Guile-GNOME: GLib

version 2.15.93, updated 2 September 2007

Owen Taylor
Matthias Clasen
Andy Wingo
(many others)

This manual is for `(gnome glib)` (version 2.15.93, updated 2 September 2007)

Short Contents

1 Base64 Encoding: encodes and decodes data in Base64 format 1
2 Bookmark file parser: parses files containing bookmarks 2
3 Character Set Conversion: convert strings between different character sets using 16
4 File Utilities: various file-related functions. 21
5 IO Channels: portable support for using files, pipes and sockets. 24
6 The Main Event Loop: manages all available sources of events. 34
7 Miscellaneous Utility Functions: a selection of portable utility functions. 45
8 Quarks: a 2-way association between a string and a unique integer identifier. 50
9 Shell-related Utilities: shell-like commandline handling. 51
10 Strings: text buffers which grow automatically as text is added. 52
11 Unicode Manipulation: functions operating on Unicode characters and UTF-8 strings. 53
Concept Index 66
Function Index 67

1 Base64 Encoding: encodes and decodes data in Base64 format

1.1 Overview

Base64 is an encoding that allows to encode a sequence of arbitrary bytes as a sequence of printable ASCII characters. For the definition of Base64, see RFC 1421 or RFC 2045. Base64 is most commonly used as a MIME transfer encoding for email.

GLib supports incremental encoding using `g-base64-encode-step` and `g-base64-encode-close`. Incremental decoding can be done with `g-base64-decode-step`. To encode or decode data in one go, use `g-base64-encode` or `g-base64-decode`.

Support for Base64 encoding has been added in GLib 2.12.

1.2 Usage

`g-base64-encode` (*data* `<guchar*>`) (*len* `size_t`) ⇒ (*ret* `mchars`) [Function]

Encode a sequence of binary data into its Base-64 stringified representation.

data	the binary data to encode
len	the length of *data*
ret	a newly allocated, zero-terminated Base-64 encoded string representing *data*.

Since 2.12

`g-base64-decode` (*text* `mchars`) ⇒ (*ret* `<guchar*>`) (*out_len* `size_t`) [Function]

Decode a sequence of Base-64 encoded text into binary data

text	zero-terminated string with base64 text to decode
out-len	The length of the decoded data is written here
ret	a newly allocated buffer containing the binary data that *text* represents

Since 2.12

2 Bookmark file parser: parses files containing bookmarks

2.1 Overview

`<g-bookmark-file>` lets you parse, edit or create files containing bookmarks to URI, along with some meta-data about the resource pointed by the URI like its MIME type, the application that is registering the bookmark and the icon that should be used to represent the bookmark. The data is stored using the Desktop Bookmark Specification.

The syntax of the bookmark files is described in detail inside the Desktop Bookmark Specification, here is a quick summary: bookmark files use a sub-class of the XML Bookmark Exchange Language specification, consisting of valid UTF-8 encoded XML, under the '`xbel`' root element; each bookmark is stored inside a '`bookmark`' element, using its URI: no relative paths can be used inside a bookmark file. The bookmark may have a user defined title and description, to be used instead of the URI. Under the '`metadata`' element, with its '`owner`' attribute set to '`http://freedesktop.org`', is stored the meta-data about a resource pointed by its URI. The meta-data consists of the resource's MIME type; the applications that have registered a bookmark; the groups to which a bookmark belongs to; a visibility flag, used to set the bookmark as "private" to the applications and groups that has it registered; the URI and MIME type of an icon, to be used when displaying the bookmark inside a GUI.

```
<?xml version="1.0"?>
<!DOCTYPE xbel PUBLIC
  "+//IDN python.org//DTD XML Bookmark Exchange Language 1.0//EN//XML"
  "http://www.python.org/topics/xml/dtds/xbel-1.0.dtd">
<xbel version="1.0"
      xmlns:mime="http://www.freedesktop.org/standards/shared-mime-info"
      xmlns:bookmark="http://www.freedesktop.org/standards/desktop-bookmarks">
  <bookmark href="file:///home/ebassi/bookmark-spec/bookmark-spec.xml">
    <title>Desktop Bookmarks Spec</title>
    <info>
      <metadata owner="http://freedesktop.org">
        <mime:mime-type>text/xml</mime:mime-type>
<bookmark:applications>
          <bookmark:application name="GEdit" count="2" exec="gedit %u" timestamp="1115
          <bookmark:application name="GViM" count="7" exec="gvim %f" timestamp="111572
</bookmark:applications>
<bookmark:groups>
  <bookmark:group>Editors</bookmark:group>
</bookmark:groups>
      </metadata>
    </info>
  </bookmark>
</xbel>
```

A bookmark file might contain more than one bookmark; each bookmark is accessed through its URI.

The important caveat of bookmark files is that when you add a new bookmark you must also add the application that is registering it, using `g-bookmark-file-add-application` or `g-bookmark-file-set-app-info`. If a bookmark has no applications then it won't be dumped when creating the on disk representation, using `g-bookmark-file-to-data` or `g-bookmark-file-to-file`.

The `<g-bookmark-file>` parser was added in GLib 2.12.

2.2 Usage

`g-bookmark-file-new` ⇒ (*ret* `<g-bookmark-file*>`) [Function]

Creates a new empty `<g-bookmark-file>` object.

Use `g-bookmark-file-load-from-file`, `g-bookmark-file-load-from-data` or `g-bookmark-file-load-from-data-dirs` to read an existing bookmark file.

ret	an empty `<g-bookmark-file>`

Since 2.12

`g-bookmark-file-free` (*bookmark* `<g-bookmark-file*>`) [Function]

Frees a `<g-bookmark-file>`.

bookmark	a `<g-bookmark-file>`

Since 2.12

`g-bookmark-file-load-from-file` (*bookmark* `<g-bookmark-file*>`) (*filename* `mchars`) ⇒ (*ret* `bool`) [Function]

Loads a desktop bookmark file into an empty `<g-bookmark-file>` structure. If the file could not be loaded then *error* is set to either a `<g-file-error>` or `<g-bookmark-file-error>`.

bookmark	an empty `<g-bookmark-file>` struct
filename	the path of a filename to load, in the GLib file name encoding
error	return location for a `<g-error>`, or '`#f`'
ret	'`TRUE`' if a desktop bookmark file could be loaded

Since 2.12

`g-bookmark-file-load-from-data` (*bookmark* `<g-bookmark-file*>`) (*data* `mchars`) (*length* `size_t`) ⇒ (*ret* `bool`) [Function]

Loads a bookmark file from memory into an empty `<g-bookmark-file>` structure. If the object cannot be created then *error* is set to a `<g-bookmark-file-error>`.

bookmark	an empty `<g-bookmark-file>` struct
data	desktop bookmarks loaded in memory
length	the length of *data* in bytes

error	return location for a `<g-error>`, or '`#f`'
ret	'`TRUE`' if a desktop bookmark could be loaded.

Since 2.12

`g-bookmark-file-load-from-data-dirs` (*bookmark* `<g-bookmark-file*>`) (*file* `mchars`) ⇒ (*ret* `bool`) (*full_path* `mchars`) [Function]

This function looks for a desktop bookmark file named *file* in the paths returned from `g-get-user-data-dir` and `g-get-system-data-dirs`, loads the file into *bookmark* 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 `<g-file-error>` or `<g-bookmark-file-error>`.

bookmark	a `<g-bookmark-file>`
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 '`#f`'
error	return location for a `<g-error>`, or '`#f`'
ret	'`TRUE`' if a key file could be loaded, '`FALSE`' othewise

Since 2.12

`g-bookmark-file-to-data` (*bookmark* `<g-bookmark-file*>`) ⇒ (*ret* `mchars`) (*length* `size_t`) [Function]

This function outputs *bookmark* as a string.

bookmark	a `<g-bookmark-file>`
length	return location for the length of the returned string, or '`#f`'
error	return location for a `<g-error>`, or '`#f`'
ret	a newly allocated string holding the contents of the `<g-bookmark-file>`

Since 2.12

`g-bookmark-file-to-file` (*bookmark* `<g-bookmark-file*>`) (*filename* `mchars`) ⇒ (*ret* `bool`) [Function]

This function outputs *bookmark* into a file. The write process is guaranteed to be atomic by using `g-file-set-contents` internally.

bookmark	a `<g-bookmark-file>`
filename	path of the output file
error	return location for a `<g-error>`, or '`#f`'
ret	'`TRUE`' if the file was successfully written.

Since 2.12

`g-bookmark-file-has-item` (*bookmark* `<g-bookmark-file*>`) (*uri* `mchars`) ⇒ (*ret* `bool`) [Function]

Looks whether the desktop bookmark has an item with its URI set to *uri*.

bookmark	a `<g-bookmark-file>`
uri	a valid URI
ret	'`TRUE`' if *uri* is inside *bookmark*, '`FALSE`' otherwise

Since 2.12

`g-bookmark-file-has-group` (*bookmark* `<g-bookmark-file*>`) (*uri* `mchars`) (*group* `mchars`) ⇒ (*ret* `bool`) [Function]

Checks whether *group* appears in the list of groups to which the bookmark for *uri* belongs to.

In the event the URI cannot be found, '`FALSE`' is returned and *error* is set to `<g-bookmark-file-error-uri-not-found>`.

bookmark	a `<g-bookmark-file>`
uri	a valid URI
group	the group name to be searched
error	return location for a `<g-error>`, or '`#f`'
ret	'`TRUE`' if *group* was found.

Since 2.12

`g-bookmark-file-has-application` (*bookmark* `<g-bookmark-file*>`) (*uri* `mchars`) (*name* `mchars`) ⇒ (*ret* `bool`) [Function]

Checks whether the bookmark for *uri* inside *bookmark* has been registered by application *name*.

In the event the URI cannot be found, '`FALSE`' is returned and *error* is set to `<g-bookmark-file-error-uri-not-found>`.

bookmark	a `<g-bookmark-file>`
uri	a valid URI
name	the name of the application
error	return location for a `<g-error>` or '`#f`'
ret	'`TRUE`' if the application *name* was found

Since 2.12

`g-bookmark-file-get-size` (*bookmark* `<g-bookmark-file*>`) ⇒ (*ret* `int`) [Function]

Gets the number of bookmarks inside *bookmark*.

bookmark	a `<g-bookmark-file>`
ret	the number of bookmarks

Since 2.12

`g-bookmark-file-get-uris` (*bookmark* `<g-bookmark-file*>`) ⇒ (*ret* `<gchar**>`) (*length* `size_t`) [Function]

Returns all URIs of the bookmarks in the bookmark file *bookmark*. The array of returned URIs will be '`#f`'-terminated, so *length* may optionally be '`#f`'.

bookmark	a `<g-bookmark-file>`
length	return location for the number of returned URIs, or '`#f`'
ret	a newly allocated '`#f`'-terminated array of strings. Use `g-strfreev` to free it.

Since 2.12

`g-bookmark-file-get-title` (*bookmark* `<g-bookmark-file*>`) (*uri* `mchars`) ⇒ (*ret* `mchars`) [Function]

Returns the title of the bookmark for *uri*.

If *uri* is '`#f`', the title of *bookmark* is returned.

In the event the URI cannot be found, '`#f`' is returned and *error* is set to `<g-bookmark-file-error-uri-not-found>`.

bookmark	a `<g-bookmark-file>`
uri	a valid URI or '`#f`'
error	return location for a `<g-error>`, or '`#f`'
ret	a newly allocated string or '`#f`' if the specified URI cannot be found.

Since 2.12

`g-bookmark-file-get-description` (*bookmark* `<g-bookmark-file*>`) (*uri* `mchars`) ⇒ (*ret* `mchars`) [Function]

Retrieves the description of the bookmark for *uri*.

In the event the URI cannot be found, '`#f`' is returned and *error* is set to `<g-bookmark-file-error-uri-not-found>`.

bookmark	a `<g-bookmark-file>`
uri	a valid URI
error	return location for a `<g-error>`, or '`#f`'
ret	a newly allocated string or '`#f`' if the specified URI cannot be found.

Since 2.12

`g-bookmark-file-get-mime-type` (*bookmark* `<g-bookmark-file*>`) (*uri* `mchars`) ⇒ (*ret* `mchars`) [Function]

Retrieves the MIME type of the resource pointed by *uri*.

In the event the URI cannot be found, '`#f`' is returned and *error* is set to `<g-bookmark-file-error-uri-not-found>`. In the event that the MIME type cannot be found, '`#f`' is returned and *error* is set to `<g-bookmark-file-error-invalid-value>`.

bookmark	a `<g-bookmark-file>`
uri	a valid URI
error	return location for a `<g-error>`, or '`#f`'
ret	a newly allocated string or '`#f`' if the specified URI cannot be found.

Since 2.12

`g-bookmark-file-get-is-private ( `*`bookmark`*` <g-bookmark-file*> ) ( `*`uri`*` mchars ) ⇒ ( `*`ret`*` bool )` [Function]

Gets whether the private flag of the bookmark for *uri* is set.

In the event the URI cannot be found, '`FALSE`' is returned and *error* is set to `<g-bookmark-file-error-uri-not-found>`. In the event that the private flag cannot be found, '`FALSE`' is returned and *error* is set to `<g-bookmark-file-error-invalid-value>`.

bookmark	a `<g-bookmark-file>`
uri	a valid URI
error	return location for a `<g-error>`, or '`#f`'
ret	'`TRUE`' if the private flag is set, '`FALSE`' otherwise.

Since 2.12

`g-bookmark-file-get-icon ( `*`bookmark`*` <g-bookmark-file*> ) ( `*`uri`*` mchars ) ⇒ ( `*`ret`*` bool ) ( `*`href`*` mchars ) ( `*`mime_type`*` mchars )` [Function]

Gets the icon of the bookmark for *uri*.

In the event the URI cannot be found, '`FALSE`' is returned and *error* is set to `<g-bookmark-file-error-uri-not-found>`.

bookmark	a `<g-bookmark-file>`
uri	a valid URI
href	return location for the icon's location or '`#f`'
mime-type	return location for the icon's MIME type or '`#f`'
error	return location for a `<g-error>` or '`#f`'
ret	'`TRUE`' if the icon for the bookmark for the URI was found. You should free the returned strings.

Since 2.12

`g-bookmark-file-get-added ( `*`bookmark`*` <g-bookmark-file*> ) ( `*`uri`*` mchars ) ⇒ ( `*`ret`*` long )` [Function]

Gets the time the bookmark for *uri* was added to *bookmark*

In the event the URI cannot be found, -1 is returned and *error* is set to `<g-bookmark-file-error-uri-not-found>`.

bookmark	a `<g-bookmark-file>`

uri	a valid URI
error	return location for a `<g-error>`, or '`#f`'
ret	a timestamp

Since 2.12

`g-bookmark-file-get-modified` (*bookmark* `<g-bookmark-file*>`) (*uri* `mchars`) ⇒ (*ret* `long`) [Function]

Gets the time when the bookmark for *uri* was last modified.

In the event the URI cannot be found, -1 is returned and *error* is set to `<g-bookmark-file-error-uri-not-found>`.

bookmark	a `<g-bookmark-file>`
uri	a valid URI
error	return location for a `<g-error>`, or '`#f`'
ret	a timestamp

Since 2.12

`g-bookmark-file-get-visited` (*bookmark* `<g-bookmark-file*>`) (*uri* `mchars`) ⇒ (*ret* `long`) [Function]

Gets the time the bookmark for *uri* was last visited.

In the event the URI cannot be found, -1 is returned and *error* is set to `<g-bookmark-file-error-uri-not-found>`.

bookmark	a `<g-bookmark-file>`
uri	a valid URI
error	return location for a `<g-error>`, or '`#f`'
ret	a timestamp.

Since 2.12

`g-bookmark-file-get-groups` (*bookmark* `<g-bookmark-file*>`) (*uri* `mchars`) ⇒ (*ret* `<gchar**>`) (*length* `size_t`) [Function]

Retrieves the list of group names of the bookmark for *uri*.

In the event the URI cannot be found, '`#f`' is returned and *error* is set to `<g-bookmark-file-error-uri-not-found>`.

The returned array is '`#f`' terminated, so *length* may optionally be '`#f`'.

bookmark	a `<g-bookmark-file>`
uri	a valid URI
length	return location for the length of the returned string, or '`#f`'
error	return location for a `<g-error>`, or '`#f`'
ret	a newly allocated '`#f`'-terminated array of group names. Use `g-strfreev` to free it.

Since 2.12

`g-bookmark-file-get-applications` (*bookmark* `<g-bookmark-file*>`) (*uri* `mchars`) ⇒ (*ret* `<gchar**>`) (*length* `size_t`) [Function]

Retrieves the names of the applications that have registered the bookmark for *uri*.

In the event the URI cannot be found, '`#f`' is returned and *error* is set to `<g-bookmark-file-error-uri-not-found>`.

bookmark	a `<g-bookmark-file>`
uri	a valid URI
length	return location of the length of the returned list, or '`#f`'
error	return location for a `<g-error>`, or '`#f`'
ret	a newly allocated '`#f`'-terminated array of strings. Use `g-strfreev` to free it.

Since 2.12

`g-bookmark-file-get-app-info` (*bookmark* `<g-bookmark-file*>`) (*uri* `mchars`) (*name* `mchars`) ⇒ (*ret* `bool`) (*exec* `mchars`) (*count* `unsigned-int`) (*stamp* `long`) [Function]

Gets the registration informations of *app-name* for the bookmark for *uri*. See `g-bookmark-file-set-app-info` for more informations about the returned data.

The string returned in *app-exec* must be freed.

In the event the URI cannot be found, '`FALSE`' is returned and *error* is set to `<g-bookmark-file-error-uri-not-found>`. In the event that no application with name *app-name* has registered a bookmark for *uri*, '`FALSE`' is returned and error is set to `<g-bookmark-file-error-app-not-registered>`.

bookmark	a `<g-bookmark-file>`
uri	a valid URI
name	an application's name
exec	location for the command line of the application, or '`#f`'
count	return location for the registration count, or '`#f`'
stamp	return location for the last registration time, or '`#f`'
error	return location for a `<g-error>`, or '`#f`'
ret	'`TRUE`' on success.

Since 2.12

`g-bookmark-file-set-title` (*bookmark* `<g-bookmark-file*>`) (*uri* `mchars`) (*title* `mchars`) [Function]

Sets *title* as the title of the bookmark for *uri* inside the bookmark file *bookmark*.

If *uri* is '`#f`', the title of *bookmark* is set.

If a bookmark for *uri* cannot be found then it is created.

bookmark	a `<g-bookmark-file>`

uri	a valid URI or '`#f`'
title	a UTF-8 encoded string

Since 2.12

`g-bookmark-file-set-description` (*bookmark* `<g-bookmark-file*>`) (*uri* `mchars`) (*description* `mchars`) [Function]

Sets *description* as the description of the bookmark for *uri*.

If *uri* is '`#f`', the description of *bookmark* is set.

If a bookmark for *uri* cannot be found then it is created.

bookmark	a `<g-bookmark-file>`
uri	a valid URI or '`#f`'
description	a string

Since 2.12

`g-bookmark-file-set-mime-type` (*bookmark* `<g-bookmark-file*>`) (*uri* `mchars`) (*mime_type* `mchars`) [Function]

Sets *mime-type* as the MIME type of the bookmark for *uri*.

If a bookmark for *uri* cannot be found then it is created.

bookmark	a `<g-bookmark-file>`
uri	a valid URI
mime-type	a MIME type

Since 2.12

`g-bookmark-file-set-is-private` (*bookmark* `<g-bookmark-file*>`) (*uri* `mchars`) (*is_private* `bool`) [Function]

Sets the private flag of the bookmark for *uri*.

If a bookmark for *uri* cannot be found then it is created.

bookmark	a `<g-bookmark-file>`
uri	a valid URI
is-private	'`TRUE`' if the bookmark should be marked as private

Since 2.12

`g-bookmark-file-set-icon` (*bookmark* `<g-bookmark-file*>`) (*uri* `mchars`) (*href* `mchars`) (*mime_type* `mchars`) [Function]

Sets the icon for the bookmark for *uri*. If *href* is '`#f`', unsets the currently set icon.

If no bookmark for *uri* is found it is created.

bookmark	a `<g-bookmark-file>`
uri	a valid URI

href	the URI of the icon for the bookmark, or '`#f`'
mime-type	the MIME type of the icon for the bookmark

Since 2.12

`g-bookmark-file-set-added` (*bookmark* `<g-bookmark-file*>`) (*uri* `mchars`) (*added* `long`) [Function]

Sets the time the bookmark for *uri* was added into *bookmark*.

If no bookmark for *uri* is found then it is created.

bookmark	a `<g-bookmark-file>`
uri	a valid URI
added	a timestamp or -1 to use the current time

Since 2.12

`g-bookmark-file-set-groups` (*bookmark* `<g-bookmark-file*>`) (*uri* `mchars`) (*groups* `<gchar**>`) (*length* `size_t`) [Function]

Sets a list of group names for the item with URI *uri*. Each previously set group name list is removed.

If *uri* cannot be found then an item for it is created.

bookmark	a `<g-bookmark-file>`
uri	an item's URI
groups	an array of group names, or '`#f`' to remove all groups
length	number of group name values in *groups*

Since 2.12

`g-bookmark-file-set-modified` (*bookmark* `<g-bookmark-file*>`) (*uri* `mchars`) (*modified* `long`) [Function]

Sets the last time the bookmark for *uri* was last modified.

If no bookmark for *uri* is found then it is created.

The `"modified"` time should only be set when the bookmark's meta-data was actually changed. Every function of `<g-bookmark-file>` that modifies a bookmark also changes the modification time, except for `g-bookmark-file-set-visited`.

bookmark	a `<g-bookmark-file>`
uri	a valid URI
modified	a timestamp or -1 to use the current time

Since 2.12

`g-bookmark-file-set-visited` (*bookmark* `<g-bookmark-file*>`) (*uri* `mchars`) (*visited* `long`) [Function]

Sets the time the bookmark for *uri* was last visited.

If no bookmark for *uri* is found then it is created.

The "visited" time should only be set if the bookmark was launched, either using the command line retrieved by `g-bookmark-file-get-app-info` or by the default application for the bookmark's MIME type, retrieved using `g-bookmark-file-get-mime-type`. Changing the "visited" time does not affect the "modified" time.

bookmark	a `<g-bookmark-file>`
uri	a valid URI
visited	a timestamp or -1 to use the current time

Since 2.12

`g-bookmark-file-set-app-info` (*bookmark* `<g-bookmark-file*>`) (*uri* `mchars`) (*name* `mchars`) (*exec* `mchars`) (*count* `int`) (*stamp* `long`) ⇒ (*ret* `bool`) [Function]

Sets the meta-data of application *name* inside the list of applications that have registered a bookmark for *uri* inside *bookmark*.

You should rarely use this function; use `g-bookmark-file-add-application` and `g-bookmark-file-remove-application` instead.

name can be any UTF-8 encoded string used to identify an application. *exec* can have one of these two modifiers: "‘`f`’", which will be expanded as the local file name retrieved from the bookmark's URI; "‘`u`’", which will be expanded as the bookmark's URI. The expansion is done automatically when retrieving the stored command line using the `g-bookmark-file-get-app-info` function. *count* is the number of times the application has registered the bookmark; if is `<` 0, the current registration count will be increased by one, if is 0, the application with *name* will be removed from the list of registered applications. *stamp* is the Unix time of the last registration; if it is -1, the current time will be used.

If you try to remove an application by setting its registration count to zero, and no bookmark for *uri* is found, ‘`FALSE`’ is returned and *error* is set to `<g-bookmark-file-error-uri-not-found>`; similarly, in the event that no application *name* has registered a bookmark for *uri*, ‘`FALSE`’ is returned and error is set to `<g-bookmark-file-error-app-not-registered>`. Otherwise, if no bookmark for *uri* is found, one is created.

bookmark	a `<g-bookmark-file>`
uri	a valid URI
name	an application's name
exec	an application's command line
count	the number of registrations done for this application
stamp	the time of the last registration for this application

error	return location for a `<g-error>` or '`#f`'
ret	'`TRUE`' if the application's meta-data was successfully changed.

Since 2.12

`g-bookmark-file-add-group` (*bookmark* `<g-bookmark-file*>`) (*uri* `mchars`) (*group* `mchars`) [Function]

Adds *group* to the list of groups to which the bookmark for *uri* belongs to.

If no bookmark for *uri* is found then it is created.

bookmark	a `<g-bookmark-file>`
uri	a valid URI
group	the group name to be added

Since 2.12

`g-bookmark-file-add-application` (*bookmark* `<g-bookmark-file*>`) (*uri* `mchars`) (*name* `mchars`) (*exec* `mchars`) [Function]

Adds the application with *name* and *exec* to the list of applications that have registered a bookmark for *uri* into *bookmark*.

Every bookmark inside a `<g-bookmark-file>` must have at least an application registered. Each application must provide a name, a command line useful for launching the bookmark, the number of times the bookmark has been registered by the application and the last time the application registered this bookmark.

If *name* is '`#f`', the name of the application will be the same returned by `g-get-application`; if *exec* is '`#f`', the command line will be a composition of the program name as returned by `g-get-prgname` and the `"'u'"` modifier, which will be expanded to the bookmark's URI.

This function will automatically take care of updating the registrations count and timestamping in case an application with the same *name* had already registered a bookmark for *uri* inside *bookmark*.

If no bookmark for *uri* is found, one is created.

bookmark	a `<g-bookmark-file>`
uri	a valid URI
name	the name of the application registering the bookmark or '`#f`'
exec	command line to be used to launch the bookmark or '`#f`'

Since 2.12

`g-bookmark-file-remove-group` (*bookmark* `<g-bookmark-file*>`) (*uri* `mchars`) (*group* `mchars`) ⇒ (*ret* `bool`) [Function]

Removes *group* from the list of groups to which the bookmark for *uri* belongs to.

In the event the URI cannot be found, '`FALSE`' is returned and *error* is set to `<g-bookmark-file-error-uri-not-found>`. In the event no group was defined, '`FALSE`' is returned and *error* is set to `<g-bookmark-file-error-invalid-value>`.

bookmark	a `<g-bookmark-file>`
uri	a valid URI
group	the group name to be removed
error	return location for a `<g-error>`, or '`#f`'
ret	'`TRUE`' if *group* was successfully removed.

Since 2.12

`g-bookmark-file-remove-application` (*bookmark* `<g-bookmark-file*>`) (*uri* `mchars`) (*name* `mchars`) ⇒ (*ret* `bool`) [Function]

Removes application registered with *name* from the list of applications that have registered a bookmark for *uri* inside *bookmark*.

In the event the URI cannot be found, '`FALSE`' is returned and *error* is set to `<g-bookmark-file-error-uri-not-found>`. In the event that no application with name *app-name* has registered a bookmark for *uri*, '`FALSE`' is returned and error is set to `<g-bookmark-file-error-app-not-registered>`.

bookmark	a `<g-bookmark-file>`
uri	a valid URI
name	the name of the application
error	return location for a `<g-error>` or '`#f`'
ret	'`TRUE`' if the application was successfully removed.

Since 2.12

`g-bookmark-file-remove-item` (*bookmark* `<g-bookmark-file*>`) (*uri* `mchars`) ⇒ (*ret* `bool`) [Function]

Removes the bookmark for *uri* from the bookmark file *bookmark*.

bookmark	a `<g-bookmark-file>`
uri	a valid URI
error	return location for a `<g-error>`, or '`#f`'
ret	'`TRUE`' if the bookmark was removed successfully.

Since 2.12

`g-bookmark-file-move-item` (*bookmark* `<g-bookmark-file*>`) (*old_uri* `mchars`) (*new_uri* `mchars`) ⇒ (*ret* `bool`) [Function]

Changes the URI of a bookmark item from *old-uri* to *new-uri*. Any existing bookmark for *new-uri* will be overwritten. If *new-uri* is '`#f`', then the bookmark is removed.

In the event the URI cannot be found, '`FALSE`' is returned and *error* is set to `<g-bookmark-file-error-uri-not-found>`.

bookmark	a `<g-bookmark-file>`
old-uri	a valid URI

new-uri	a valid URI, or '`#f`'
error	return location for a `<g-error>` or '`#f`'
ret	'`TRUE`' if the URI was successfully changed

Since 2.12

3 Character Set Conversion: convert strings between different character sets using .

3.1 Overview

3.2 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 "'Presentacin.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     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        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. *(the missing section, file-name-encodings-diagram* illustrates how these functions are used to convert between UTF-8 and the encoding for file names in the file system.

3.2.1 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.
2.
3.

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)` &#x2014; those are "raw" file names which the file system understands.

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.

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.

3.3 Usage

`g-locale-to-utf8` (*opsysstring* `mchars`) (*len* `ssize_t`) ⇒ (*ret* `mchars`) (*bytes_read* `size_t`) (*bytes_written* `size_t`) [Function]

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. On Windows this means the system codepage.

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 ‘`#f`’. 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 '`#f`' to ignore errors. Any of the errors in `<g-convert-error>` may occur.

ret The converted string, or '`#f`' on an error.

`g-filename-to-utf8` (***`opsysstring`*** `mchars`) (***`len`*** `ssize_t`) ⇒ (***`ret`*** `mchars`) (***`bytes_read`*** `size_t`) (***`bytes_written`*** `size_t`) [Function]

Converts a string which is in the encoding used by GLib for filenames into a UTF-8 string. Note that on Windows GLib uses UTF-8 for filenames.

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 '`#f`'. 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 '`#f`' to ignore errors. Any of the errors in `<g-convert-error>` may occur.

ret The converted string, or '`#f`' on an error.

`g-filename-from-utf8` (***`utf8string`*** `mchars`) (***`len`*** `ssize_t`) ⇒ (***`ret`*** `mchars`) (***`bytes_read`*** `size_t`) (***`bytes_written`*** `size_t`) [Function]

Converts a string from UTF-8 to the encoding GLib uses for filenames. Note that on Windows GLib uses UTF-8 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 '`#f`'. 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 '`#f`' to ignore errors. Any of the errors in `<g-convert-error>` may occur.

ret The converted string, or '`#f`' on an error.

`g-filename-from-uri` (*uri* `mchars`) ⇒ (*ret* `mchars`) (*hostname* `mchars`) [Function]

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 '`#f`'. If there is no hostname in the URI, '`#f`' will be stored in this location.

error location to store the error occuring, or '`#f`' to ignore errors. Any of the errors in `<g-convert-error>` may occur.

ret a newly-allocated string holding the resulting filename, or '`#f`' on an error.

`g-filename-to-uri` (*filename* `mchars`) (*hostname* `mchars`) ⇒ (*ret* `mchars`) [Function]

Converts an absolute filename to an escaped ASCII-encoded URI, with the path component following Section 3.3. of RFC 2396.

filename an absolute filename specified in the GLib file name encoding, which is the on-disk file name bytes on Unix, and UTF-8 on Windows

hostname A UTF-8 encoded hostname, or '`#f`' for none.

error location to store the error occuring, or '`#f`' to ignore errors. Any of the errors in `<g-convert-error>` may occur.

ret a newly-allocated string holding the resulting URI, or '`#f`' on an error.

`g-filename-display-name` (*filename* `mchars`) ⇒ (*ret* `mchars`) [Function]

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-'`#f`' even if the filename actually isn't in the GLib file name encoding.

If GLib can not make sense of the encoding of *filename*, as a last resort it replaces unknown characters with U+FFFD, the Unicode replacement character. You can search the result for the UTF-8 encoding of this character (which is `"\357\277\275"` in octal notation) to find out if *filename* was in an invalid encoding.

If you know the whole pathname of the file you should use `g-filename-display-basename`, since that allows location-based translation of filenames.

filename a pathname hopefully in the GLib file name encoding

ret a newly allocated string containing a rendition of the filename in valid UTF-8

Since 2.6

`g-filename-display-basename` (*filename* `mchars`) ⇒ (*ret* `mchars`) [Function]

Returns the display basename for the particular filename, guaranteed to be valid UTF-8. The display name might not be identical to the filename, for instance there might be problems converting it to UTF-8, and some files can be translated in the display.

If GLib can not make sense of the encoding of *filename*, as a last resort it replaces unknown characters with U+FFFD, the Unicode replacement character. You can search the result for the UTF-8 encoding of this character (which is `"\357\277\275"` in octal notation) to find out if *filename* was in an invalid encoding.

You must pass the whole absolute pathname to this functions so that translation of well known locations can be done.

This function is preferred over `g-filename-display-name` if you know the whole path, as it allows translation.

filename	an absolute pathname in the GLib file name encoding
ret	a newly allocated string containing a rendition of the basename of the filename in valid UTF-8

Since 2.6

`g-locale-from-utf8` (***utf8string*** `mchars`) (***len*** `ssize_t`) ⇒ (***ret*** `mchars`) (***bytes_read*** `size_t`) (***bytes_written*** `size_t`) [Function]

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 '`#f`'. 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 '`#f`' to ignore errors. Any of the errors in `<g-convert-error>` may occur.
ret	The converted string, or '`#f`' on an error.

4 File Utilities: various file-related functions.

4.1 Overview

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`.

4.2 Usage

`g-file-error-from-errno` (*err_no* `int`) ⇒ (*ret* `<g-file-error>`) [Function]

Gets a `<g-file-error>` 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 `<g-file-error>` values will exist.

Normally a `<g-file-error>` value goes into a `<g-error>` returned from a function that manipulates files. So you would use `g-file-error-from-errno` when constructing a `<g-error>`.

err-no	an "errno" value
ret	`<g-file-error>` corresponding to the given *errno*

`g-file-get-contents` (*filename* `mchars`) ⇒ (*ret* `bool`) (*contents* `mchars`) (*length* `size_t`) [Function]

Reads an entire file into allocated memory, with good error checking.

If the call was successful, it returns '`TRUE`' and sets *contents* to the file contents and *length* to the length of the file contents in bytes. The string stored in *contents* will be nul-terminated, so for text files you can pass '`#f`' for the *length* argument. If the call was not successful, it returns '`FALSE`' and sets *error*. The error domain is `<g-file-error>`. Possible error codes are those in the `<g-file-error>` enumeration. In the error case, *contents* is set to '`#f`' and *length* is set to zero.

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, or '`#f`'

error	return location for a `<g-error>`, or '`#f`'
ret	'`TRUE`' on success, '`FALSE`' if an error occurred

g-file-test (*filename* mchars) (*test* <g-file-test>) ⇒ (*ret* bool) [Function]

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 operation, 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!

```
/&#x002A; DON'T DO THIS &#x002A;/
 if (!g_file_test (filename, G_FILE_TEST_IS_SYMLINK)) {
   fd = g_open (filename, O_WRONLY);
   /&#x002A; write to fd &#x002A;/
 }
```

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.

On Windows, there are no symlinks, so testing for '`G_FILE_TEST_IS_SYMLINK`' will always return '`FALSE`'. Testing for '`G_FILE_TEST_IS_EXECUTABLE`' will just check that the file exists and its name indicates that it is executable, checking for well-known extensions and those listed in the '`PATHEXT`' environment variable.

filename	a filename to test in the GLib file name encoding
test	bitfield of `<g-file-test>` flags
ret	whether a test was '`TRUE`'

g-mkstemp (*tmpl* mchars) ⇒ (*ret* int) [Function]

Opens a temporary file. See the `mkstemp` documentation on most UNIX-like systems.

The parameter is a string that should follow the rules for `mkstemp` templates, i.e. contain the string "XXXXXX". `g-mkstemp` is slightly more flexible than `mkstemp` in that the sequence does not have to occur at the very end of the template. 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
ret	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` (*tmpl* `mchars`) ⇒ (*ret* `int`) (*name_used* `mchars`) [Function]

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 containing a sequence of six 'X' characters, as the parameter to `g-mkstemp`. However, unlike these functions, the template should only be a basename, no directory components are allowed. If template is '`#f`', 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-'`#f`'. 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, or '`#f`', to a default template
name-used	location to store actual name used
error	return location for a `<g-error>`
ret	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` (*filename* `mchars`) ⇒ (*ret* `mchars`) [Function]

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 `<g-error>`
ret	A newly allocated string with the contents of the symbolic link, or '`#f`' if an error occurred.

Since 2.4

5 IO Channels: portable support for using files, pipes and sockets.

5.1 Overview

The `<gio-channel>` 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 `<gio-channel>` 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 `<gio-channel>` 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-shutdown`.

To add a `<gio-channel>` 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 `<gio-channel>`, and provide a function to be called whenever these events occur.

`<gio-channel>` 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 `<gio-channel>` 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.

5.2 Usage

`g-io-channel-unix-new` (*fd* `int`) ⇒ (*ret* `<gio-channel*>`) [Function]

Creates a new `<gio-channel>` given a file descriptor. On UNIX systems this works for plain files, pipes, and sockets.

The returned `<gio-channel>` has a reference count of 1.

The default encoding for `<gio-channel>` 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-channel-set-encoding` function with '`#f`' for the encoding argument.

This function is available in GLib on Windows, too, but you should avoid using it on Windows. The domain of file descriptors and sockets overlap. There is no way for GLib to know which one you mean in case the argument you pass to this function happens to be both a valid file descriptor and socket. If that happens a warning is issued, and GLib assumes that it is the file descriptor you mean.

fd a file descriptor.

ret a new `<gio-channel>`.

`g-io-channel-unix-get-fd` (*channel* `<gio-channel*>`) ⇒ (*ret* `int`) [Function]

Returns the file descriptor of the `<gio-channel>`.

On Windows this function returns the file descriptor or socket of the `<gio-channel>`.

channel a `<gio-channel>`, created with `g-io-channel-unix-new`.

ret the file descriptor of the `<gio-channel>`.

`g-io-channel-init` (*channel* `<gio-channel*>`) [Function]

Initializes a `<gio-channel>` struct. This is called by each of the above functions when creating a `<gio-channel>`, and so is not often needed by the application programmer (unless you are creating a new type of `<gio-channel>`).

channel a `<gio-channel>`.

`g-io-channel-new-file` (*filename* `mchars`) (*mode* `mchars`) ⇒ (*ret* `<gio-channel*>`) [Function]

Open a file *filename* as a `<gio-channel>` 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`'.

ret A `<gio-channel>` on success, '`#f`' on failure.

`g-io-channel-read-chars` (*channel* `<gio-channel*>`) (*buf* `mchars`) (*count* `size_t`) ⇒ (*ret* `<gio-status>`) (*bytes_read* `size_t`) [Function]

Replacement for `g-io-channel-read` with the new API.

channel a `<gio-channel>`

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-'`#f`'. This indicates that the next UTF-8 character is too wide for the buffer.

error A location to return an error of type `<g-convert-error>` or `<gio-channel-error>`.

ret the status of the operation.

`g-io-channel-read-unichar` (*channel* `<gio-channel*>`) ⇒ (*ret* `<gio-status>`) (*thechar* `unsigned-int32`) [Function]

This function cannot be called on a channel with '`#f`' encoding.

channel	a `<gio-channel>`
thechar	a location to return a character
error	A location to return an error of type `<g-convert-error>` or `<gio-channel-error>`
ret	a `<gio-status>`

`g-io-channel-read-line` (*channel* `<gio-channel*>`) ⇒ (*ret* `<gio-status>`) (*str_return* `mchars`) (*length* `size_t`) (*terminator_pos* `size_t`) [Function]

Reads a line, including the terminating character(s), from a `<gio-channel>` into a newly-allocated string. *str-return* will contain allocated memory if the return is '`G_IO_STATUS_NORMAL`'.

channel	a `<gio-channel>`
str-return	The line read from the `<gio-channel>`, 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 '`#f`' instead.
length	location to store length of the read data, or '`#f`'
terminator-pos	location to store position of line terminator, or '`#f`'
error	A location to return an error of type `<g-convert-error>` or `<gio-channel-error>`
ret	the status of the operation.

`g-io-channel-read-line-string` (*channel* `<gio-channel*>`) (*buffer* `<g-string*>`) ⇒ (*ret* `<gio-status>`) (*terminator_pos* `size_t`) [Function]

Reads a line from a `<gio-channel>`, using a `<g-string>` as a buffer.

channel	a `<gio-channel>`
buffer	a `<g-string>` 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 '`#f`'
error	a location to store an error of type `<g-convert-error>` or `<gio-channel-error>`
ret	the status of the operation.

`g-io-channel-read-to-end` (*channel* `<gio-channel*>`) ⇒ (*ret* `<gio-status>`) (*str_return* `mchars`) (*length* `size_t`) [Function]

Reads all the remaining data from the file.

channel	a `<gio-channel>`
str-return	Location to store a pointer to a string holding the remaining data in the `<gio-channel>`. 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 `<g-convert-error>` or `<gio-channel-error>`
ret	'`G_IO_STATUS_NORMAL`' on success. This function never returns '`G_IO_STATUS_EOF`'.

`g-io-channel-write-chars` (*channel* `<gio-channel*>`) (*buf* `mchars`) (*count* `ssize_t`) ⇒ (*ret* `<gio-status>`) (*bytes_written* `size_t`) [Function]

Replacement for `g-io-channel-write` with the new API.

On seekable channels with encodings other than '`#f`' 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 `<gio-channel>`
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 `<g-convert-error>` or `<gio-channel-error>`
ret	the status of the operation.

`g-io-channel-write-unichar` (*channel* `<gio-channel*>`) (*thechar* `unsigned-int32`) ⇒ (*ret* `<gio-status>`) [Function]

This function cannot be called on a channel with '`#f`' encoding.

channel	a `<gio-channel>`
thechar	a character
error	A location to return an error of type `<g-convert-error>` or `<gio-channel-error>`

ret a `<gio-status>`

`g-io-channel-flush` (*channel* `<gio-channel*>`) ⇒ (*ret* `<gio-status>`) [Function]

Flushes the write buffer for the GIOChannel.

channel a `<gio-channel>`

error location to store an error of type `<gio-channel-error>`

ret 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` (*self* `<gio-channel*>`) (*offset* `int64`) (*type* `<g-seek-type>`) ⇒ (*ret* `<gio-status>`) [Function]

Replacement for `g-io-channel-seek` with the new API.

channel a `<gio-channel>`

offset The offset in bytes from the position specified by *type*

type a `<g-seek-type>`. 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 `<gio-channel-error>`

ret the status of the operation.

`g-io-channel-shutdown` (*channel* `<gio-channel*>`) (*flush* `bool`) ⇒ (*ret* `<gio-status>`) [Function]

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 `<gio-channel>`

flush if '`TRUE`', flush pending

err location to store a `<gio-channel-error>`

ret the status of the operation.

`g-io-channel-error-from-errno` (*en* `int`) ⇒ (*ret* `<gio-channel-error>`) [Function]

Converts an '`errno`' error number to a `<gio-channel-error>`.

en an '`errno`' error number, e.g. '`EINVAL`'.

ret a `<gio-channel-error>` error number, e.g. '`G_IO_CHANNEL_ERROR_INVAL`'.

`g-io-channel-ref` (*channel* `<gio-channel*>`) ⇒ (*ret* `<gio-channel*>`) [Function]

Increments the reference count of a `<gio-channel>`.

channel a `<gio-channel>`.

ret the *channel* that was passed in (since 2.6) `g-io-channel-unref` (*channel* <gio-channel*>) [Function]

Decrements the reference count of a `<gio-channel>`.

channel	a `<gio-channel>`.

`g-io-create-watch` (*channel* <gio-channel*>) (*condition* <gio-condition>) ⇒ (*ret* <g-source*>) [Function]

Creates a `<g-source>` 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.

On Windows, polling a `<g-source>` created to watch a channel for a socket puts the socket in non-blocking mode. This is a side-effect of the implementation and unavoidable.

channel	a `<gio-channel>` to watch
condition	conditions to watch for
ret	a new `<g-source>`

`g-io-add-watch` (*channel* <gio-channel*>) (*condition* <gio-condition>) (*func* scm) ⇒ (*ret* bool) [Function]

Adds the `<gio-channel>` into the main event loop with the default priority.

channel	a `<gio-channel>`.
condition	the condition to watch for.
func	the function to call when the condition is satisfied.
user-data	user data to pass to *func*.
ret	the event source id.

`g-io-channel-get-buffer-size` (*channel* <gio-channel*>) ⇒ (*ret* size_t) [Function]

Gets the buffer size.

channel	a `<gio-channel>`
ret	the size of the buffer.

`g-io-channel-set-buffer-size` (*channel* <gio-channel*>) (*size* size_t) [Function]

Sets the buffer size.

channel	a `<gio-channel>`
size	the size of the buffer. 0 == pick a good size

`g-io-channel-get-buffer-condition` (*channel* <gio-channel*>) ⇒ (*ret* <gio-condition>) [Function]

This function returns a `<gio-condition>` depending on whether there is data to be read/space to write data in the internal buffers in the `<gio-channel>`. Only the flags '`G_IO_IN`' and '`G_IO_OUT`' may be set.

channel	A `<gio-channel>`
ret	A `<gio-condition>`

g-io-channel-get-flags (*channel* `<gio-channel*>`) ⇒ (*ret* `<gio-flags>`) [Function]

Gets the current flags for a `<gio-channel>`, 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 `<gio-channel>`
ret	the flags which are set on the channel

g-io-channel-set-flags (*channel* `<gio-channel*>`) (*flags* `<gio-flags>`) ⇒ (*ret* `<gio-status>`) [Function]

Sets the (writeable) flags in *channel* to (*flags* & '`G_IO_CHANNEL_SET_MASK`').

channel	a `<gio-channel>`.
flags	the flags to set on the IO channel.
error	A location to return an error of type `<gio-channel-error>`.
ret	the status of the operation.

g-io-channel-get-line-term (*channel* `<gio-channel*>`) ⇒ (*ret* `mchars`) (*length* `int`) [Function]

This returns the string that `<gio-channel>` uses to determine where in the file a line break occurs. A value of '`#f`' indicates auto detection.

channel	a `<gio-channel>`
length	a location to return the length of the line terminator
ret	The line termination string. This value is owned by GLib and must not be freed.

g-io-channel-set-line-term (*channel* `<gio-channel*>`) (*line_term* `mchars`) (*length* `int`) [Function]

This sets the string that `<gio-channel>` uses to determine where in the file a line break occurs.

channel	a `<gio-channel>`
line-term	The line termination string. Use '`#f`' 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` (*channel* `<gio-channel*>`) ⇒ (*ret* `bool`) [Function]

Returns whether *channel* is buffered.

channel	a `<gio-channel>`.
ret	'`TRUE`' if the *channel* is buffered.

`g-io-channel-set-buffered` (*channel* `<gio-channel*>`) (*buffered* `bool`) [Function]

The buffering state can only be set if the channel's encoding is '`#f`'. 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 `<gio-channel>`
buffered	whether to set the channel buffered or unbuffered

`g-io-channel-get-encoding` (*channel* `<gio-channel*>`) ⇒ (*ret* `mchars`) [Function]

Gets the encoding for the input/output of the channel. The internal encoding is always UTF-8. The encoding '`#f`' makes the channel safe for binary data.

channel	a `<gio-channel>`
ret	A string containing the encoding, this string is owned by GLib and must not be freed.

`g-io-channel-set-encoding` (*channel* `<gio-channel*>`) (*encoding* `mchars`) ⇒ (*ret* `<gio-status>`) [Function]

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 '`#f`' 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 '`#f`' 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 `<gio-channel>`
encoding	the encoding type
error	location to store an error of type `<g-convert-error>`.
ret	'`G_IO_STATUS_NORMAL`' if the encoding was successfully set.

g-io-channel-get-close-on-unref (*channel* <gio-channel*>) ⇒ (*ret* bool) [Function]

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 `<gio-channel>`.
ret	Whether the channel will be closed on the final unref of the GIOChannel data structure.

g-io-channel-set-close-on-unref (*channel* <gio-channel*>) (*do_close* bool) [Function]

Setting this flag to '`TRUE`' for a channel you have already closed can cause problems.

channel	a `<gio-channel>`
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 (*channel* <gio-channel*>) (*buf* mchars) (*count* size_t) ⇒ (*ret* <gio-error>) (*bytes_read* size_t) [Function]

'`g_io_channel_read`' has been deprecated since version 2.2 and should not be used in newly-written code. Use `g-io-channel-read-chars` instead.

Reads data from a `<gio-channel>`.

channel	a `<gio-channel>`.
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 `<gio-channel>`.
bytes-read	returns the number of bytes actually read.
ret	'`G_IO_ERROR_NONE`' if the operation was successful.

`g-io-channel-write` (*channel* `<gio-channel*>`) (*buf* `mchars`) (*count* `size_t`) ⇒ (*ret* `<gio-error>`) (*bytes_written* `size_t`) [Function]

'`g_io_channel_write`' has been deprecated since version 2.2 and should not be used in newly-written code. Use `g-io-channel-write-chars` instead.

Writes data to a `<gio-channel>`.

channel a `<gio-channel>`.

buf the buffer containing the data to write.

count the number of bytes to write.

bytes-written
the number of bytes actually written.

ret '`G_IO_ERROR_NONE`' if the operation was successful.

`g-io-channel-seek` (*channel* `<gio-channel*>`) (*offset* `int64`) (*type* `<g-seek-type>`) ⇒ (*ret* `<gio-error>`) [Function]

'`g_io_channel_seek`' has been deprecated since version 2.2 and should not be used in newly-written code. Use `g-io-channel-seek-position` instead.

Sets the current position in the `<gio-channel>`, similar to the standard library function `fseek`.

channel a `<gio-channel>`.

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).

ret '`G_IO_ERROR_NONE`' if the operation was successful.

`g-io-channel-close` (*channel* `<gio-channel*>`) [Function]

'`g_io_channel_close`' has been deprecated since version 2.2 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 `<gio-channel>`

6 The Main Event Loop: manages all available sources of events.

6.1 Overview

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 `<g-main-context>`. A `<g-main-context>` 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 `<g-main-loop>` data type represents a main event loop. A `<g-main-loop>` 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 `<g-main-loop>` recursively. This is often used in GTK+ applications when showing modal dialog boxes. Note that event sources are associated with a particular `<g-main-context>`, and will be checked and dispatched for all main loops associated with that `<g-main-context>`.

GTK+ contains wrappers of some of these functions, e.g. `gtk-main`, `gtk-main-quit` and `gtk-events-pending`.

6.2 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 `<g-source>` structure. The derived type of source is represented by a structure that has the `<g-source>` 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 `<g-source-funcs>` 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 `<g-source-funcs>` 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`.

6.3 Customizing the main loop iteration

Single iterations of a `<g-main-context>` 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 `<g-main-loop>` 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 *(the missing section, mainloop-states.*

6.4 Usage

`g-main-loop-new` (*context* `<g-main-context*>`) (*is_running* `bool`) ⇒ (*ret* `<g-main-loop*>`) [Function]

Creates a new `<g-main-loop>` structure.

context a `<g-main-context>` (if '`#f`', 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.

ret a new `<g-main-loop>`.

`g-main-loop-ref` (*loop* `<g-main-loop*>`) ⇒ (*ret* `<g-main-loop*>`) [Function]

Increases the reference count on a `<g-main-loop>` object by one.

loop a `<g-main-loop>`

ret *loop*

`g-main-loop-unref` (*loop* `<g-main-loop*>`) [Function]

Decreases the reference count on a `<g-main-loop>` object by one. If the result is zero, free the loop and free all associated memory.

loop a `<g-main-loop>`

`g-main-loop-run` (*self* `<g-main-loop*>`) [Function]

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 `<g-main-context>`, it will process events from the loop, otherwise it will simply wait.

loop a `<g-main-loop>`

`g-main-loop-quit` (*loop* `<g-main-loop*>`) [Function]

Stops a `<g-main-loop>` from running. Any calls to `g-main-loop-run` for the loop will return.

loop a `<g-main-loop>`

`g-main-loop-is-running` (*loop* `<g-main-loop*>`) ⇒ (*ret* `bool`) [Function]

Checks to see if the main loop is currently being run via `g-main-loop-run`.

loop	a `<g-main-loop>`.
ret	'`TRUE`' if the mainloop is currently being run.

`g-main-loop-get-context` (*`loop`* `<g-main-loop*>`) ⇒ (*`ret`* `<g-main-context*>`) [Function]

Returns the `<g-main-context>` of *loop*.

loop	a `<g-main-loop>`.
ret	the `<g-main-context>` of *loop*

`g-main-context-new` ⇒ (*`ret`* `<g-main-context*>`) [Function]

Creates a new `<g-main-context>` strcuture

ret	the new `<g-main-context>`

`g-main-context-ref` (*`context`* `<g-main-context*>`) ⇒ (*`ret`* `<g-main-context*>`) [Function]

Increases the reference count on a `<g-main-context>` object by one.

context	a `<g-main-context>`
ret	the *context* that was passed in (since 2.6)

`g-main-context-unref` (*`context`* `<g-main-context*>`) [Function]

Decreases the reference count on a `<g-main-context>` object by one. If the result is zero, free the context and free all associated memory.

context	a `<g-main-context>`

`g-main-context-default` ⇒ (*`ret`* `<g-main-context*>`) [Function]

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

ret	the default main context.

`g-main-context-iteration` (*`self`* `<g-main-context*>`) (*`may_block`* `bool`) ⇒ (*`ret`* `bool`) [Function]

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 `<g-main-context>` (if '`#f`', the default context will be used)
may-block	whether the call may block.
ret	'`TRUE`' if events were dispatched.

`g-main-context-pending` (*`context`* `<g-main-context*>`) ⇒ (*`ret`* `bool`) [Function]

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

context a `<g-main-context>` (if '`#f`', the default context will be used)

ret '`TRUE`' if events are pending.

`g-main-context-find-source-by-id` (*context* `<g-main-context*>`) (*source_id* `unsigned-int`) ⇒ (*ret* `<g-source*>`) [Function]

Finds a `<g-source>` given a pair of context and ID.

context a `<g-main-context>` (if '`#f`', the default context will be used)

source-id the source ID, as returned by `g-source-get-id`.

ret the `<g-source>` if found, otherwise, '`#f`'

`g-main-context-wakeup` (*context* `<g-main-context*>`) [Function]

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

context a `<g-main-context>`

`g-main-context-acquire` (*context* `<g-main-context*>`) ⇒ (*ret* `bool`) [Function]

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 `<g-main-context>`

ret '`TRUE`' if the operation succeeded, and this thread is now the owner of *context*.

`g-main-context-release` (*context* `<g-main-context*>`) [Function]

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 `<g-main-context>`

`g-main-context-is-owner` (*context* `<g-main-context*>`) ⇒ (*ret* `bool`) [Function]

Determines whether this thread holds the (recursive) ownership of this `<g-maincontext>`. This is useful to know before waiting on another thread that may be blocking to get ownership of *context*.

context a `<g-main-context>`

ret '`TRUE`' if current thread is owner of *context*.

Since 2.10

`g-main-context-prepare` (*context* `<g-main-context*>`) ⇒ (*ret* `bool`) (*priority* `int`) [Function]

Prepares to poll sources within a main loop. The resulting information for polling is determined by calling `g-main-context-query`.

context	a `<g-main-context>`
priority	location to store priority of highest priority source already ready.
ret	'`TRUE`' if some source is ready to be dispatched prior to polling.

`g-main-context-query` (*context* `<g-main-context*>`) (*max_priority* `int`) (*fds* `<g-poll-fd*>`) (*n_fds* `int`) ⇒ (*ret* `int`) (*timeout_* `int`) [Function]

Determines information necessary to poll this main loop.

context	a `<g-main-context>`
max-priority	maximum priority source to check
timeout	location to store timeout to be used in polling
fds	location to store `<g-poll-fd>` records that need to be polled.
n-fds	length of *fds*.
ret	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` (*context* `<g-main-context*>`) (*max_priority* `int`) (*fds* `<g-poll-fd*>`) (*n_fds* `int`) ⇒ (*ret* `int`) [Function]

Passes the results of polling back to the main loop.

context	a `<g-main-context>`
max-priority	the maximum numerical priority of sources to check
fds	array of `<g-poll-fd>`'s that was passed to the last call to `g-main-context-query`
n-fds	return value of `g-main-context-query`
ret	'`TRUE`' if some sources are ready to be dispatched.

`g-main-context-dispatch` (*context* `<g-main-context*>`) [Function]

Dispatches all pending sources.

context	a `<g-main-context>`

`g-main-context-add-poll` (*self* `<g-main-context*>`) (*fd* `<g-poll-fd*>`) (*priority* `int`) [Function]

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 `<g-main-context>` (or '`#f`' for the default context)

fd a `<g-poll-fd>` 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` (*context* `<g-main-context*>`) (*fd* `<g-poll-fd*>`) [Function]

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

context a `<g-main-context>`

fd a `<g-poll-fd>` descriptor previously added with `g-main-context-add-poll`

`g-main-depth` ⇒ (*ret* `int`) [Function]

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

ret the depth of the stack of calls to `g-main-context-dispatch` on any `<g-main-context>` 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. 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` 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.
2.

Use `gtk-widget-set-sensitive` or modal dialogs to prevent the user from interacting with elements while the main loop is recursing.

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-current-source` ⇒ (*ret* `<g-source*>`) [Function]

Returns the currently firing source for this thread.

ret	The currently firing source or '`#f`'.

Since 2.12

`g-timeout-source-new` (*interval* `unsigned-int`) ⇒ (*ret* `<g-source*>`) [Function]

Creates a new timeout source.

The source will not initially be associated with any `<g-main-context>` and must be added to one with `g-source-attach` before it will be executed.

interval	the timeout interval in milliseconds.
ret	the newly-created timeout source

`g-idle-source-new` ⇒ (*ret* `<g-source*>`) [Function]

Creates a new idle source.

The source will not initially be associated with any `<g-main-context>` 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`'.

ret	the newly-created idle source

`g-child-watch-source-new` (*pid* `int`) ⇒ (*ret* `<g-source*>`) [Function]

Creates a new child_watch source.

The source will not initially be associated with any `<g-main-context>` and must be added to one with `g-source-attach` before it will be executed.

Note that child watch sources can only be used in conjunction with '`g_spawn...`' when the '`G_SPAWN_DO_NOT_REAP_CHILD`' flag is used.

Note that on platforms where `<g-pid>` must be explicitly 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.

Note further that using `g-child-watch-source-new` is not compatible with calling '`waitpid(-1)`' in the application. Calling `waitpid` for individual pids will still work fine.

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).
ret	the newly-created child watch source

Since 2.4

`g-source-new` (*source_funcs* `<g-source-funcs*>`) (*struct_size* `unsigned-int`) ⇒ (*ret* `<g-source*>`) [Function]

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

The source will not initially be associated with any `<g-main-context>` 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 `<g-source>` structure to create.
ret	the newly-created `<g-source>`.

`g-source-ref` (*source* `<g-source*>`) ⇒ (*ret* `<g-source*>`) [Function]

Increases the reference count on a source by one.

source	a `<g-source>`
ret	*source*

`g-source-unref` (*source* `<g-source*>`) [Function]

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 `<g-source>`

`g-source-attach` (*self* `<g-source*>`) (*context* `<g-main-context*>`) ⇒ (*ret* `unsigned-int`) [Function]

Adds a `<g-source>` to a *context* so that it will be executed within that context.

source	a `<g-source>`
context	a `<g-main-context>` (if '`#f`', the default context will be used)
ret	the ID (greater than 0) for the source within the `<g-main-context>`.

`g-source-destroy` (*source* `<g-source*>`) [Function]

Removes a source from its `<g-main-context>`, if any, and mark it as destroyed. The source cannot be subsequently added to another context.

source	a `<g-source>`

`g-source-is-destroyed` (*source* `<g-source*>`) ⇒ (*ret* `bool`) [Function]

Returns whether *source* has been destroyed.

This is important when you operate upon your objects from within idle handlers, but may have freed the object before the dispatch of your idle handler.

```
static gboolean
idle_callback (gpointer data)
{
  SomeWidget *self = data;
  GDK_THREADS_ENTER ();
  /* do stuff with self */
  self->idle_id = 0;
  GDK_THREADS_LEAVE ();

  return FALSE;
}

static void
some_widget_do_stuff_later (SomeWidget *self)
{
  self->idle_id = g_idle_add (idle_callback, self);
}

static void
some_widget_finalize (GObject *object)
{
  SomeWidget *self = SOME_WIDGET (object);

  if (self->idle_id)
    g_source_remove (self->idle_id);

  G_OBJECT_CLASS (parent_class)->finalize (object);
}
```

This will fail in a multi-threaded application if the widget is destroyed before the idle handler fires due to the use after free in the callback. A solution, to this particular problem, is to check to if the source has already been destroy within the callback.

```
static gboolean
idle_callback (gpointer data)
{
  SomeWidget *self = data;

  GDK_THREADS_ENTER ();
  if (!g_source_is_destroyed (g_main_current_source ()))
    {
      /* do stuff with self */
    }
  GDK_THREADS_LEAVE ();

  return FALSE;
}
```

source	a `<g-source>`
ret	'`TRUE`' if the source has been destroyed

Since 2.12

`g-source-set-priority` (*source* `<g-source*>`) (*priority* `int`) [Function]

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 `<g-source>`

priority the new priority.

`g-source-get-priority` (*source* `<g-source*>`) ⇒ (*ret* `int`) [Function]

Gets the priority of a source.

source a `<g-source>`

ret the priority of the source

`g-source-set-can-recurse` (*source* `<g-source*>`) (*can_recurse* `bool`) [Function]

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 `<g-source>`

can-recurse
whether recursion is allowed for this source

`g-source-get-can-recurse` (*source* `<g-source*>`) ⇒ (*ret* `bool`) [Function]

Checks whether a source is allowed to be called recursively. see `g-source-set-can-recurse`.

source a `<g-source>`

ret whether recursion is allowed.

`g-source-get-id` (*source* `<g-source*>`) ⇒ (*ret* `unsigned-int`) [Function]

Returns the numeric ID for a particular source. The ID of a source is a positive integer which 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 `<g-source>`

ret the ID (greater than 0) for the source

`g-source-get-context` (*source* `<g-source*>`) ⇒ (*ret* `<g-main-context*>`) [Function]

Gets the `<g-main-context>` with which the source is associated. Calling this function on a destroyed source is an error.

source a `<g-source>`

ret the `<g-main-context>` with which the source is associated, or '`#f`' if the context has not yet been added to a source.

`g-source-add-poll` (*source* `<g-source*>`) (*fd* `<g-poll-fd*>`) [Function]

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 `<g-poll-fd>` struct and return '`TRUE`' if events need to be processed.

source	a `<g-source>`
fd	a `<g-poll-fd>` structure holding information about a file descriptor to watch.

`g-source-remove-poll` (*source* `<g-source*>`) (*fd* `<g-poll-fd*>`) [Function]

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

source	a `<g-source>`
fd	a `<g-poll-fd>` structure previously passed to `g-source-add-poll`.

`g-source-get-current-time` (*source* `<g-source*>`) (*timeval* `<g-time-val*>`) [Function]

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 `<g-source>`
timeval	`<g-time-val>` structure in which to store current time.

`g-source-remove` (*tag* `unsigned-int`) ⇒ (*ret* `bool`) [Function]

Removes the source with the given id from the default main context. The id of a `<g-source>` 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.
ret	'`TRUE`' if the source was found and removed.

7 Miscellaneous Utility Functions: a selection of portable utility functions.

7.1 Overview

These are portable utility functions.

7.2 Usage

`g-get-application-name` ⇒ (*ret* `mchars`) [Function]

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 '`#f`' if `g-set-prgname` has also not been called).

ret human-readable application name. may return '`#f`'

Since 2.2

`g-set-application-name` (*application_name* `mchars`) [Function]

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` ⇒ (*ret* `mchars`) [Function]

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]`'.)

ret the name of the program. The returned string belongs to GLib and must not be modified or freed.

`g-set-prgname` (*prgname* `mchars`) [Function]

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` (*variable* `mchars`) ⇒ (*ret* `mchars`) [Function]

Returns the value of an environment variable. The name and value are in the GLib file name encoding. On UNIX, this means the actual bytes which might or might not be in some consistent character set and encoding. On Windows, it is in UTF-8. On Windows, in case the environment variable's value contains references to other environment variables, they are expanded.

variable	the environment variable to get, in the GLib file name encoding.
ret	the value of the environment variable, or '`#f`' 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` (*variable* `mchars`) (*value* `mchars`) (*overwrite* `bool`) ⇒ (*ret* `bool`) [Function]

Sets an environment variable. Both the variable's name and value should be in the GLib file name encoding. On UNIX, this means that they can be any sequence of bytes. On Windows, they should be in UTF-8.

Note that on some systems, when variables are overwritten, the memory used for the previous variables and its value isn't reclaimed.

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.
ret	'`FALSE`' if the environment variable couldn't be set.

Since 2.4

`g-unsetenv` (*variable* `mchars`) [Function]

Removes an environment variable from the environment.

Note that on some systems, when variables are overwritten, the memory used for the previous variables and its value isn't 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` ⇒ (*ret* `mchars`) [Function]

Gets the user name of the current user. The encoding of the returned string is system-defined. On UNIX, it might be the preferred file name encoding, or something else, and there is no guarantee that it is even consistent on a machine. On Windows, it is always UTF-8.

ret	the user name of the current user.

`g-get-real-name` ⇒ (*ret* `mchars`) [Function]

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. (On Windows, it is, however, always UTF-8.) If the real user name cannot be determined, the string "Unknown" is returned.

ret	the user's real name.

`g-get-user-cache-dir` ⇒ (*ret* `mchars`) [Function]

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

ret a string owned by GLib that must not be modified or freed.

Since 2.6

`g-get-user-data-dir` ⇒ (*ret* `mchars`) [Function]

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

ret a string owned by GLib that must not be modified or freed.

Since 2.6

`g-get-user-config-dir` ⇒ (*ret* `mchars`) [Function]

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

ret a string owned by GLib that must not be modified or freed.

Since 2.6

`g-get-host-name` ⇒ (*ret* `mchars`) [Function]

Return a name for the machine.

The returned name is not necessarily a fully-qualified domain name, or even present in DNS or some other name service at all. It need not even be unique on your local network or site, but usually it is. Callers should not rely on the return value having any specific properties like uniqueness for security purposes. Even if the name of the machine is changed while an application is running, the return value from this function does not change. The returned string is owned by GLib and should not be modified or freed. If no name can be determined, a default fixed string "localhost" is returned.

ret the host name of the machine.

Since 2.8

`g-get-home-dir` ⇒ (*ret* `mchars`) [Function]

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.

ret the current user's home directory.

`g-get-tmp-dir` ⇒ (*ret* `mchars`) [Function]

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. The encoding of the returned string is system-defined. On Windows, it is always UTF-8. The return value is never '`#f`'.

ret	the directory to use for temporary files.

`g-get-current-dir` ⇒ (*ret* `mchars`) [Function]

Gets the current directory. The returned string should be freed when no longer needed. The encoding of the returned string is system defined. On Windows, it is always UTF-8.

ret	the current directory.

`g-basename` (*file_name* `mchars`) ⇒ (*ret* `mchars`) [Function]

'`g_basename`' has been deprecated since version 2.2 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.
ret	the name of the file without any leading directory components.

`g-path-is-absolute` (*file_name* `mchars`) ⇒ (*ret* `bool`) [Function]

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.
ret	'`TRUE`' if *file-name* is an absolute path.

`g-path-skip-root` (*file_name* `mchars`) ⇒ (*ret* `mchars`) [Function]

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 '`#f`'.

file-name	a file name.
ret	a pointer into *file-name* after the root component.

`g-path-get-basename` (*file_name* `mchars`) ⇒ (*ret* `mchars`) [Function]

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.
ret	a newly allocated string containing the last component of the filename.

`g-path-get-dirname` (*file_name* `mchars`) ⇒ (*ret* `mchars`) [Function]

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.
ret	the directory components of the file.

`g-find-program-in-path` (*program* `mchars`) ⇒ (*ret* `mchars`) [Function]

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 '`#f`' 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 '`#f`' otherwise. On Windows, if *program* does not have a file type suffix, tries with the suffixes .exe, .cmd, .bat and .com, and the suffixes in the `PATHEXT` environment variable.

On Windows, it looks for the file in the same way as `create-process` would. This means first in the directory where the executing 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 in the GLib file name encoding
ret	absolute path, or '`#f`'

`g-nullify-pointer` (*nullify_location* `<gpointer*>`) [Function]

Set the pointer at the specified location to '`#f`'.

nullify-location	the memory address of the pointer.

8 Quarks: a 2-way association between a string and a unique integer identifier.

8.1 Overview

Quarks are associations between strings and integer identifiers. Given either the string or the `<g-quark>` 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 `<g-quark>`, use `g-quark-to-string`.

To find the `<g-quark>` corresponding to a given string, use `g-quark-try-string`.

Another use for the string pool maintained for the quark functions is string interning, using `g-intern-string` or `g-intern-static-string`. An interned string is a canonical representation for a string. One important advantage of interned strings is that they can be compared for equality by a simple pointer comparision, rather than using `strcmp`.

8.2 Usage

`g-quark-from-string` (*string* `mchars`) ⇒ (*ret* `unsigned-int`) [Function]

Gets the `<g-quark>` identifying the given string. If the string does not currently have an associated `<g-quark>`, a new `<g-quark>` is created, using a copy of the string.

string	a string.
ret	the `<g-quark>` identifying the string.

`g-quark-to-string` (*quark* `unsigned-int`) ⇒ (*ret* `mchars`) [Function]

Gets the string associated with the given `<g-quark>`.

quark	a `<g-quark>`.
ret	the string associated with the `<g-quark>`.

`g-quark-try-string` (*string* `mchars`) ⇒ (*ret* `unsigned-int`) [Function]

Gets the `<g-quark>` associated with the given string, or 0 if the string has no associated `<g-quark>`.

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.
ret	the `<g-quark>` associated with the string, or 0 if there is no `<g-quark>` associated with the string.

9 Shell-related Utilities: shell-like commandline handling.

9.1 Overview

9.2 Usage

`g-shell-quote` (*unquoted_string* `mchars`) ⇒ (*ret* `mchars`) [Function]

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

ret quoted string

`g-shell-unquote` (*quoted_string* `mchars`) ⇒ (*ret* `mchars`) [Function]

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 '`#f`' 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

ret an unquoted string

10 Strings: text buffers which grow automatically as text is added.

10.1 Overview

A `<g-string>` 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.

10.2 Usage

`g-string-new` (*init* `mchars`) ⇒ (*ret* `<g-string*>`) [Function]

Creates a new `<g-string>`, initialized with the given string.

init	the initial text to copy into the string.
ret	the new `<g-string>`.

11 Unicode Manipulation: functions operating on Unicode characters and UTF-8 strings.

11.1 Overview

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.

The implementations of the Unicode functions in GLib are based on the Unicode Character Data tables, which are available from www.unicode.org. GLib 2.8 supports Unicode 4.0, GLib 2.10 supports Unicode 4.1, GLib 2.12 supports Unicode 5.0.

11.2 Usage

`g-unichar-validate` (*ch* `unsigned-int32`) ⇒ (*ret* `bool`) [Function]

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
ret	'`TRUE`' if *ch* is a valid Unicode character

`g-unichar-isalnum` (*c* `unsigned-int32`) ⇒ (*ret* `bool`) [Function]

Determines whether a character is alphanumeric. Given some UTF-8 text, obtain a character value with `g-utf8-get-char`.

c	a Unicode character
ret	'`TRUE`' if *c* is an alphanumeric character

`g-unichar-isalpha` (*c* `unsigned-int32`) ⇒ (*ret* `bool`) [Function]

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
ret	'`TRUE`' if *c* is an alphabetic character

`g-unichar-iscntrl` (*c* `unsigned-int32`) ⇒ (*ret* `bool`) [Function]

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
ret	'`TRUE`' if *c* is a control character

`g-unichar-isdigit` (*c* `unsigned-int32`) ⇒ (*ret* `bool`) [Function]

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
ret	'TRUE' if *c* is a digit

g-unichar-isgraph (*c* **unsigned-int32**) ⇒ (*ret* **bool**) [Function]

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
ret	'TRUE' if *c* is printable unless it's a space

g-unichar-islower (*c* **unsigned-int32**) ⇒ (*ret* **bool**) [Function]

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
ret	'TRUE' if *c* is a lowercase letter

g-unichar-isprint (*c* **unsigned-int32**) ⇒ (*ret* **bool**) [Function]

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
ret	'TRUE' if *c* is printable

g-unichar-ispunct (*c* **unsigned-int32**) ⇒ (*ret* **bool**) [Function]

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
ret	'TRUE' if *c* is a punctuation or symbol character

g-unichar-isspace (*c* **unsigned-int32**) ⇒ (*ret* **bool**) [Function]

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
ret	'TRUE' if *c* is a space character

g-unichar-isupper (*c* **unsigned-int32**) ⇒ (*ret* **bool**) [Function]

Determines if a character is uppercase.

c	a Unicode character
ret	'TRUE' if *c* is an uppercase character

`g-unichar-isxdigit` (*c* `unsigned-int32`) ⇒ (*ret* `bool`) [Function]

Determines if a character is a hexidecimal digit.

c	a Unicode character.
ret	'`TRUE`' if the character is a hexadecimal digit

`g-unichar-istitle` (*c* `unsigned-int32`) ⇒ (*ret* `bool`) [Function]

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
ret	'`TRUE`' if the character is titlecase

`g-unichar-isdefined` (*c* `unsigned-int32`) ⇒ (*ret* `bool`) [Function]

Determines if a given character is assigned in the Unicode standard.

c	a Unicode character
ret	'`TRUE`' if the character has an assigned value

`g-unichar-iswide` (*c* `unsigned-int32`) ⇒ (*ret* `bool`) [Function]

Determines if a character is typically rendered in a double-width cell.

c	a Unicode character
ret	'`TRUE`' if the character is wide

`g-unichar-iswide-cjk` (*c* `unsigned-int32`) ⇒ (*ret* `bool`) [Function]

Determines if a character is typically rendered in a double-width cell under legacy East Asian locales. If a character is wide according to `g-unichar-iswide`, then it is also reported wide with this function, but the converse is not necessarily true. See the Unicode Standard Annex for details.

c	a Unicode character
ret	'`TRUE`' if the character is wide in legacy East Asian locales

Since 2.12

`g-unichar-toupper` (*c* `unsigned-int32`) ⇒ (*ret* `unsigned-int32`) [Function]

Converts a character to uppercase.

c	a Unicode character
ret	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` (*c* `unsigned-int32`) ⇒ (*ret* `unsigned-int32`) [Function]

Converts a character to lower case.

c	a Unicode character.
ret	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 (*c* `unsigned-int32`) ⇒ (*ret* `unsigned-int32`) [Function]

Converts a character to the titlecase.

c	a Unicode character
ret	the result of converting *c* to titlecase. If *c* is not an uppercase or lowercase character, *c* is returned unchanged.

g-unichar-digit-value (*c* `unsigned-int32`) ⇒ (*ret* `int`) [Function]

Determines the numeric value of a character as a decimal digit.

c	a Unicode character
ret	If *c* is a decimal digit (according to `g-unichar-isdigit`), its numeric value. Otherwise, -1.

g-unichar-xdigit-value (*c* `unsigned-int32`) ⇒ (*ret* `int`) [Function]

Determines the numeric value of a character as a hexidecimal digit.

c	a Unicode character
ret	If *c* is a hex digit (according to `g-unichar-isxdigit`), its numeric value. Otherwise, -1.

g-unichar-type (*c* `unsigned-int32`) ⇒ (*ret* `<g-unicode-type>`) [Function]

Classifies a Unicode character by type.

c	a Unicode character
ret	the type of the character.

g-unichar-break-type (*c* `unsigned-int32`) ⇒ (*ret* `<g-unicode-break-type>`) [Function]

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
ret	the break type of *c*

g-unichar-get-mirror-char (*ch* `unsigned-int32`) ⇒ (*ret* `bool`) (*mirrored_ch* `unsigned-int32`) [Function]

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 and *mirrored-ch* is set, it puts that character in the address pointed to by *mirrored-ch*. Otherwise the original character is put.

ch	a Unicode character
mirrored-ch	location to store the mirrored character
ret	'`TRUE`' if *ch* has a mirrored character, '`FALSE`' otherwise

Since 2.4

`g-utf8-get-char` (*p* `mchars`) ⇒ (*ret* `unsigned-int32`) [Function]

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
ret	the resulting character

`g-utf8-get-char-validated` (*p* `mchars`) (*max_len* `ssize_t`) ⇒ (*ret* `unsigned-int32`) [Function]

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.
ret	the resulting character. If *p* points to a partial sequence at the end of a string that could begin a valid character (or if *max-len* is zero), 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` (*str* `mchars`) (*offset* `long`) ⇒ (*ret* `mchars`) [Function]

Converts from an integer character offset to a pointer to a position within the string.

Since 2.10, this function allows to pass a negative *offset* to step backwards. It is usually worth stepping backwards from the end instead of forwards if *offset* is in the last fourth of the string, since moving forward is about 3 times faster than moving backward.

str	a UTF-8 encoded string
offset	a character offset within *str*
ret	the resulting pointer

`g-utf8-pointer-to-offset` (*str* `mchars`) (*pos* `mchars`) ⇒ (*ret* `long`) [Function]

Converts from a pointer to position within a string to a integer character offset.

Since 2.10, this function allows *pos* to be before *str*, and returns a negative offset in this case.

str	a UTF-8 encoded string
pos	a pointer to a position within *str*
ret	the resulting character offset

`g-utf8-prev-char` (*p* `mchars`) ⇒ (*ret* `mchars`) [Function]

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
ret	a pointer to the found character.

`g-utf8-find-next-char` (*p* `mchars`) (*end* `mchars`) ⇒ (*ret* `mchars`) [Function]

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 '`#f`' to indicate that the string is nul-terminated, in which case the returned value will be
ret	a pointer to the found character or '`#f`'

`g-utf8-find-prev-char` (*str* `mchars`) (*p* `mchars`) ⇒ (*ret* `mchars`) [Function]

Given a position *p* with a UTF-8 encoded string *str*, find the start of the previous UTF-8 character starting before *p*. Returns '`#f`' if no UTF-8 characters are present in *str* 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.

str	pointer to the beginning of a UTF-8 encoded string
p	pointer to some position within *str*
ret	a pointer to the found character or '`#f`'.

`g-utf8-strlen` (*p* `mchars`) (*max* `ssize_t`) ⇒ (*ret* `long`) [Function]

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 '`#f`'.
ret	the length of the string in characters

`g-utf8-strncpy` (*dest* `mchars`) (*src* `mchars`) (*n* `size_t`) ⇒ (*ret* `mchars`) [Function]

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
ret	*dest*

`g-utf8-strchr` (*p* `mchars`) (*len* `ssize_t`) (*c* `unsigned-int32`) ⇒ (*ret* `mchars`) [Function]

Finds the leftmost occurrence of the given Unicode 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 Unicode character
ret	'`#f`' 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` (*p* `mchars`) (*len* `ssize_t`) (*c* `unsigned-int32`) ⇒ (*ret* `mchars`) [Function]

Find the rightmost occurrence of the given Unicode 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 Unicode character
ret	'`#f`' 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` (*str* `mchars`) (*len* `ssize_t`) ⇒ (*ret* `mchars`) [Function]

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.
ret	a newly-allocated string which is the reverse of *str*.

Since 2.2

`g-utf8-validate` (*str* `mchars`) (*max_len* `ssize_t`) (*end* `<gchar**>`) ⇒ (*ret* `bool`) [Function]

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-'`#f`', then the end of the valid range will be stored there (i.e. the start of the first invalid character if some bytes were invalid, or the end of the text being validated otherwise).

Note that `g-utf8-validate` returns '`FALSE`' if *max-len* is positive and NUL is met before *max-len* bytes have been read.

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
ret	'`TRUE`' if the text was valid UTF-8

`g-utf8-strup` (*str* `mchars`) (*len* `ssize_t`) ⇒ (*ret* `mchars`) [Function]

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.
ret	a newly allocated string, with all characters converted to uppercase.

`g-utf8-strdown` (*str* `mchars`) (*len* `ssize_t`) ⇒ (*ret* `mchars`) [Function]

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.
ret	a newly allocated string, with all characters converted to lowercase.

`g-utf8-casefold` (*str* `mchars`) (*len* `ssize_t`) ⇒ (*ret* `mchars`) [Function]

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.
ret	a newly allocated string, that is a case independent form of *str*.

`g-utf8-normalize` (*str* `mchars`) (*len* `ssize_t`) (*mode* `<g-normalize-mode>`) ⇒ (*ret* `mchars`) [Function]

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.
ret	a newly allocated string, that is the normalized form of *str*.

`g-utf8-collate` (*str1* `mchars`) (*str2* `mchars`) ⇒ (*ret* `int`) [Function]

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

ret < 0 if *str1* compares before *str2*, 0 if they compare equal, > 0 if *str1* compares after *str2*.

`g-utf8-collate-key` (*str* `mchars`) (*len* `ssize_t`) ⇒ (*ret* `mchars`) [Function]

Converts a string into a collation key that can be compared with other collation keys produced by the same function 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.

ret a newly allocated string. This string should be freed with `g-free` when you are done with it.

`g-utf8-collate-key-for-filename` (*str* `mchars`) (*len* `ssize_t`) ⇒ (*ret* `mchars`) [Function]

Converts a string into a collation key that can be compared with other collation keys produced by the same function using `strcmp`.

In order to sort filenames correctly, this function treats the dot '.' as a special case. Most dictionary orderings seem to consider it insignificant, thus producing the ordering "event.c" "eventgenerator.c" "event.h" instead of "event.c" "event.h" "eventgenerator.c". Also, we would like to treat numbers intelligently so that "file1" "file10" "file5" is sorted as "file1" "file5" "file10".

str a UTF-8 encoded string.

len length of *str*, in bytes, or -1 if *str* is nul-terminated.

ret a newly allocated string. This string should be freed with `g-free` when you are done with it.

Since 2.8

`g-utf8-to-utf16` (*str* `mchars`) (*len* `long`) ⇒ (*ret* `<gunichar2*>`) (*items_read* `long`) (*items_written* `long`) [Function]

Convert a string from UTF-8 to UTF-16. A 0 character will be added to the result after the converted text.

str a UTF-8 encoded string

len the maximum length (number of characters) of *str* to use. If *len* < 0, then the string is nul-terminated.

items-read location to store number of bytes read, or '`#f`'. If '`#f`', 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 `<gunichar2>` written, or '`#f`'. The value stored here does not include the trailing 0.

error location to store the error occuring, or '#f' to ignore errors. Any of the errors in <g-convert-error> other than 'G_CONVERT_ERROR_NO_CONVERSION' may occur.

ret a pointer to a newly allocated UTF-16 string. This value must be freed with g-free. If an error occurs, '#f' will be returned and *error* set.

g-utf8-to-ucs4 (*str* mchars) (*len* long) ⇒ (*ret* <gunichar*>) (*items_read* long) (*items_written* long) [Function]

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 nulterminated.

items-read location to store number of bytes read, or '#f'. If '#f', 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 '#f'. The value here stored does not include the trailing 0 character.

error location to store the error occuring, or '#f' to ignore errors. Any of the errors in <g-convert-error> other than 'G_CONVERT_ERROR_NO_CONVERSION' may occur.

ret a pointer to a newly allocated UCS-4 string. This value must be freed with g-free. If an error occurs, '#f' will be returned and *error* set.

g-utf8-to-ucs4-fast (*str* mchars) (*len* long) ⇒ (*ret* <gunichar*>) (*items_written* long) [Function]

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 nulterminated.

items-written
location to store the number of characters in the result, or '#f'.

ret a pointer to a newly allocated UCS-4 string. This value must be freed with g-free.

g-utf16-to-ucs4 (*str* <gunichar2*>) (*len* long) ⇒ (*ret* <gunichar*>) (*items_read* long) (*items_written* long) [Function]

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 (number of `<gunichar2>`) 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 '`#f`'. If '`#f`', 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 '`#f`'. The value stored here does not include the trailing 0 character.
error	location to store the error occuring, or '`#f`' to ignore errors. Any of the errors in `<g-convert-error>` other than '`G_CONVERT_ERROR_NO_CONVERSION`' may occur.
ret	a pointer to a newly allocated UCS-4 string. This value must be freed with `g-free`. If an error occurs, '`#f`' will be returned and *error* set.

g-utf16-to-utf8 (*str* <gunichar2*>) (*len* long) ⇒ (*ret* mchars) (*items_read* long) (*items_written* long) [Function]

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 (number of `<gunichar2>`) 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 '`#f`'. If '`#f`', 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 '`#f`'. The value stored here does not include the trailing 0 byte.
error	location to store the error occuring, or '`#f`' to ignore errors. Any of the errors in `<g-convert-error>` other than '`G_CONVERT_ERROR_NO_CONVERSION`' may occur.
ret	a pointer to a newly allocated UTF-8 string. This value must be freed with `g-free`. If an error occurs, '`#f`' will be returned and *error* set.

g-ucs4-to-utf16 (*str* <gunichar*>) (*len* long) ⇒ (*ret* <gunichar2*>) (*items_read* long) (*items_written* long) [Function]

Convert a string from UCS-4 to UTF-16. A 0 character will be added to the result after the converted text.

str a UCS-4 encoded string

len the maximum length (number of characters) 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 '`#f`'. If an error occurs then the index of the invalid input is stored here.

items-written
location to store number of `<gunichar2>` written, or '`#f`'. The value stored here does not include the trailing 0.

error location to store the error occuring, or '`#f`' to ignore errors. Any of the errors in `<g-convert-error>` other than '`G_CONVERT_ERROR_NO_CONVERSION`' may occur.

ret a pointer to a newly allocated UTF-16 string. This value must be freed with `g-free`. If an error occurs, '`#f`' will be returned and *error* set.

`g-ucs4-to-utf8` (*str* `<gunichar*>`) (*len* `long`) ⇒ (*ret* `mchars`) (*items_read* `long`) (*items_written* `long`) [Function]

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 (number of characters) 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, or '`#f`'.

items-written
location to store number of bytes written or '`#f`'. The value here stored does not include the trailing 0 byte.

error location to store the error occuring, or '`#f`' to ignore errors. Any of the errors in `<g-convert-error>` other than '`G_CONVERT_ERROR_NO_CONVERSION`' may occur.

ret a pointer to a newly allocated UTF-8 string. This value must be freed with `g-free`. If an error occurs, '`#f`' will be returned and *error* set. In that case, *items-read* will be set to the position of the first invalid input character.

`g-unichar-to-utf8` (*c* `unsigned-int32`) (*outbuf* `mchars`) ⇒ (*ret* `int`) [Function]

Converts a single character to UTF-8.

c a Unicode character code

outbuf output buffer, must have at least 6 bytes of space. If '`#f`', the length will be computed and returned and nothing will be written to *outbuf*.

ret number of bytes written

Concept Index

(Index is nonexistent)

Function Index

g-base64-decode . . . 1
g-base64-encode . . . 1
g-basename . . . 48
g-bookmark-file-add-application . . . 13
g-bookmark-file-add-group . . . 13
g-bookmark-file-free . . . 3
g-bookmark-file-get-added . . . 7
g-bookmark-file-get-app-info . . . 9
g-bookmark-file-get-applications . . . 9
g-bookmark-file-get-description . . . 6
g-bookmark-file-get-groups . . . 8
g-bookmark-file-get-icon . . . 7
g-bookmark-file-get-is-private . . . 7
g-bookmark-file-get-mime-type . . . 6
g-bookmark-file-get-modified . . . 8
g-bookmark-file-get-size . . . 5
g-bookmark-file-get-title . . . 6
g-bookmark-file-get-uris . . . 6
g-bookmark-file-get-visited . . . 8
g-bookmark-file-has-application . . . 5
g-bookmark-file-has-group . . . 5
g-bookmark-file-has-item . . . 4
g-bookmark-file-load-from-data . . . 3
g-bookmark-file-load-from-data-dirs . . . 4
g-bookmark-file-load-from-file . . . 3
g-bookmark-file-move-item . . . 14
g-bookmark-file-new . . . 3
g-bookmark-file-remove-application . . . 14
g-bookmark-file-remove-group . . . 13
g-bookmark-file-remove-item . . . 14
g-bookmark-file-set-added . . . 11
g-bookmark-file-set-app-info . . . 12
g-bookmark-file-set-description . . . 10
g-bookmark-file-set-groups . . . 11
g-bookmark-file-set-icon . . . 10
g-bookmark-file-set-is-private . . . 10
g-bookmark-file-set-mime-type . . . 10
g-bookmark-file-set-modified . . . 11
g-bookmark-file-set-title . . . 9
g-bookmark-file-set-visited . . . 12
g-bookmark-file-to-data . . . 4
g-bookmark-file-to-file . . . 4
g-child-watch-source-new . . . 40
g-file-error-from-errno . . . 21
g-file-get-contents . . . 21
g-file-open-tmp . . . 23
g-file-read-link . . . 23
g-file-test . . . 22
g-filename-display-basename . . . 19
g-filename-display-name . . . 19
g-filename-from-uri . . . 19
g-filename-from-utf8 . . . 18
g-filename-to-uri . . . 19
g-filename-to-utf8 . . . 18
g-find-program-in-path . . . 49
g-get-application-name . . . 45
g-get-current-dir . . . 48
g-get-home-dir . . . 47
g-get-host-name . . . 47
g-get-prgname . . . 45
g-get-real-name . . . 46
g-get-tmp-dir . . . 48
g-get-user-cache-dir . . . 47
g-get-user-config-dir . . . 47
g-get-user-data-dir . . . 47
g-get-user-name . . . 46
g-getenv . . . 45
g-idle-source-new . . . 40
g-io-add-watch . . . 29
g-io-channel-close . . . 33
g-io-channel-error-from-errno . . . 28
g-io-channel-flush . . . 28
g-io-channel-get-buffer-condition . . . 29
g-io-channel-get-buffer-size . . . 29
g-io-channel-get-buffered . . . 31
g-io-channel-get-close-on-unref . . . 32
g-io-channel-get-encoding . . . 31
g-io-channel-get-flags . . . 30
g-io-channel-get-line-term . . . 30
g-io-channel-init . . . 25
g-io-channel-new-file . . . 25
g-io-channel-read . . . 32
g-io-channel-read-chars . . . 25
g-io-channel-read-line . . . 26
g-io-channel-read-line-string . . . 26
g-io-channel-read-to-end . . . 27
g-io-channel-read-unichar . . . 26
g-io-channel-ref . . . 28
g-io-channel-seek . . . 33
g-io-channel-seek-position . . . 28
g-io-channel-set-buffer-size . . . 29
g-io-channel-set-buffered . . . 31
g-io-channel-set-close-on-unref . . . 32
g-io-channel-set-encoding . . . 31
g-io-channel-set-flags . . . 30
g-io-channel-set-line-term . . . 30
g-io-channel-shutdown . . . 28
g-io-channel-unix-get-fd . . . 25
g-io-channel-unix-new . . . 24
g-io-channel-unref . . . 29
g-io-channel-write . . . 33
g-io-channel-write-chars . . . 27
g-io-channel-write-unichar . . . 27
g-io-create-watch . . . 29
g-locale-from-utf8 . . . 20
g-locale-to-utf8 . . . 17
g-main-context-acquire . . . 37
g-main-context-add-poll . . . 38
g-main-context-check . . . 38
g-main-context-default . . . 36

g-main-context-dispatch 38
g-main-context-find-source-by-id......... 37
g-main-context-is-owner 37
g-main-context-iteration 36
g-main-context-new........................ 36
g-main-context-pending 36
g-main-context-prepare 38
g-main-context-query...................... 38
g-main-context-ref 36
g-main-context-release 37
g-main-context-remove-poll 39
g-main-context-unref...................... 36
g-main-context-wakeup 37
g-main-current-source 40
g-main-depth 39
g-main-loop-get-context 36
g-main-loop-is-running 35
g-main-loop-new 35
g-main-loop-quit 35
g-main-loop-ref 35
g-main-loop-run 35
g-main-loop-unref 35
g-mkstemp 22
g-nullify-pointer 49
g-path-get-basename....................... 48
g-path-get-dirname........................ 49
g-path-is-absolute........................ 48
g-path-skip-root 48
g-quark-from-string....................... 50
g-quark-to-string 50
g-quark-try-string........................ 50
g-set-application-name 45
g-set-prgname 45
g-setenv................................... 46
g-shell-quote 51
g-shell-unquote 51
g-source-add-poll 44
g-source-attach 41
g-source-destroy 41
g-source-get-can-recurse 43
g-source-get-context...................... 43
g-source-get-current-time 44
g-source-get-id 43
g-source-get-priority 43
g-source-is-destroyed 41
g-source-new 41
g-source-ref 41
g-source-remove 44
g-source-remove-poll...................... 44
g-source-set-can-recurse 43
g-source-set-priority 43
g-source-unref 41
g-string-new 52
g-timeout-source-new...................... 40
g-ucs4-to-utf16 64
g-ucs4-to-utf8 65
g-unichar-break-type...................... 56
g-unichar-digit-value 56
g-unichar-get-mirror-char 56
g-unichar-isalnum 53
g-unichar-isalpha 53
g-unichar-iscntrl 53
g-unichar-isdefined....................... 55
g-unichar-isdigit 53
g-unichar-isgraph 54
g-unichar-islower 54
g-unichar-isprint 54
g-unichar-ispunct 54
g-unichar-isspace 54
g-unichar-istitle 55
g-unichar-isupper 54
g-unichar-iswide 55
g-unichar-iswide-cjk...................... 55
g-unichar-isxdigit 55
g-unichar-to-utf8 65
g-unichar-tolower 55
g-unichar-totitle 56
g-unichar-toupper 55
g-unichar-type 56
g-unichar-validate 53
g-unichar-xdigit-value 56
g-unsetenv 46
g-utf16-to-ucs4 63
g-utf16-to-utf8 64
g-utf8-casefold 61
g-utf8-collate 61
g-utf8-collate-key 62
g-utf8-collate-key-for-filename.......... 62
g-utf8-find-next-char 58
g-utf8-find-prev-char 58
g-utf8-get-char 57
g-utf8-get-char-validated 57
g-utf8-normalize 61
g-utf8-offset-to-pointer 57
g-utf8-pointer-to-offset 58
g-utf8-prev-char 58
g-utf8-strchr 59
g-utf8-strdown 60
g-utf8-strlen 59
g-utf8-strncpy 59
g-utf8-strrchr 59
g-utf8-strreverse 59
g-utf8-strup 60
g-utf8-to-ucs4 63
g-utf8-to-ucs4-fast....................... 63
g-utf8-to-utf16 62
g-utf8-validate 60