE754_DOUBLE_BIAS
G_IEEE754_FLOAT_BIAS, G_IEEE754_FLOAT_BIAS
G_INLINE_FUNC, G_INLINE_FUNC

g_int_equal, g_int_equal ()
g_int_hash, g_int_hash ()
g_io_add_watch, g_io_add_watch ()
g_io_add_watch_full, g_io_add_watch_full ()
g_io_channel_close, g_io_channel_close ()
G_IO_CHANNEL_ERROR, G_IO_CHANNEL_ERROR
g_io_channel_error_from_errno, g_io_channel_error_from_errno ()
g_io_channel_flush, g_io_channel_flush ()
g_io_channel_get_buffered, g_io_channel_get_buffered ()
g_io_channel_get_buffer_condition, g_io_channel_get_buffer_condition ()
g_io_channel_get_buffer_size, g_io_channel_get_buffer_size ()
g_io_channel_get_close_on_unref, g_io_channel_get_close_on_unref ()
g_io_channel_get_encoding, g_io_channel_get_encoding ()
g_io_channel_get_flags, g_io_channel_get_flags ()
g_io_channel_get_line_term, g_io_channel_get_line_term ()
g_io_channel_init, g_io_channel_init ()
g_io_channel_new_file, g_io_channel_new_file ()
g_io_channel_read, g_io_channel_read ()
g_io_channel_read_chars, g_io_channel_read_chars ()
g_io_channel_read_line, g_io_channel_read_line ()
g_io_channel_read_line_string, g_io_channel_read_line_string ()
g_io_channel_read_to_end, g_io_channel_read_to_end ()
g_io_channel_read_unichar, g_io_channel_read_unichar ()
g_io_channel_ref, g_io_channel_ref ()
g_io_channel_seek, g_io_channel_seek ()
g_io_channel_seek_position, g_io_channel_seek_position ()
g_io_channel_set_buffered, g_io_channel_set_buffered ()
g_io_channel_set_buffer_size, g_io_channel_set_buffer_size ()
g_io_channel_set_close_on_unref, g_io_channel_set_close_on_unref ()
g_io_channel_set_encoding, g_io_channel_set_encoding ()
g_io_channel_set_flags, g_io_channel_set_flags ()
g_io_channel_set_line_term, g_io_channel_set_line_term ()
g_io_channel_shutdown, g_io_channel_shutdown ()
g_io_channel_unix_get_fd, g_io_channel_unix_get_fd ()
g_io_channel_unix_new, g_io_channel_unix_new ()
g_io_channel_unref, g_io_channel_unref ()
g_io_channel_write, g_io_channel_write ()
g_io_channel_write_chars, g_io_channel_write_chars ()
g_io_channel_write_unichar, g_io_channel_write_unichar ()
g_io_create_watch, g_io_create_watch ()
G_IS_DIR_SEPARATOR, G_IS_DIR_SEPARATOR()
G_KEY_FILE_ERROR, G_KEY_FILE_ERROR
g_key_file_free, g_key_file_free ()
g_key_file_get_boolean, g_key_file_get_boolean ()
g_key_file_get_boolean_list, g_key_file_get_boolean_list ()
g_key_file_get_comment, g_key_file_get_comment ()
g_key_file_get_groups, g_key_file_get_groups ()
g_key_file_get_integer, g_key_file_get_integer ()
g_key_file_get_integer_list, g_key_file_get_integer_list ()
g_key_file_get_keys, g_key_file_get_keys ()
g_key_file_get_locale_string, g_key_file_get_locale_string ()
g_key_file_get_locale_string_list, g_key_file_get_locale_string_list ()
g_key_file_get_start_group, g_key_file_get_start_group ()
g_key_file_get_string, g_key_file_get_string ()
g_key_file_get_string_list, g_key_file_get_string_list ()

g_key_file_get_value, g_key_file_get_value ()
g_key_file_has_group, g_key_file_has_group ()
g_key_file_has_key, g_key_file_has_key ()
g_key_file_load_from_data, g_key_file_load_from_data ()
g_key_file_load_from_data_dirs, g_key_file_load_from_data_dirs ()
g_key_file_load_from_file, g_key_file_load_from_file ()
g_key_file_new, g_key_file_new ()
g_key_file_remove_comment, g_key_file_remove_comment ()
g_key_file_remove_group, g_key_file_remove_group ()
g_key_file_remove_key, g_key_file_remove_key ()
g_key_file_set_boolean, g_key_file_set_boolean ()
g_key_file_set_boolean_list, g_key_file_set_boolean_list ()
g_key_file_set_comment, g_key_file_set_comment ()
g_key_file_set_integer, g_key_file_set_integer ()
g_key_file_set_integer_list, g_key_file_set_integer_list ()
g_key_file_set_list_separator, g_key_file_set_list_separator ()
g_key_file_set_locale_string, g_key_file_set_locale_string ()
g_key_file_set_locale_string_list, g_key_file_set_locale_string_list ()
g_key_file_set_string, g_key_file_set_string ()
g_key_file_set_string_list, g_key_file_set_string_list ()
g_key_file_set_value, g_key_file_set_value ()
g_key_file_to_data, g_key_file_to_data ()
G_LIKELY, G_LIKELY()
g_list_alloc, g_list_alloc ()
g_list_append, g_list_append ()
g_list_concat, g_list_concat ()
g_list_copy, g_list_copy ()
g_list_delete_link, g_list_delete_link ()
g_list_find, g_list_find ()
g_list_find_custom, g_list_find_custom ()
g_list_first, g_list_first ()
g_list_foreach, g_list_foreach ()
g_list_free, g_list_free ()
g_list_free_1, g_list_free_1 ()
g_list_index, g_list_index ()
g_list_insert, g_list_insert ()
g_list_insert_before, g_list_insert_before ()
g_list_insert_sorted, g_list_insert_sorted ()
g_list_last, g_list_last ()
g_list_length, g_list_length ()
g_list_next, g_list_next()
g_list_nth, g_list_nth ()
g_list_nth_data, g_list_nth_data ()
g_list_nth_prev, g_list_nth_prev ()
g_list_pop_allocator, g_list_pop_allocator ()
g_list_position, g_list_position ()
g_list_prepend, g_list_prepend ()
g_list_previous, g_list_previous()
g_list_push_allocator, g_list_push_allocator ()
g_list_remove, g_list_remove ()
g_list_remove_all, g_list_remove_all ()
g_list_remove_link, g_list_remove_link ()
g_list_reverse, g_list_reverse ()
g_list_sort, g_list_sort ()
g_list_sort_with_data, g_list_sort_with_data ()

G_LITTLE_ENDIAN, G_LITTLE_ENDIAN
G_LN10, G_LN10
G_LN2, G_LN2
g_locale_from_utf8, g_locale_from_utf8 ()
g_locale_to_utf8, g_locale_to_utf8 ()
G_LOCK, G_LOCK()
G_LOCK_DEFINE, G_LOCK_DEFINE()
G_LOCK_DEFINE_STATIC, G_LOCK_DEFINE_STATIC()
G_LOCK_EXTERN, G_LOCK_EXTERN()
g_log, g_log ()
g_logv, g_logv ()
G_LOG_2_BASE_10, G_LOG_2_BASE_10
g_log_default_handler, g_log_default_handler ()
G_LOG_DOMAIN, G_LOG_DOMAIN
G_LOG_FATAL_MASK, G_LOG_FATAL_MASK
G_LOG_LEVEL_USER_SHIFT, G_LOG_LEVEL_USER_SHIFT
g_log_remove_handler, g_log_remove_handler ()
g_log_set_always_fatal, g_log_set_always_fatal ()
g_log_set_default_handler, g_log_set_default_handler ()
g_log_set_fatal_mask, g_log_set_fatal_mask ()
g_log_set_handler, g_log_set_handler ()
g_lstat, g_lstat ()
g_main_context_acquire, g_main_context_acquire ()
g_main_context_add_poll, g_main_context_add_poll ()
g_main_context_check, g_main_context_check ()
g_main_context_default, g_main_context_default ()
g_main_context_dispatch, g_main_context_dispatch ()
g_main_context_find_source_by_funcs_user_data,
g_main_context_find_source_by_funcs_user_data ()
g_main_context_find_source_by_id, g_main_context_find_source_by_id ()
g_main_context_find_source_by_user_data, g_main_context_find_source_by_user_data ()
g_main_context_get_poll_func, g_main_context_get_poll_func ()
g_main_context_iteration, g_main_context_iteration ()
g_main_context_new, g_main_context_new ()
g_main_context_pending, g_main_context_pending ()
g_main_context_prepare, g_main_context_prepare ()
g_main_context_query, g_main_context_query ()
g_main_context_ref, g_main_context_ref ()
g_main_context_release, g_main_context_release ()
g_main_context_remove_poll, g_main_context_remove_poll ()
g_main_context_set_poll_func, g_main_context_set_poll_func ()
g_main_context_unref, g_main_context_unref ()
g_main_context_wait, g_main_context_wait ()
g_main_context_wakeup, g_main_context_wakeup ()
g_main_depth, g_main_depth ()
g_main_destroy, g_main_destroy()
g_main_is_running, g_main_is_running()
g_main_iteration, g_main_iteration()
g_main_loop_get_context, g_main_loop_get_context ()
g_main_loop_is_running, g_main_loop_is_running ()
g_main_loop_new, g_main_loop_new ()
g_main_loop_quit, g_main_loop_quit ()
g_main_loop_ref, g_main_loop_ref ()
g_main_loop_run, g_main_loop_run ()
g_main_loop_unref, g_main_loop_unref ()

g_main_new, g_main_new()
g_main_pending, g_main_pending()
g_main_quit, g_main_quit()
g_main_run, g_main_run()
g_main_set_poll_func, g_main_set_poll_func()
g_malloc, g_malloc ()
g_malloc0, g_malloc0 ()
G_MARKUP_ERROR, G_MARKUP_ERROR
g_markup_escape_text, g_markup_escape_text ()
g_markup_parse_context_end_parse, g_markup_parse_context_end_parse ()
g_markup_parse_context_free, g_markup_parse_context_free ()
g_markup_parse_context_get_element, g_markup_parse_context_get_element ()
g_markup_parse_context_get_position, g_markup_parse_context_get_position ()
g_markup_parse_context_new, g_markup_parse_context_new ()
g_markup_parse_context_parse, g_markup_parse_context_parse ()
g_markup_printf_escaped, g_markup_printf_escaped ()
g_markup_vprintf_escaped, g_markup_vprintf_escaped ()
G_MAXDOUBLE, G_MAXDOUBLE
G_MAXFLOAT, G_MAXFLOAT
G_MAXINT, G_MAXINT
G_MAXINT16, G_MAXINT16
G_MAXINT32, G_MAXINT32
G_MAXINT64, G_MAXINT64
G_MAXINT8, G_MAXINT8
G_MAXLONG, G_MAXLONG
G_MAXSHORT, G_MAXSHORT
G_MAXSIZE, G_MAXSIZE
G_MAXUINT, G_MAXUINT
G_MAXUINT16, G_MAXUINT16
G_MAXUINT32, G_MAXUINT32
G_MAXUINT64, G_MAXUINT64
G_MAXUINT8, G_MAXUINT8
G_MAXULONG, G_MAXULONG
G_MAXUSHORT, G_MAXUSHORT
g_memdup, g_memdup ()
g_memmove, g_memmove()
G_MEM_ALIGN, G_MEM_ALIGN
g_mem_chunk_alloc, g_mem_chunk_alloc ()
g_mem_chunk_alloc0, g_mem_chunk_alloc0 ()
g_mem_chunk_clean, g_mem_chunk_clean ()
g_mem_chunk_create, g_mem_chunk_create()
g_mem_chunk_destroy, g_mem_chunk_destroy ()
g_mem_chunk_free, g_mem_chunk_free ()
g_mem_chunk_info, g_mem_chunk_info ()
g_mem_chunk_new, g_mem_chunk_new ()
g_mem_chunk_print, g_mem_chunk_print ()
g_mem_chunk_reset, g_mem_chunk_reset ()
g_mem_is_system_malloc, g_mem_is_system_malloc ()
g_mem_profile, g_mem_profile ()
g_mem_set_vtable, g_mem_set_vtable ()
g_message, g_message()
G_MINDOUBLE, G_MINDOUBLE
G_MINFLOAT, G_MINFLOAT
G_MININT, G_MININT
G_MININT16, G_MININT16

G_MININT32, G_MININT32
G_MININT64, G_MININT64
G_MININT8, G_MININT8
G_MINLONG, G_MINLONG
G_MINSHORT, G_MINSHORT
g_mkdir, g_mkdir ()
g_mkstemp, g_mkstemp ()
g_module_build_path, g_module_build_path ()
g_module_close, g_module_close ()
g_module_error, g_module_error ()
G_MODULE_EXPORT, G_MODULE_EXPORT
G_MODULE_IMPORT, G_MODULE_IMPORT
g_module_make_resident, g_module_make_resident ()
g_module_name, g_module_name ()
g_module_open, g_module_open ()
G_MODULE_SUFFIX, G_MODULE_SUFFIX
g_module_supported, g_module_supported ()
g_module_symbol, g_module_symbol ()
g_mutex_free, g_mutex_free ()
g_mutex_lock, g_mutex_lock ()
g_mutex_new, g_mutex_new ()
g_mutex_trylock, g_mutex_trylock ()
g_mutex_unlock, g_mutex_unlock ()
g_new, g_new()
g_new0, g_new0()
g_newa, g_newa()
g_node_append, g_node_append()
g_node_append_data, g_node_append_data()
g_node_children_foreach, g_node_children_foreach ()
g_node_child_index, g_node_child_index ()
g_node_child_position, g_node_child_position ()
g_node_copy, g_node_copy ()
g_node_copy_deep, g_node_copy_deep ()
g_node_depth, g_node_depth ()
g_node_destroy, g_node_destroy ()
g_node_find, g_node_find ()
g_node_find_child, g_node_find_child ()
g_node_first_child, g_node_first_child()
g_node_first_sibling, g_node_first_sibling ()
g_node_get_root, g_node_get_root ()
g_node_insert, g_node_insert ()
g_node_insert_after, g_node_insert_after ()
g_node_insert_before, g_node_insert_before ()
g_node_insert_data, g_node_insert_data()
g_node_insert_data_before, g_node_insert_data_before()
g_node_is_ancestor, g_node_is_ancestor ()
G_NODE_IS_LEAF, G_NODE_IS_LEAF()
G_NODE_IS_ROOT, G_NODE_IS_ROOT()
g_node_last_child, g_node_last_child ()
g_node_last_sibling, g_node_last_sibling ()
g_node_max_height, g_node_max_height ()
g_node_new, g_node_new ()
g_node_next_sibling, g_node_next_sibling()
g_node_nth_child, g_node_nth_child ()
g_node_n_children, g_node_n_children ()

g_node_n_nodes, g_node_n_nodes ()
g_node_pop_allocator, g_node_pop_allocator ()
g_node_prepend, g_node_prepend ()
g_node_prepend_data, g_node_prepend_data()
g_node_prev_sibling, g_node_prev_sibling()
g_node_push_allocator, g_node_push_allocator ()
g_node_reverse_children, g_node_reverse_children ()
g_node_traverse, g_node_traverse ()
g_node_unlink, g_node_unlink ()
g_ntohl, g_ntohl()
g_ntohs, g_ntohs()
g_nullify_pointer, g_nullify_pointer ()
G_N_ELEMENTS, G_N_ELEMENTS()
g_once, g_once()
G_ONCE_INIT, G_ONCE_INIT
g_on_error_query, g_on_error_query ()
g_on_error_stack_trace, g_on_error_stack_trace ()
g_open, g_open ()
g_option_context_add_group, g_option_context_add_group ()
g_option_context_add_main_entries, g_option_context_add_main_entries ()
g_option_context_free, g_option_context_free ()
g_option_context_get_help_enabled, g_option_context_get_help_enabled ()
g_option_context_get_ignore_unknown_options, g_option_context_get_ignore_unknown_options ()
g_option_context_get_main_group, g_option_context_get_main_group ()
g_option_context_new, g_option_context_new ()
g_option_context_parse, g_option_context_parse ()
g_option_context_set_help_enabled, g_option_context_set_help_enabled ()
g_option_context_set_ignore_unknown_options, g_option_context_set_ignore_unknown_options ()
g_option_context_set_main_group, g_option_context_set_main_group ()
G_OPTION_ERROR, G_OPTION_ERROR
g_option_group_add_entries, g_option_group_add_entries ()
g_option_group_free, g_option_group_free ()
g_option_group_new, g_option_group_new ()
g_option_group_set_error_hook, g_option_group_set_error_hook ()
g_option_group_set_parse_hooks, g_option_group_set_parse_hooks ()
g_option_group_set_translate_func, g_option_group_set_translate_func ()
g_option_group_set_translation_domain, g_option_group_set_translation_domain ()
G_OPTION_REMAINING, G_OPTION_REMAINING
G_OS_BEOS, G_OS_BEOS
G_OS_UNIX, G_OS_UNIX
G_OS_WIN32, G_OS_WIN32
g_parse_debug_string, g_parse_debug_string ()
g_path_get_basename, g_path_get_basename ()
g_path_get_dirname, g_path_get_dirname ()
g_path_is_absolute, g_path_is_absolute ()
g_path_skip_root, g_path_skip_root ()
g_pattern_match, g_pattern_match ()
g_pattern_match_simple, g_pattern_match_simple ()
g_pattern_match_string, g_pattern_match_string ()
g_pattern_spec_equal, g_pattern_spec_equal ()
g_pattern_spec_free, g_pattern_spec_free ()
g_pattern_spec_new, g_pattern_spec_new ()
G_PDP_ENDIAN, G_PDP_ENDIAN
G_PI, G_PI
G_PI_2, G_PI_2

G_PI_4, G_PI_4
g_print, g_print ()
g_printerr, g_printerr ()
g_printf, g_printf ()
g_printf_string_upper_bound, g_printf_string_upper_bound ()
G_PRIORITY_DEFAULT, G_PRIORITY_DEFAULT
G_PRIORITY_DEFAULT_IDLE, G_PRIORITY_DEFAULT_IDLE
G_PRIORITY_HIGH, G_PRIORITY_HIGH
G_PRIORITY_HIGH_IDLE, G_PRIORITY_HIGH_IDLE
G_PRIORITY_LOW, G_PRIORITY_LOW
g_private_get, g_private_get ()
g_private_new, g_private_new ()
g_private_set, g_private_set ()
g_propagate_error, g_propagate_error ()
g_ptr_array_add, g_ptr_array_add ()
g_ptr_array_foreach, g_ptr_array_foreach ()
g_ptr_array_free, g_ptr_array_free ()
g_ptr_array_index, g_ptr_array_index()
g_ptr_array_new, g_ptr_array_new ()
g_ptr_array_remove, g_ptr_array_remove ()
g_ptr_array_remove_fast, g_ptr_array_remove_fast ()
g_ptr_array_remove_index, g_ptr_array_remove_index ()
g_ptr_array_remove_index_fast, g_ptr_array_remove_index_fast ()
g_ptr_array_remove_range, g_ptr_array_remove_range ()
g_ptr_array_set_size, g_ptr_array_set_size ()
g_ptr_array_sized_new, g_ptr_array_sized_new ()
g_ptr_array_sort, g_ptr_array_sort ()
g_ptr_array_sort_with_data, g_ptr_array_sort_with_data ()
g_qsort_with_data, g_qsort_with_data ()
g_quark_from_static_string, g_quark_from_static_string ()
g_quark_from_string, g_quark_from_string ()
g_quark_to_string, g_quark_to_string ()
g_quark_try_string, g_quark_try_string ()
g_queue_copy, g_queue_copy ()
g_queue_delete_link, g_queue_delete_link ()
g_queue_find, g_queue_find ()
g_queue_find_custom, g_queue_find_custom ()
g_queue_foreach, g_queue_foreach ()
g_queue_free, g_queue_free ()
g_queue_get_length, g_queue_get_length ()
g_queue_index, g_queue_index ()
g_queue_insert_after, g_queue_insert_after ()
g_queue_insert_before, g_queue_insert_before ()
g_queue_insert_sorted, g_queue_insert_sorted ()
g_queue_is_empty, g_queue_is_empty ()
g_queue_link_index, g_queue_link_index ()
g_queue_new, g_queue_new ()
g_queue_peek_head, g_queue_peek_head ()
g_queue_peek_head_link, g_queue_peek_head_link ()
g_queue_peek_nth, g_queue_peek_nth ()
g_queue_peek_nth_link, g_queue_peek_nth_link ()
g_queue_peek_tail, g_queue_peek_tail ()
g_queue_peek_tail_link, g_queue_peek_tail_link ()
g_queue_pop_head, g_queue_pop_head ()
g_queue_pop_head_link, g_queue_pop_head_link ()

g_queue_pop_nth, g_queue_pop_nth ()
g_queue_pop_nth_link, g_queue_pop_nth_link ()
g_queue_pop_tail, g_queue_pop_tail ()
g_queue_pop_tail_link, g_queue_pop_tail_link ()
g_queue_push_head, g_queue_push_head ()
g_queue_push_head_link, g_queue_push_head_link ()
g_queue_push_nth, g_queue_push_nth ()
g_queue_push_nth_link, g_queue_push_nth_link ()
g_queue_push_tail, g_queue_push_tail ()
g_queue_push_tail_link, g_queue_push_tail_link ()
g_queue_remove, g_queue_remove ()
g_queue_remove_all, g_queue_remove_all ()
g_queue_reverse, g_queue_reverse ()
g_queue_sort, g_queue_sort ()
g_queue_unlink, g_queue_unlink ()
g_random_boolean, g_random_boolean()
g_random_double, g_random_double ()
g_random_double_range, g_random_double_range ()
g_random_int, g_random_int ()
g_random_int_range, g_random_int_range ()
g_random_set_seed, g_random_set_seed ()
g_rand_boolean, g_rand_boolean()
g_rand_copy, g_rand_copy ()
g_rand_double, g_rand_double ()
g_rand_double_range, g_rand_double_range ()
g_rand_free, g_rand_free ()
g_rand_int, g_rand_int ()
g_rand_int_range, g_rand_int_range ()
g_rand_new, g_rand_new ()
g_rand_new_with_seed, g_rand_new_with_seed ()
g_rand_new_with_seed_array, g_rand_new_with_seed_array ()
g_rand_set_seed, g_rand_set_seed ()
g_rand_set_seed_array, g_rand_set_seed_array ()
g_realloc, g_realloc ()
g_relation_count, g_relation_count ()
g_relation_delete, g_relation_delete ()
g_relation_destroy, g_relation_destroy ()
g_relation_exists, g_relation_exists ()
g_relation_index, g_relation_index ()
g_relation_insert, g_relation_insert ()
g_relation_new, g_relation_new ()
g_relation_print, g_relation_print ()
g_relation_select, g_relation_select ()
g_remove, g_remove ()
g_rename, g_rename ()
g_renew, g_renew()
g_return_if_fail, g_return_if_fail()
g_return_if_reached, g_return_if_reached()
g_return_val_if_fail, g_return_val_if_fail()
g_return_val_if_reached, g_return_val_if_reached()
g_scanner_add_symbol, g_scanner_add_symbol()
g_scanner_cur_line, g_scanner_cur_line ()
g_scanner_cur_position, g_scanner_cur_position ()
g_scanner_cur_token, g_scanner_cur_token ()
g_scanner_cur_value, g_scanner_cur_value ()

g_scanner_destroy, g_scanner_destroy ()
g_scanner_eof, g_scanner_eof ()
g_scanner_error, g_scanner_error ()
g_scanner_foreach_symbol, g_scanner_foreach_symbol()
g_scanner_freeze_symbol_table, g_scanner_freeze_symbol_table()
g_scanner_get_next_token, g_scanner_get_next_token ()
g_scanner_input_file, g_scanner_input_file ()
g_scanner_input_text, g_scanner_input_text ()
g_scanner_lookup_symbol, g_scanner_lookup_symbol ()
g_scanner_new, g_scanner_new ()
g_scanner_peek_next_token, g_scanner_peek_next_token ()
g_scanner_remove_symbol, g_scanner_remove_symbol()
g_scanner_scope_add_symbol, g_scanner_scope_add_symbol ()
g_scanner_scope_foreach_symbol, g_scanner_scope_foreach_symbol ()
g_scanner_scope_lookup_symbol, g_scanner_scope_lookup_symbol ()
g_scanner_scope_remove_symbol, g_scanner_scope_remove_symbol ()
g_scanner_set_scope, g_scanner_set_scope ()
g_scanner_sync_file_offset, g_scanner_sync_file_offset ()
g_scanner_thaw_symbol_table, g_scanner_thaw_symbol_table()
g_scanner_unexp_token, g_scanner_unexp_token ()
g_scanner_warn, g_scanner_warn ()
G_SEARCHPATH_SEPARATOR, G_SEARCHPATH_SEPARATOR
G_SEARCHPATH_SEPARATOR_S, G_SEARCHPATH_SEPARATOR_S
g_setenv, g_setenv ()
g_set_application_name, g_set_application_name ()
g_set_error, g_set_error ()
g_set_prgname, g_set_prgname ()
g_set_printerr_handler, g_set_printerr_handler ()
g_set_print_handler, g_set_print_handler ()
G_SHELL_ERROR, G_SHELL_ERROR
g_shell_parse_argv, g_shell_parse_argv ()
g_shell_quote, g_shell_quote ()
g_shell_unquote, g_shell_unquote ()
g_slist_alloc, g_slist_alloc ()
g_slist_append, g_slist_append ()
g_slist_concat, g_slist_concat ()
g_slist_copy, g_slist_copy ()
g_slist_delete_link, g_slist_delete_link ()
g_slist_find, g_slist_find ()
g_slist_find_custom, g_slist_find_custom ()
g_slist_foreach, g_slist_foreach ()
g_slist_free, g_slist_free ()
g_slist_free_1, g_slist_free_1 ()
g_slist_index, g_slist_index ()
g_slist_insert, g_slist_insert ()
g_slist_insert_before, g_slist_insert_before ()
g_slist_insert_sorted, g_slist_insert_sorted ()
g_slist_last, g_slist_last ()
g_slist_length, g_slist_length ()
g_slist_next, g_slist_next()
g_slist_nth, g_slist_nth ()
g_slist_nth_data, g_slist_nth_data ()
g_slist_pop_allocator, g_slist_pop_allocator ()
g_slist_position, g_slist_position ()
g_slist_prepend, g_slist_prepend ()

g_slist_push_allocator, g_slist_push_allocator ()
g_slist_remove, g_slist_remove ()
g_slist_remove_all, g_slist_remove_all ()
g_slist_remove_link, g_slist_remove_link ()
g_slist_reverse, g_slist_reverse ()
g_slist_sort, g_slist_sort ()
g_slist_sort_with_data, g_slist_sort_with_data ()
g_snprintf, g_snprintf ()
g_source_add_poll, g_source_add_poll ()
g_source_attach, g_source_attach ()
g_source_destroy, g_source_destroy ()
g_source_get_can_recurse, g_source_get_can_recurse ()
g_source_get_context, g_source_get_context ()
g_source_get_current_time, g_source_get_current_time ()
g_source_get_id, g_source_get_id ()
g_source_get_priority, g_source_get_priority ()
g_source_new, g_source_new ()
g_source_ref, g_source_ref ()
g_source_remove, g_source_remove ()
g_source_remove_by_funcs_user_data, g_source_remove_by_funcs_user_data ()
g_source_remove_by_user_data, g_source_remove_by_user_data ()
g_source_remove_poll, g_source_remove_poll ()
g_source_set_callback, g_source_set_callback ()
g_source_set_callback_indirect, g_source_set_callback_indirect ()
g_source_set_can_recurse, g_source_set_can_recurse ()
g_source_set_priority, g_source_set_priority ()
g_source_unref, g_source_unref ()
g_spaced_primes_closest, g_spaced_primes_closest ()
g_spawn_async, g_spawn_async ()
g_spawn_async_with_pipes, g_spawn_async_with_pipes ()
g_spawn_close_pid, g_spawn_close_pid ()
g_spawn_command_line_async, g_spawn_command_line_async ()
g_spawn_command_line_sync, g_spawn_command_line_sync ()
G_SPAWN_ERROR, G_SPAWN_ERROR
g_spawn_sync, g_spawn_sync ()
g_sprintf, g_sprintf ()
G_SQRT2, G_SQRT2
g_stat, g_stat ()
g_static_mutex_free, g_static_mutex_free ()
g_static_mutex_get_mutex, g_static_mutex_get_mutex ()
G_STATIC_MUTEX_INIT, G_STATIC_MUTEX_INIT
g_static_mutex_init, g_static_mutex_init ()
g_static_mutex_lock, g_static_mutex_lock ()
g_static_mutex_trylock, g_static_mutex_trylock ()
g_static_mutex_unlock, g_static_mutex_unlock ()
g_static_private_free, g_static_private_free ()
g_static_private_get, g_static_private_get ()
G_STATIC_PRIVATE_INIT, G_STATIC_PRIVATE_INIT
g_static_private_init, g_static_private_init ()
g_static_private_set, g_static_private_set ()
g_static_rec_mutex_free, g_static_rec_mutex_free ()
G_STATIC_REC_MUTEX_INIT, G_STATIC_REC_MUTEX_INIT
g_static_rec_mutex_init, g_static_rec_mutex_init ()
g_static_rec_mutex_lock, g_static_rec_mutex_lock ()
g_static_rec_mutex_lock_full, g_static_rec_mutex_lock_full ()

g_static_rec_mutex_trylock, g_static_rec_mutex_trylock ()
g_static_rec_mutex_unlock, g_static_rec_mutex_unlock ()
g_static_rec_mutex_unlock_full, g_static_rec_mutex_unlock_full ()
g_static_rw_lock_free, g_static_rw_lock_free ()
G_STATIC_RW_LOCK_INIT, G_STATIC_RW_LOCK_INIT
g_static_rw_lock_init, g_static_rw_lock_init ()
g_static_rw_lock_reader_lock, g_static_rw_lock_reader_lock ()
g_static_rw_lock_reader_trylock, g_static_rw_lock_reader_trylock ()
g_static_rw_lock_reader_unlock, g_static_rw_lock_reader_unlock ()
g_static_rw_lock_writer_lock, g_static_rw_lock_writer_lock ()
g_static_rw_lock_writer_trylock, g_static_rw_lock_writer_trylock ()
g_static_rw_lock_writer_unlock, g_static_rw_lock_writer_unlock ()
G_STMT_END, G_STMT_END
G_STMT_START, G_STMT_START
g_stpcpy, g_stpcpy ()
g_strcanon, g_strcanon ()
g_strcasecmp, g_strcasecmp ()
g_strchomp, g_strchomp ()
g_strchug, g_strchug ()
g_strcompress, g_strcompress ()
g_strconcat, g_strconcat ()
g_strdelimit, g_strdelimit ()
g_strdown, g_strdown ()
g_strdup, g_strdup ()
g_strdupv, g_strdupv ()
g_strdup_printf, g_strdup_printf ()
g_strdup_vprintf, g_strdup_vprintf ()
g_strerror, g_strerror ()
g_strescape, g_strescape ()
g_strfreev, g_strfreev ()
G_STRFUNC, G_STRFUNC
G_STRINGIFY, G_STRINGIFY()
g_string_append, g_string_append ()
g_string_append_c, g_string_append_c ()
g_string_append_len, g_string_append_len ()
g_string_append_printf, g_string_append_printf ()
g_string_append_unichar, g_string_append_unichar ()
g_string_ascii_down, g_string_ascii_down ()
g_string_ascii_up, g_string_ascii_up ()
g_string_assign, g_string_assign ()
g_string_chunk_free, g_string_chunk_free ()
g_string_chunk_insert, g_string_chunk_insert ()
g_string_chunk_insert_const, g_string_chunk_insert_const ()
g_string_chunk_insert_len, g_string_chunk_insert_len ()
g_string_chunk_new, g_string_chunk_new ()
g_string_down, g_string_down ()
g_string_equal, g_string_equal ()
g_string_erase, g_string_erase ()
g_string_free, g_string_free ()
g_string_hash, g_string_hash ()
g_string_insert, g_string_insert ()
g_string_insert_c, g_string_insert_c ()
g_string_insert_len, g_string_insert_len ()
g_string_insert_unichar, g_string_insert_unichar ()
g_string_new, g_string_new ()

g_string_new_len, g_string_new_len ()
g_string_prepend, g_string_prepend ()
g_string_prepend_c, g_string_prepend_c ()
g_string_prepend_len, g_string_prepend_len ()
g_string_prepend_unichar, g_string_prepend_unichar ()
g_string_printf, g_string_printf ()
g_string_set_size, g_string_set_size ()
g_string_sized_new, g_string_sized_new ()
g_string_sprintf, g_string_sprintf
g_string_sprintfa, g_string_sprintfa
g_string_truncate, g_string_truncate ()
g_string_up, g_string_up ()
g_strip_context, g_strip_context ()
g_strjoin, g_strjoin ()
g_strjoinv, g_strjoinv ()
g_strlcat, g_strlcat ()
g_strlcpy, g_strlcpy ()
G_STRLOC, G_STRLOC
g_strncasecmp, g_strncasecmp ()
g_strndup, g_strndup ()
g_strnfill, g_strnfill ()
g_strreverse, g_strreverse ()
g_strrstr, g_strrstr ()
g_strrstr_len, g_strrstr_len ()
g_strsignal, g_strsignal ()
g_strsplit, g_strsplit ()
g_strsplit_set, g_strsplit_set ()
g_strstrip, g_strstrip()
g_strstr_len, g_strstr_len ()
g_strtod, g_strtod ()
G_STRUCT_MEMBER, G_STRUCT_MEMBER()
G_STRUCT_MEMBER_P, G_STRUCT_MEMBER_P()
G_STRUCT_OFFSET, G_STRUCT_OFFSET()
g_strup, g_strup ()
g_strv_length, g_strv_length ()
G_STR_DELIMITERS, G_STR_DELIMITERS
g_str_equal, g_str_equal ()
g_str_hash, g_str_hash ()
g_str_has_prefix, g_str_has_prefix ()
g_str_has_suffix, g_str_has_suffix ()
G_THREADS_ENABLED, G_THREADS_ENABLED
G_THREADS_IMPL_NONE, G_THREADS_IMPL_NONE
G_THREADS_IMPL_POSIX, G_THREADS_IMPL_POSIX
G_THREADS_IMPL_SOLARIS, G_THREADS_IMPL_SOLARIS
g_thread_create, g_thread_create ()
g_thread_create_full, g_thread_create_full ()
G_THREAD_ERROR, G_THREAD_ERROR
g_thread_exit, g_thread_exit ()
g_thread_init, g_thread_init ()
g_thread_join, g_thread_join ()
g_thread_pool_free, g_thread_pool_free ()
g_thread_pool_get_max_threads, g_thread_pool_get_max_threads ()
g_thread_pool_get_max_unused_threads, g_thread_pool_get_max_unused_threads ()
g_thread_pool_get_num_threads, g_thread_pool_get_num_threads ()
g_thread_pool_get_num_unused_threads, g_thread_pool_get_num_unused_threads ()

g_thread_pool_new, g_thread_pool_new ()
g_thread_pool_push, g_thread_pool_push ()
g_thread_pool_set_max_threads, g_thread_pool_set_max_threads ()
g_thread_pool_set_max_unused_threads, g_thread_pool_set_max_unused_threads ()
g_thread_pool_stop_unused_threads, g_thread_pool_stop_unused_threads ()
g_thread_pool_unprocessed, g_thread_pool_unprocessed ()
g_thread_self, g_thread_self ()
g_thread_set_priority, g_thread_set_priority ()
g_thread_supported, g_thread_supported ()
g_thread_yield, g_thread_yield ()
g_timeout_add, g_timeout_add ()
g_timeout_add_full, g_timeout_add_full ()
g_timeout_source_new, g_timeout_source_new ()
g_timer_continue, g_timer_continue ()
g_timer_destroy, g_timer_destroy ()
g_timer_elapsed, g_timer_elapsed ()
g_timer_new, g_timer_new ()
g_timer_reset, g_timer_reset ()
g_timer_start, g_timer_start ()
g_timer_stop, g_timer_stop ()
g_time_val_add, g_time_val_add ()
g_trash_stack_height, g_trash_stack_height ()
g_trash_stack_peek, g_trash_stack_peek ()
g_trash_stack_pop, g_trash_stack_pop ()
g_trash_stack_push, g_trash_stack_push ()
g_tree_destroy, g_tree_destroy ()
g_tree_foreach, g_tree_foreach ()
g_tree_height, g_tree_height ()
g_tree_insert, g_tree_insert ()
g_tree_lookup, g_tree_lookup ()
g_tree_lookup_extended, g_tree_lookup_extended ()
g_tree_new, g_tree_new ()
g_tree_new_full, g_tree_new_full ()
g_tree_new_with_data, g_tree_new_with_data ()
g_tree_nnodes, g_tree_nnodes ()
g_tree_remove, g_tree_remove ()
g_tree_replace, g_tree_replace ()
g_tree_search, g_tree_search ()
g_tree_steal, g_tree_steal ()
g_tree_traverse, g_tree_traverse ()
G_TRYLOCK, G_TRYLOCK()
g_try_malloc, g_try_malloc ()
g_try_realloc, g_try_realloc ()
g_tuples_destroy, g_tuples_destroy ()
g_tuples_index, g_tuples_index ()
g_ucs4_to_utf16, g_ucs4_to_utf16 ()
g_ucs4_to_utf8, g_ucs4_to_utf8 ()
g_unichar_break_type, g_unichar_break_type ()
g_unichar_digit_value, g_unichar_digit_value ()
g_unichar_get_mirror_char, g_unichar_get_mirror_char ()
g_unichar_isalnum, g_unichar_isalnum ()
g_unichar_isalpha, g_unichar_isalpha ()
g_unichar_iscntrl, g_unichar_iscntrl ()
g_unichar_isdefined, g_unichar_isdefined ()
g_unichar_isdigit, g_unichar_isdigit ()

g_unichar_isgraph, g_unichar_isgraph ()
g_unichar_islower, g_unichar_islower ()
g_unichar_isprint, g_unichar_isprint ()
g_unichar_ispunct, g_unichar_ispunct ()
g_unichar_isspace, g_unichar_isspace ()
g_unichar_istitle, g_unichar_istitle ()
g_unichar_isupper, g_unichar_isupper ()
g_unichar_iswide, g_unichar_iswide ()
g_unichar_isxdigit, g_unichar_isxdigit ()
g_unichar_tolower, g_unichar_tolower ()
g_unichar_totitle, g_unichar_totitle ()
g_unichar_toupper, g_unichar_toupper ()
g_unichar_to_utf8, g_unichar_to_utf8 ()
g_unichar_type, g_unichar_type ()
g_unichar_validate, g_unichar_validate ()
g_unichar_xdigit_value, g_unichar_xdigit_value ()
g_unicode_canonical_decomposition, g_unicode_canonical_decomposition ()
g_unicode_canonical_ordering, g_unicode_canonical_ordering ()
G_UNLIKELY, G_UNLIKELY()
g_unlink, g_unlink ()
G_UNLOCK, G_UNLOCK()
g_unsetenv, g_unsetenv ()
g_uri_list_extract_uris, g_uri_list_extract_uris ()
G_USEC_PER_SEC, G_USEC_PER_SEC
g_usleep, g_usleep ()
g_utf16_to_ucs4, g_utf16_to_ucs4 ()
g_utf16_to_utf8, g_utf16_to_utf8 ()
g_utf8_casefold, g_utf8_casefold ()
g_utf8_collate, g_utf8_collate ()
g_utf8_collate_key, g_utf8_collate_key ()
g_utf8_find_next_char, g_utf8_find_next_char ()
g_utf8_find_prev_char, g_utf8_find_prev_char ()
g_utf8_get_char, g_utf8_get_char ()
g_utf8_get_char_validated, g_utf8_get_char_validated ()
g_utf8_next_char, g_utf8_next_char()
g_utf8_normalize, g_utf8_normalize ()
g_utf8_offset_to_pointer, g_utf8_offset_to_pointer ()
g_utf8_pointer_to_offset, g_utf8_pointer_to_offset ()
g_utf8_prev_char, g_utf8_prev_char ()
g_utf8_strchr, g_utf8_strchr ()
g_utf8_strdown, g_utf8_strdown ()
g_utf8_strlen, g_utf8_strlen ()
g_utf8_strncpy, g_utf8_strncpy ()
g_utf8_strrchr, g_utf8_strrchr ()
g_utf8_strreverse, g_utf8_strreverse ()
g_utf8_strup, g_utf8_strup ()
g_utf8_to_ucs4, g_utf8_to_ucs4 ()
g_utf8_to_ucs4_fast, g_utf8_to_ucs4_fast ()
g_utf8_to_utf16, g_utf8_to_utf16 ()
g_utf8_validate, g_utf8_validate ()
g_vasprintf, g_vasprintf ()
G_VA_COPY, G_VA_COPY
g_vfprintf, g_vfprintf ()
g_vprintf, g_vprintf ()
g_vsnprintf, g_vsnprintf ()

g_vsprintf, g_vsprintf ()
g_warning, g_warning()
G_WIN32_DLLMAIN_FOR_DLL_NAME, G_WIN32_DLLMAIN_FOR_DLL_NAME()
g_win32_error_message, g_win32_error_message ()
g_win32_getlocale, g_win32_getlocale ()
g_win32_get_package_installation_directory, g_win32_get_package_installation_directory ()
g_win32_get_package_installation_subdirectory, g_win32_get_package_installation_subdirectory ()
g_win32_get_windows_version, g_win32_get_windows_version ()
G_WIN32_HAVE_WIDECHAR_API, G_WIN32_HAVE_WIDECHAR_API()
G_WIN32_IS_NT_BASED, G_WIN32_IS_NT_BASED()

M

MAX, MAX()
MAXPATHLEN, MAXPATHLEN
MIN, MIN()

N

NULL, NULL
N_, N_()

P

pid_t, pid_t
pipe, pipe()

Q

Q_, Q_()

T

TRUE, TRUE

<< glib-gettextize **Index of deprecated symbols >>**

Index of deprecated symbols

Symbols

G

g_async_queue_ref_unlocked, g_async_queue_ref_unlocked ()
g_async_queue_unref_and_unlock, g_async_queue_unref_and_unlock ()
g_basename, g_basename ()
g_dirname, g_dirname
g_hash_table_freeze, g_hash_table_freeze()
g_hash_table_thaw, g_hash_table_thaw()
g_io_channel_close, g_io_channel_close ()
g_io_channel_read, g_io_channel_read ()
g_io_channel_seek, g_io_channel_seek ()
g_io_channel_write, g_io_channel_write ()
g_main_destroy, g_main_destroy()
g_main_is_running, g_main_is_running()
g_main_iteration, g_main_iteration()
g_main_new, g_main_new()
g_main_pending, g_main_pending()
g_main_quit, g_main_quit()
g_main_run, g_main_run()
g_main_set_poll_func, g_main_set_poll_func()
g_scanner_add_symbol, g_scanner_add_symbol()
g_scanner_foreach_symbol, g_scanner_foreach_symbol()
g_scanner_freeze_symbol_table, g_scanner_freeze_symbol_table()
g_scanner_remove_symbol, g_scanner_remove_symbol()
g_scanner_thaw_symbol_table, g_scanner_thaw_symbol_table()
g_strcasecmp, g_strcasecmp ()
g_strdown, g_strdown ()
g_string_down, g_string_down ()
g_string_sprintf, g_string_sprintf
g_string_sprintfa, g_string_sprintfa
g_string_up, g_string_up ()
g_strncasecmp, g_strncasecmp ()
g_strup, g_strup ()
g_tree_traverse, g_tree_traverse ()

<< Index **Index of new symbols in 2.2 >>**

Index of new symbols in 2.2

Symbols

G

g_ascii_strtoull, g_ascii_strtoull ()
g_fprintf, g_fprintf ()
g_get_application_name, g_get_application_name ()
G_GNUC_DEPRECATED, G_GNUC_DEPRECATED
G_LIKELY, G_LIKELY()
g_markup_parse_context_get_element, g_markup_parse_context_get_element ()
g_printf, g_printf ()
g_sprintf, g_sprintf ()
g_str_has_prefix, g_str_has_prefix ()
g_str_has_suffix, g_str_has_suffix ()
G_UNLIKELY, G_UNLIKELY()
g_utf8_strreverse, g_utf8_strreverse ()
g_vfprintf, g_vfprintf ()
g_vprintf, g_vprintf ()
g_vsprintf, g_vsprintf ()

<< Index of deprecated symbols **Index of new symbols in 2.4 >>**

GLib Reference Manual

Index of new symbols in 2.4

Symbols

_, _()

G

GCopyFunc, GCopyFunc ()
GOnce, GOnce
GOnceStatus, enum GOnceStatus
g_array_remove_range, g_array_remove_range ()
g_atomic_int_add, g_atomic_int_add ()
g_atomic_int_compare_and_exchange, g_atomic_int_compare_and_exchange ()
g_atomic_int_dec_and_test, g_atomic_int_dec_and_test ()
g_atomic_int_exchange_and_add, g_atomic_int_exchange_and_add ()
g_atomic_int_get, g_atomic_int_get ()
g_atomic_int_inc, g_atomic_int_inc ()
g_atomic_pointer_compare_and_exchange, g_atomic_pointer_compare_and_exchange ()
g_atomic_pointer_get, g_atomic_pointer_get ()
g_byte_array_remove_range, g_byte_array_remove_range ()
g_child_watch_add, g_child_watch_add ()
g_child_watch_add_full, g_child_watch_add_full ()
g_child_watch_source_new, g_child_watch_source_new ()
g_completion_complete_utf8, g_completion_complete_utf8 ()
g_file_read_link, g_file_read_link ()
G_GINT16_MODIFIER, G_GINT16_MODIFIER
G_GINT32_MODIFIER, G_GINT32_MODIFIER
G_GINT64_MODIFIER, G_GINT64_MODIFIER
g_hash_table_find, g_hash_table_find ()
g_markup_printf_escaped, g_markup_printf_escaped ()
g_markup_vprintf_escaped, g_markup_vprintf_escaped ()
G_MAXINT16, G_MAXINT16
G_MAXINT32, G_MAXINT32
G_MAXINT8, G_MAXINT8
G_MAXSIZE, G_MAXSIZE
G_MAXUINT16, G_MAXUINT16
G_MAXUINT32, G_MAXUINT32
G_MAXUINT8, G_MAXUINT8
G_MININT16, G_MININT16
G_MININT32, G_MININT32
G_MININT8, G_MININT8
g_node_copy_deep, g_node_copy_deep ()
g_once, g_once()
G_ONCE_INIT, G_ONCE_INIT
g_ptr_array_foreach, g_ptr_array_foreach ()
g_ptr_array_remove_range, g_ptr_array_remove_range ()
g_queue_copy, g_queue_copy ()
g_queue_delete_link, g_queue_delete_link ()
g_queue_find, g_queue_find ()

g_queue_find_custom, g_queue_find_custom ()
g_queue_foreach, g_queue_foreach ()
g_queue_get_length, g_queue_get_length ()
g_queue_index, g_queue_index ()
g_queue_insert_after, g_queue_insert_after ()
g_queue_insert_before, g_queue_insert_before ()
g_queue_insert_sorted, g_queue_insert_sorted ()
g_queue_link_index, g_queue_link_index ()
g_queue_peek_head_link, g_queue_peek_head_link ()
g_queue_peek_nth, g_queue_peek_nth ()
g_queue_peek_nth_link, g_queue_peek_nth_link ()
g_queue_peek_tail_link, g_queue_peek_tail_link ()
g_queue_pop_nth, g_queue_pop_nth ()
g_queue_pop_nth_link, g_queue_pop_nth_link ()
g_queue_push_nth, g_queue_push_nth ()
g_queue_push_nth_link, g_queue_push_nth_link ()
g_queue_remove, g_queue_remove ()
g_queue_remove_all, g_queue_remove_all ()
g_queue_reverse, g_queue_reverse ()
g_queue_sort, g_queue_sort ()
g_queue_unlink, g_queue_unlink ()
g_rand_copy, g_rand_copy ()
g_rand_new_with_seed_array, g_rand_new_with_seed_array ()
g_rand_set_seed_array, g_rand_set_seed_array ()
g_setenv, g_setenv ()
G_STRFUNC, G_STRFUNC
g_string_chunk_insert_len, g_string_chunk_insert_len ()
g_strip_context, g_strip_context ()
g_strsplit_set, g_strsplit_set ()
g_timer_continue, g_timer_continue ()
g_unichar_get_mirror_char, g_unichar_get_mirror_char ()
g_unsetenv, g_unsetenv ()
g_vasprintf, g_vasprintf ()

N

N_, N_()

Q

Q_, Q_()

<< Index of new symbols in 2.2 Index of new symbols in 2.6 >>

GLib Reference Manual

Index of new symbols in 2.6

Symbols

G

glib_check_version, glib_check_version ()
g_date_get_iso8601_week_of_year, g_date_get_iso8601_week_of_year ()
g_debug, g_debug()
g_filename_display_name, g_filename_display_name ()
g_fopen, g_fopen ()
g_freopen, g_freopen ()
g_get_filename_charsets, g_get_filename_charsets ()
g_get_language_names, g_get_language_names ()
g_get_system_config_dirs, g_get_system_config_dirs ()
g_get_system_data_dirs, g_get_system_data_dirs ()
g_get_user_cache_dir, g_get_user_cache_dir ()
g_get_user_config_dir, g_get_user_config_dir ()
g_get_user_data_dir, g_get_user_data_dir ()
G_GSIZE_FORMAT, G_GSIZE_FORMAT
G_GSIZE_MODIFIER, G_GSIZE_MODIFIER
G_GSSIZE_FORMAT, G_GSSIZE_FORMAT
G_IS_DIR_SEPARATOR, G_IS_DIR_SEPARATOR()
g_key_file_free, g_key_file_free ()
g_key_file_get_boolean, g_key_file_get_boolean ()
g_key_file_get_boolean_list, g_key_file_get_boolean_list ()
g_key_file_get_comment, g_key_file_get_comment ()
g_key_file_get_groups, g_key_file_get_groups ()
g_key_file_get_integer, g_key_file_get_integer ()
g_key_file_get_integer_list, g_key_file_get_integer_list ()
g_key_file_get_keys, g_key_file_get_keys ()
g_key_file_get_locale_string, g_key_file_get_locale_string ()
g_key_file_get_locale_string_list, g_key_file_get_locale_string_list ()
g_key_file_get_start_group, g_key_file_get_start_group ()
g_key_file_get_string, g_key_file_get_string ()
g_key_file_get_string_list, g_key_file_get_string_list ()
g_key_file_get_value, g_key_file_get_value ()
g_key_file_has_group, g_key_file_has_group ()
g_key_file_has_key, g_key_file_has_key ()
g_key_file_load_from_data, g_key_file_load_from_data ()
g_key_file_load_from_data_dirs, g_key_file_load_from_data_dirs ()
g_key_file_load_from_file, g_key_file_load_from_file ()
g_key_file_new, g_key_file_new ()
g_key_file_remove_comment, g_key_file_remove_comment ()
g_key_file_remove_group, g_key_file_remove_group ()
g_key_file_remove_key, g_key_file_remove_key ()
g_key_file_set_boolean, g_key_file_set_boolean ()
g_key_file_set_boolean_list, g_key_file_set_boolean_list ()
g_key_file_set_comment, g_key_file_set_comment ()
g_key_file_set_integer, g_key_file_set_integer ()

g_key_file_set_integer_list, g_key_file_set_integer_list ()
g_key_file_set_list_separator, g_key_file_set_list_separator ()
g_key_file_set_locale_string, g_key_file_set_locale_string ()
g_key_file_set_locale_string_list, g_key_file_set_locale_string_list ()
g_key_file_set_string, g_key_file_set_string ()
g_key_file_set_string_list, g_key_file_set_string_list ()
g_key_file_set_value, g_key_file_set_value ()
g_key_file_to_data, g_key_file_to_data ()
g_log_set_default_handler, g_log_set_default_handler ()
g_lstat, g_lstat ()
g_mkdir, g_mkdir ()
g_open, g_open ()
g_option_context_add_group, g_option_context_add_group ()
g_option_context_add_main_entries, g_option_context_add_main_entries ()
g_option_context_free, g_option_context_free ()
g_option_context_get_help_enabled, g_option_context_get_help_enabled ()
g_option_context_get_ignore_unknown_options, g_option_context_get_ignore_unknown_options ()
g_option_context_get_main_group, g_option_context_get_main_group ()
g_option_context_new, g_option_context_new ()
g_option_context_parse, g_option_context_parse ()
g_option_context_set_help_enabled, g_option_context_set_help_enabled ()
g_option_context_set_ignore_unknown_options, g_option_context_set_ignore_unknown_options ()
g_option_context_set_main_group, g_option_context_set_main_group ()
g_option_group_add_entries, g_option_group_add_entries ()
g_option_group_free, g_option_group_free ()
g_option_group_new, g_option_group_new ()
g_option_group_set_error_hook, g_option_group_set_error_hook ()
g_option_group_set_parse_hooks, g_option_group_set_parse_hooks ()
g_option_group_set_translate_func, g_option_group_set_translate_func ()
g_option_group_set_translation_domain, g_option_group_set_translation_domain ()
G_OPTION_REMAINING, G_OPTION_REMAINING
g_remove, g_remove ()
g_rename, g_rename ()
g_stat, g_stat ()
g_strv_length, g_strv_length ()
g_unlink, g_unlink ()
g_uri_list_extract_uris, g_uri_list_extract_uris ()
g_win32_get_windows_version, g_win32_get_windows_version ()
G_WIN32_HAVE_WIDECHAR_API, G_WIN32_HAVE_WIDECHAR_API()
G_WIN32_IS_NT_BASED, G_WIN32_IS_NT_BASED()

<< Index of new symbols in 2.4