Guile-GNOME: GConf

version 2.16.2, updated 9 December 2011

Havoc Pennington

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

Copyright 1999-2007 Havoc Pennington

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU General Public License, Version 2 or any later version published by the Free Software Foundation.

Short Contents

1 Overview 1
2 GConfClient 2
3 GConf Core Interfaces 9
4 GConfChangeSet 11
5 GConfEngine 12
6 GError 13
7 GConfSchema 14
8 GConfValue, GConfEntry, GConfMetaInfo 17
9 Undocumented 18
Type Index 19
Function Index 20

1 Overview

`(gnome gconf)` wraps the GNOME configuration library for Guile. It is a part of Guile-GNOME.

1.1 Example code

```
(define *gconf-client* (gconf-client-get-default))
(define *gconf-dir* "/apps/my-app")
(gconf-client-add-dir gconf-client gconf-dir 'preload-onelevel)
(define (gconf-key s)
  (string-append gconf-dir "/" (symbol->string s)))

(define (state-ref key default)
  (catch #t
         (lambda ()
           (gconf-client-get *gconf-client* (gconf-key key)))
         (lambda args default)))
(define (state-set! key val)
  (gconf-client-set *gconf-client* (gconf-key key) val))

(define (value-changed client cnxn-id key val)
  (format #t "~a: ~a\n" key val))
(gconf-client-add-notify *gconf-client* "/apps/my-app"
                         value-changed)
```

Note that the `value-changed` procedure will only be called if you have a main loop running.

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

2 GConfClient

convenience wrapper

2.1 Overview

<g-conf-client> adds the following features to plain GConf:

A client-side cache for a specified list of directories you're interested in. You can "preload" entire directories into the cache, speeding things up even more.

Some automatic error handling, if you request it.

Signals when a value changes or an error occurs.

If you use <g-conf-client>, you should not use the underlying <g-conf-engine> directly, or you'll break things. This is why there's no gconf-client-get-engine function; in fact, if you create the <g-conf-client> with gconf-client-get-default, there is no (legitimate) way to obtain a pointer to the underlying <g-conf-engine>. If you create a <g-conf-client> from an existing engine, you'll have to be disciplined enough to avoid using that engine directly.

This is all a white lie; *some* direct <g-conf-engine> operations are safe. But it's complicated to know which, and if an operation isn't safe the resulting bugs will mangle the cache and cause weird bugs at an indeterminate time in the future; you don't want to risk this situation.

A <g-conf-client> has a list of directories that it "watches." These directories are optionally pre-loaded into the cache, and monitored in order to emit the <value-changed> signal. The <g-conf-client> can also be used to access directories not in the list, but those directories won't be preloaded and the "value_changed" signal won't be emitted for them.

There are two error-related signals in <g-conf-client>. The first is plain "error"; it's emitted anytime an error occurs. The second is "unreturned_error"; this signal is emitted if you pass as the <g-error>** to any <g-conf-client> function. The idea is that you can have a global error handler attached to the "unreturned_error" signal; if you want to use this handler, you don't need to use the normal GConf error handling mechanism. However, if you ever need to handle errors for a specific function call, you can override the global handler by passing a non-<g-error>** to the function. If you want an error handler that's *always* invoked, use the "error" signal.

The "value_changed" signal is emitted whenever the server notifies your client program that a value has changed in the GConf database. There's one problem with this signal: the signal handler has to use strcmp to determine whether the changed value is the one it was interested in. If you are interested in lots of values, then every time a value changes you'll be making lots of calls to strcmp and getting O(n) performance. gconf-client-notify-add is a superior interface in most cases for this reason. Note that calling gconf-client-set and its relatives will cause "value_changed" to be emitted, but "value_changed" is also emitted if another process changes the value.

Most of the <g-conf-client> interface mirrors the functions you'd use to manipulate a <g-conf-engine> (gconf-engine-get and gconf-client-get, for example). These should

all work just like the <g-conf-engine> versions, except that they use the cache from <g-conf-client> and emit the <g-conf-client> signals.

As always with GConf, applications based on <g-conf-client> should use a model-controller-view architecture. Typically, this means that areas of your application affected by a setting will monitor the relevant key and update themselves when necessary. The preferences dialog will simply change keys, allowing GConf to notify the rest of the application that changes have occurred. Here the application proper is the "view," GConf is the "model", and the preferences dialog is the "controller." In no case should you do this: This breaks if a setting is changed *outside* your application—or even from a different part of your application. The correct way (in pseudo-code) is: See the example programs that come with GConf for more details.

```
gconf_client_set(client, key, value);
application_update_to_reflect_setting();

/* At application startup */
gconf_client_notify_add(client, key, application_update_to_reflect_setting, data);

/* From preferences dialog */
gconf_client_set(client, key, value);
```

2.2 Usage

`gconf-client-get-default` ⇒ (*ret* <g-conf-client>) [Function]

Creates a new <g-conf-client> using the default <g-conf-engine>. Normally this is the engine you want. If someone else is already using the default <g-conf-client>, this function returns the same one they're using, but with the reference count incremented. So you have to unref either way.

It's important to call `g-type-init` before using this GObject, to initialize the type system.

ret a new <g-conf-client>. `g-object-unref` when you're done.

`gconf-client-add-dir` (*self* <g-conf-client>) (*dir* `mchars`) [Function]
(*preload* <g-conf-client-preload-type>)

Add a directory to the list of directories the <g-conf-client> will watch. Any changes to keys below this directory will cause the "value_changed" signal to be emitted. When you add the directory, you can request that the <g-conf-client> preload its contents; see <g-conf-client-preload-type> for details.

Added directories may not overlap. That is, if you add "/foo", you may not add "/foo/bar". However you can add "/foo" and "/bar". You can also add "/foo" multiple times; if you add a directory multiple times, it will not be removed until you call `gconf-client-remove-dir` an equal number of times.

client a <g-conf-client>.

dir directory to add to the list.

preload degree of preload.

err the return location for an allocated `<g-error>`, or `#f` to ignore errors.

`gconf-client-remove-dir` (*self* `<g-conf-client>`) (*dir* `mchars`) [Function]
Remove a directory from the list created with `gconf-client-add-dir`. If any notifications have been added below this directory with `gconf-client-notify-add`, those notifications will be disabled until you re-add the removed directory. Note that if a directory has been added multiple times, you must remove it the same number of times before the remove takes effect.

client a `<g-conf-client>`.

dir directory to remove.

err the return location for an allocated `<g-error>`, or `#f` to ignore errors.

`gconf-client-notify-add` (*self* `<g-conf-client>`) [Function]
(*namespace_section* `mchars`) (*proc* `scm`) ⇒ (*ret* `unsigned-int`)
Request notification of changes to *namespace-section*. This includes the key *namespace-section* itself, and any keys below it (the behavior is identical to `gconf-engine-notify-add`, but while `gconf-engine-notify-add` places a notification request on the server for every notify function, `<g-conf-client>` requests server notification for directories added with `gconf-client-add-dir` and keeps the list of `<g-conf-client-notify-func>` on the client side).

For the notification to happen, *namespace-section* must be equal to or below one of the directories added with `gconf-client-add-dir`. You can still call `gconf-client-notify-add` for other directories, but no notification will be received until you add a directory above or equal to *namespace-section*. One implication of this is that `gconf-client-remove-dir` temporarily disables notifications that were below the removed directory.

The function returns a connection ID you can use to call `gconf-client-notify-remove`.

See the description of `<g-conf-client-notify-func>` for details on how the notification function is called.

client a `<g-conf-client>`.

namespace-section
where to listen for changes.

func function to call when changes occur.

user-data user data to pass to *func*.

destroy-notify
function to call on *user-data* when the notify is removed or the `<g-conf-client>` is destroyed, or `#f` for none.

err the return location for an allocated `<g-error>`, or `#f` to ignore errors.

ret a connection ID for removing the notification.

`gconf-client-notify-remove` (*self* `<g-conf-client>`) [Function]
(*cnxn* `unsigned-int`)

Remove a notification using the ID returned from `gconf-client-notify-add`. Invokes the destroy notify function on the notification's user data, if appropriate.

client a `<g-conf-client>`.

cnxn connection ID.

`gconf-client-notify` (*self* `<g-conf-client>`) (*key* `mchars`) [Function]

Emits the "value-changed" signal and notifies listeners as if *key* had been changed

client a `<g-conf-client>`.

key the key that has changed.

Since 2.4.

`gconf-client-set-error-handling` (*self* `<g-conf-client>`) [Function]
(*mode* `<g-conf-client-error-handling-mode>`)

Controls the default error handling for `<g-conf-client>`. See `<g-conf-client-error-handling-mode>` and `<g-conf-client-parent-window-func>` for details on this.

client a `<g-conf-client>`.

mode error handling mode.

`gconf-client-clear-cache` (*self* `<g-conf-client>`) [Function]

Dumps everything out of the `<g-conf-client>` client-side cache. If you know you're done using the `<g-conf-client>` for a while, you can call this function to save some memory.

client a `<g-conf-client>`.

`gconf-client-preload` (*self* `<g-conf-client>`) (*dirname* `mchars`) [Function]
(*type* `<g-conf-client-preload-type>`)

Preloads a directory. Normally you do this when you call `gconf-client-add-dir`, but if you've called `gconf-client-clear-cache` there may be a reason to do it again.

client a `<g-conf-client>`.

dirname directory to preload.

type degree of preload.

err the return location for an allocated `<g-error>`, or `#f` to ignore errors.

`gconf-client-set` (*self* `<g-conf-client>`) (*key* `mchars`) [Function]
(*val* `<g-conf-value>`)

Sets the value of a configuration key. Just like `gconf-engine-set`, but uses `<g-conf-client>` caching and error-handling features. The *val* argument will not be modified.

client a `<g-conf-client>`.

key key to set.

val new value.

err the return location for an allocated `<g-error>`, or `#f` to ignore errors.

`gconf-client-get` (*self* `<g-conf-client>`) (*key* `mchars`) [Function]
⇒ (*ret* `<g-conf-value>`)

Gets the value of a configuration key. Just like `gconf-engine-get`, but uses `<g-conf-client>` caching and error-handling features.

client a `<g-conf-client>`.

key key to get.

err the return location for an allocated `<g-error>`, or `#f` to ignore errors.

ret newly-allocated `<g-conf-value>`, or `#f` if unset and no default exists.

`gconf-client-get-without-default` (*self* `<g-conf-client>`) [Function]
(*key* `mchars`) ⇒ (*ret* `<g-conf-value>`)

Gets the value of a configuration key. Just like `gconf-client-get` but doesn't look for a default value if the key is unset.

client a `<g-conf-client>`.

key key to get.

err the return location for an allocated `<g-error>`, or `#f` to ignore errors.

ret newly-allocated `<g-conf-value>`, or `#f` if unset (even if a default exists).

`gconf-client-unset` (*self* `<g-conf-client>`) (*key* `mchars`) [Function]
⇒ (*ret* `bool`)

Unsets the value of *key*; if *key* is already unset, has no effect. An error of note is '`GCONF_OVERRIDDEN`', indicating that the system administrator has `"forced"` a value for this key. Just like `gconf-engine-unset`, but uses `<g-conf-client>` caching and error-handling features.

client a `<g-conf-client>`.

key key to unset.

err the return location for an allocated `<g-error>`, or `#f` to ignore errors.

ret `#t` on success, `#f` on error.

`gconf-client-recursive-unset` (*self* `<g-conf-client>`) [Function]
(*key* `mchars`) (*flags* `unsigned-int`) ⇒ (*ret* `bool`)

Unsets all keys below *key*, including *key* itself. If any unset fails, continues on to unset as much as it can. The first failure is returned in *err*. Just like `gconf-engine-recursive-unset`, but uses `<g-conf-client>` caching and error-handling features.

client a `<g-conf-client>`.

key a key or directory name to be unset.

flags change how the unset is done.

err the return location for an allocated `<g-error>`, or `#f` to ignore errors.

ret `#t` on success, `#f` on error.

Since 2.4.

`gconf-client-all-dirs` (*self* `<g-conf-client>`) (*dir* `mchars`) [Function]
⇒ (*ret* `gslist-of`)

Lists the subdirectories in *dir*. The returned list contains allocated strings. Each string is the absolute path of a subdirectory. You should `g-free` each string in the list, then `g-slist-free` the list itself. Just like `gconf-engine-all-dirs`, but uses `<g-conf-client>` caching and error-handling features.

client a `<g-conf-client>`.

dir directory to get subdirectories from.

err the return location for an allocated `<g-error>`, or `#f` to ignore errors.

ret List of allocated subdirectory names.

`gconf-client-suggest-sync` (*self* `<g-conf-client>`) [Function]

Suggests to *gconfd* that you've just finished a block of changes, and it would be an optimal time to sync to permanent storage. This is only a suggestion; and *gconfd* will eventually sync even if you don't call `gconf-engine-suggest-sync`. This function is just a "hint" provided to *gconfd* to maximize efficiency and minimize data loss. Just like `gconf-engine-suggest-sync`.

client a `<g-conf-client>`.

err the return location for an allocated `<g-error>`, or `#f` to ignore errors.

`gconf-client-dir-exists` (*self* `<g-conf-client>`) (*dir* `mchars`) [Function]
⇒ (*ret* `bool`)

Queries whether the directory *dir* exists in the GConf database. Returns `#t` or `#f`. Just like `gconf-engine-dir-exists`, but uses `<g-conf-client>` caching and error-handling features.

client a `<g-conf-client>`.

dir directory to check for

err the return location for an allocated `<g-error>`, or `#f` to ignore errors.

ret `#t` or `#f`.

`gconf-client-key-is-writable` (*self* `<g-conf-client>`) [Function]
(*key* `mchars`) ⇒ (*ret* `bool`)

Checks whether the key is writable.

client a `<g-conf-client>`.

key the value to be changed.

err the return location for an allocated `<g-error>`, or `#f` to ignore errors.

ret `#t` if the key is writable, `#f` if the key is read only.

`gconf-client-value-changed` (*self* `<g-conf-client>`) (*key* `mchars`) (*value* `<g-conf-value>`) [Function]

Emits the "value_changed" signal. Rarely useful.

client a `<g-conf-client>`.

key key to pass to signal handlers.

value value of *key* to pass to signal handlers.

3 GConf Core Interfaces

Basic functions to initialize GConf and get/set values

3.1 Overview

These functions initialize GConf, and communicate with the server via a `<g-conf-engine>` object. You can install a notification request on the server, get values, set values, list directories, and associate schema names with keys.

Most of this interface is replicated in the `<gobject>` wrapper (`<g-conf-client>` object); an alternative to the value-setting functions is the `<g-conf-change-set>` interface.

3.2 Usage

`gconf-valid-key` (*key* `mchars`) ⇒ (*ret* `bool`) (*why_invalid* `mchars`) [Function]

Asks whether a key is syntactically correct, that is, it ensures that the key consists of slash-separated strings and contains only legal characters. Normally you shouldn't need to call this function; the GConf functions all check this for you and return an error if the key is invalid. However, it may be useful to validate input to an entry field or the like. If you pass a non-`#f` address as the *why-invalid* argument, an allocated string is returned explaining why the key is invalid, if it is. If the key is valid the *why-invalid* argument is unused.

key key to check.

why-invalid
return location for an explanation of the problem, if any. `g-free` the returned string.

ret `#t` if the key is valid, or `#f` if not.

`gconf-key-is-below` (*above* `mchars`) (*below* `mchars`) ⇒ (*ret* `bool`) [Function]

Asks whether the key *below* would be found below the key *above*, were they both to exist in the database. For example, `/foo` is always found below `/` and above `/foo/bar`. This probably isn't useful but GConf uses it internally so here it is if you need it.

above the key on the "left hand side" of the predicate.

below the key on the "right hand side."

ret `#t` or `#f`.

`gconf-concat-dir-and-key` (*dir* `mchars`) (*key* `mchars`) [Function]
⇒ (*ret* `mchars`)

Concatenates the dir and key passed removing the unnecessary '/' characters and returns the new string.

dir the directory.

key the key.

ret the newly concatenated string.

`gconf-unique-key` ⇒ (*ret* `mchars`) [Function]

Generates a new and unique key using serial number, process id, current time and a random number generated.

ret a newly created key, a `<gchar>` value.

`gconf-escape-key` (*arbitrary_text* `mchars`) (*len* `int`) ⇒ (*ret* `mchars`) [Function]

Escape *arbitrary-text* such that it's a valid key element (i.e. one part of the key path). The escaped key won't pass `gconf-valid-key` because it isn't a whole key (i.e. it doesn't have a preceding slash), but prepending a slash to the escaped text should always result in a valid key.

arbitrary-text
some text in any encoding or format

len length of *arbitrary-text* in bytes, or -1 if *arbitrary-text* is nul-terminated

ret a nul-terminated valid GConf key

`gconf-unescape-key` (*escaped_key* `mchars`) (*len* `int`) ⇒ (*ret* `mchars`) [Function]

Converts a string escaped with `gconf-escape-key` back into its original form.

escaped-key
a key created with `gconf-escape-key`

len length of *escaped-key* in bytes, or -1 if *escaped-key* is nul-terminated

ret the original string that was escaped to create *escaped-key*

4 GConfChangeSet

a set of configuration changes to be made.

4.1 Overview

a `<g-conf-change-set>` allows you to collect a set of changes to configuration keys (set/unset operations). You can then commit all the changes at once. This is convenient for something like a preferences dialog; you can collect all the pending changes in a `<g-conf-change-set>`, then when the user clicks "apply" send them all to the configuration database. The `<g-conf-change-set>` allows you to avoid sending every preferences setting when "apply" is clicked; you only have to send the settings the user changed.

In the future, GConf may also have optimizations so that changing a group of values with `<g-conf-change-set>` is faster than calling `gconf-engine-set` for each value. In the future, `<g-conf-change-set>` may also represent an atomic transaction, where all or none of the values are set; however, for now the operation is *not* atomic.

4.2 Usage

5 GConfEngine

a GConf "database"

5.1 Overview

A `<g-conf-engine>` represents a connection to the GConf database. The default `<g-conf-engine>`, returned from `gconf-engine-get-default`, represents the user's normal configuration source search path. Configuration-related utilities, such as a configuration editor tool, might wish to access a particular configuration source directly; they can obtain a non-default `<g-conf-engine>` with `gconf-engine-get-for-address`.

Once you have a `<g-conf-engine>`, you can query and manipulate configuration values.

5.2 Usage

6 GError

error reporting.

6.1 Overview

The <g-error> object is used to report errors that occur in GConf library routines. All functions that report errors work the same way:

The last argument to the function is a <g-error>**, a pointer to a location where a <g-error>* can be placed.

This last argument may be , in which case no error will be returned.

If non-, the argument should be the address of a <g-error>* variable, which should be initialized to .

If an error occurs, a <g-error> will be allocated and placed in the return location; the caller must free the <g-error> with g-error-free. If no error occurs, the return location will be left untouched. That is, the test 'error != NULL' should always be a reliable indicator of whether the operation failed.

It's also common that the return value of a function indicates whether or not an error occurred. Typically, is returned on success. In some cases, a return value indicates failure. Either way, if the return value indicates failure and you passed a non- value for the last argument to the function, a <g-error> will be returned. If the return value indicates success, then a <g-error> will never be returned. These relationships are guaranteed; that is, you can reliably use the return value to decide whether a <g-error> was placed in the return location. If a function does *not* indicate success/failure by return value, you must check whether the <g-error> is to detect errors.

Here's a short error handling example:

```
GError* err = NULL;

if (!gconf_init(&err))
  {
    fprintf(stderr, _("Failed to init GConf: %s\n"), err->message);
    g_error_free(err);
    err = NULL;
  }
```

6.2 Usage

7 GConfSchema

A describes a

7.1 Overview

A "schema" describes a key-value pair in a GConf database. It may include information such as default value and value type, as well as documentation describing the pair, the name of the application that created the pair, etc.

A <g-conf-schema> duplicates some of the information about the value it describes, such as type information. In these cases, the type information provided describes what the type of the value *should be*, not what the type actually is.

7.2 Usage

<g-conf-schema> [Class]

Derives from <gboxed>.

This class defines no direct slots.

gconf-schema-new ⇒ (*ret* <g-conf-schema>) [Function]

Creates a new <g-conf-schema>.

ret newly allocated <g-conf-schema>

gconf-schema-get-locale (*self* <g-conf-schema>) ⇒ (*ret* mchars) [Function]

Returns the locale for a <g-conf-schema>. The returned string is *not* a copy, so don't try to free it. It is "owned" by the <g-conf-schema> and will be destroyed when the <g-conf-schema> is destroyed.

schema a <g-conf-schema>

ret the locale

gconf-schema-get-short-desc (*self* <g-conf-schema>) [Function]
⇒ (*ret* mchars)

Returns the short description for a <g-conf-schema>. The returned string is *not* a copy, don't try to free it. It is "owned" by the <g-conf-schema> and will be destroyed when the <g-conf-schema> is destroyed.

schema a <g-conf-schema>.

ret the short description.

gconf-schema-get-long-desc (*self* <g-conf-schema>) [Function]
⇒ (*ret* mchars)

Returns the long description for a <g-conf-schema>. The returned string is *not* a copy, don't try to free it. It is "owned" by the <g-conf-schema> and will be destroyed when the <g-conf-schema> is destroyed.

schema a <g-conf-schema>

ret the long description.

`gconf-schema-get-owner` (*self* `<g-conf-schema>`) ⇒ (*ret* `mchars`) [Function]

Returns the owner of a `<g-conf-schema>`. The returned string is *not* a copy, don't try to free it. It is "owned" by the `<g-conf-schema>` and will be destroyed when the `<g-conf-schema>` is destroyed.

schema a `<g-conf-schema>`.

ret the owner.

`gconf-schema-get-default-value` (*self* `<g-conf-schema>`) [Function]
⇒ (*ret* `<g-conf-value>`)

Returns the default value of the entry that is described by a `<g-conf-schema>`.

schema a `<g-conf-schema>`.

ret the default value of the entry.

`gconf-schema-get-car-type` (*self* `<g-conf-schema>`) [Function]
⇒ (*ret* `<g-conf-value-type>`)

Returns the default type of the first member of the pair in the entry (which should be of type '`GCONF_VALUE_PAIR`') described by *schema*.

schema a `<g-conf-schema>`.

ret the type of the first member of the pair element of the entry.

`gconf-schema-get-cdr-type` (*self* `<g-conf-schema>`) [Function]
⇒ (*ret* `<g-conf-value-type>`)

Returns the default type of the second member of the pair in the entry (which should be of type '`GCONF_VALUE_PAIR`') described by *schema*.

schema a `<g-conf-schema>`.

ret the type of the second member of the pair element of the entry.

`gconf-schema-get-list-type` (*self* `<g-conf-schema>`) [Function]
⇒ (*ret* `<g-conf-value-type>`)

Returns the default type of the list elements of the entry (which should be of default type '`GCONF_VALUE_LIST`') described by *schema*.

schema

ret

`gconf-schema-set-type` (*self* `<g-conf-schema>`) [Function]
(*type* `<g-conf-value-type>`)

Sets the `<g-conf-value-type>` of the `<g-conf-schema>` to *type*.

sc a `<g-conf-schema>`.

type the type.

`gconf-schema-set-locale` (*self* `<g-conf-schema>`) (*locale* `mchars`) [Function]

Sets the locale for a `<g-conf-schema>` to *locale*. *locale* is copied.

sc a `<g-conf-schema>`.

locale the locale.

`gconf-schema-set-short-desc` (*self* `<g-conf-schema>`) (*desc* `mchars`) [Function]

Sets the short description of a `<g-conf-schema>` to *desc*. *desc* is copied.

sc a `<g-conf-schema>`.

desc the short description.

`gconf-schema-set-long-desc` (*self* `<g-conf-schema>`) (*desc* `mchars`) [Function]

Sets the long description of a `<g-conf-schema>` to *desc*. *desc* is copied.

sc a `<g-conf-schema>`.

desc the long description.

`gconf-schema-set-owner` (*self* `<g-conf-schema>`) (*owner* `mchars`) [Function]

Sets the "owner" of the `<g-conf-schema>`, where the owner is the name of the application that created the entry.

sc a `<g-conf-schema>`.

owner the name of the creating application.

`gconf-schema-set-default-value` (*self* `<g-conf-schema>`) (*val* `<g-conf-value>`) [Function]

Sets the default value for the entry described by the `<g-conf-schema>`. The `<g-conf-value>` is copied. Alternatively, use `gconf-schema-set-default-value-nocopy`.

sc a `<g-conf-schema>`.

val the default value.

`gconf-schema-set-car-type` (*self* `<g-conf-schema>`) (*type* `<g-conf-value-type>`) [Function]

Sets the `<g-conf-value-type>` of the first member (car) of the entry (which should be of type '`GCONF_VALUE_PAIR`') described by `<g-conf-schema>` to *type*.

sc a `<g-conf-schema>`.

type the type.

`gconf-schema-set-cdr-type` (*self* `<g-conf-schema>`) (*type* `<g-conf-value-type>`) [Function]

Sets the `<g-conf-value-type>` of the second member (cdr) of the entry (which should be of type '`GCONF_VALUE_PAIR`') described by `<g-conf-schema>` to *type*.

sc a `<g-conf-schema>`.

type the type.

`gconf-schema-set-list-type` (*self* `<g-conf-schema>`) (*type* `<g-conf-value-type>`) [Function]

Sets the `<g-conf-value-type>` of the list elements of the entry (which should be of type '`GCONF_VALUE_LIST`') described by `<g-conf-schema>` to *type*.

sc a `<g-conf-schema>`.

type the type.

8 GConfValue, GConfEntry, GConfMetaInfo

A stores a dynamically-typed value. A stores a key-value pair. A stores metainformation about a key.

8.1 Overview

<g-conf-value> stores one of the value types GConf understands; GConf uses <g-conf-value> to pass values around because it doesn't know the type of its values at compile time.

A <g-conf-entry> pairs a relative key name with a value, for example if the value "10" is stored at the key "/foo/bar/baz", the <g-conf-entry> will store "baz" and "10".

A <g-conf-meta-info> object holds metainformation about a key, such as its last modification time and the name of the schema associated with it. You should rarely if ever need to use <g-conf-meta-info>. (In fact you can't get the metainfo for a key using the current API.)

8.2 Usage

<g-conf-value> [Class]

Derives from <gboxed>.

This class defines no direct slots.

9 Undocumented

The following symbols, if any, have not been properly documented.

9.1 (gnome gw gconf)

`gconf-client-get-default-from-schema` [Function]

`gconf-schema-set-default-value-nocopy` [Variable]

Type Index

<g-conf-schema> 14
<g-conf-value> 17

Function Index

gconf-client-add-dir 3
gconf-client-all-dirs 7
gconf-client-clear-cache 5
gconf-client-dir-exists 7
gconf-client-get 6
gconf-client-get-default 3
gconf-client-get-default-from-schema 18
gconf-client-get-without-default 6
gconf-client-key-is-writable 7
gconf-client-notify 5
gconf-client-notify-add 4
gconf-client-notify-remove 5
gconf-client-preload 5
gconf-client-recursive-unset 6
gconf-client-remove-dir 4
gconf-client-set 5
gconf-client-set-error-handling 5
gconf-client-suggest-sync 7
gconf-client-unset 6
gconf-client-value-changed 8
gconf-concat-dir-and-key 9
gconf-escape-key 10
gconf-key-is-below 9
gconf-schema-get-car-type 15
gconf-schema-get-cdr-type 15
gconf-schema-get-default-value 15
gconf-schema-get-list-type 15
gconf-schema-get-locale 14
gconf-schema-get-long-desc 14
gconf-schema-get-owner 15
gconf-schema-get-short-desc 14
gconf-schema-new 14
gconf-schema-set-car-type 16
gconf-schema-set-cdr-type 16
gconf-schema-set-default-value 16
gconf-schema-set-list-type 16
gconf-schema-set-locale 15
gconf-schema-set-long-desc 16
gconf-schema-set-owner 16
gconf-schema-set-short-desc 16
gconf-schema-set-type 15
gconf-unescape-key 10
gconf-unique-key 10
gconf-valid-key 9