Guile-GNOME: GNOME-VFS

version 2.16.2, updated 9 December 2011

The Free Software Foundation
Christophe Fergeau
Christian Neumair

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

Short Contents

1 Overview

`(gnome vfs)` wraps the GNOME Virtual Filesystem library for Guile. It is a part of Guile-GNOME.

The basic idea in this module is to map, as transparently as possible, GNOME-VFS files onto Scheme ports. Once you have a Scheme port, the native Scheme API is the preferred API. Other GNOME-VFS library procedures are defined that have no counterpart on the Scheme level, such as the MIME database procedures.

So, for example, to write to a file over SCP, you might do this:

```
(define (with-output-to-port/dynamic make-port thunk)
  (let ((port #f))
    (dynamic-wind (lambda () (set! port (make-port)))
                  (lambda () (with-output-to-port port thunk))
                  (lambda () (close-port port) (set! port #f)))))

(define (make-output-port uri-string exclusive?)
  (gnome-vfs-create
   (gnome-vfs-make-uri-from-input uri-string)
   'write exclusive? #o644))

(with-output-to-port/dynamic
    (lambda ()
      (make-output-port "sftp://me@example.com/tmp/foo.txt" #t))
  (lambda ()
    (write "Hello world!")))
```

The `dynamic-wind` trickery is to ensure that the port is closed after execution leaves the thunk, and not left for GC to close in the future.

Exceptions raised during I/O are thrown to the `gnome-vfs-error` key, with the second argument being a symbol corresponding to a particular `<gnome-vfs-result>` value, such as `error-file-exists`.

To enable integration with the GNOME keyring, for SSH keys and the like, you will need to call `(gnome-authentication-manager-init)`, which is a procedure defined in the `(gnome gnome-ui)` library.

This manual is admittedly a bit incomplete. Patches are accepted, of course, but the best thing to do would be eventually to wrap GIO, the new VFS layer that was pushed down into GLib.

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

2 GnomeVFSResult

Result of I/O operations, the equivalent of errno

2.1 Overview

2.2 Usage

`gnome-vfs-result-to-string` (*res* `<gnome-vfs-result>`) [Function]
⇒ (*ret* `mchars`)

Returns a string representing *result*, useful for debugging purposes, but probably not appropriate for passing to the user.

result a `<gnome-vfs-result>` to convert to a string.

ret a string representing *result*.

`gnome-vfs-result-from-errno-code` (*errno_code* `int`) [Function]
⇒ (*ret* `<gnome-vfs-result>`)

Converts a system errno value to a `<gnome-vfs-result>`.

errno-code
integer of the same type as the system "errno".

ret a `<gnome-vfs-result>` equivalent to *errno-code*.

`gnome-vfs-result-from-errno` ⇒ (*ret* `<gnome-vfs-result>`) [Function]

Converts the system errno to a `<gnome-vfs-result>`.

ret a `<gnome-vfs-result>` equivalent to the current system errno.

`gnome-vfs-result-from-h-errno` ⇒ (*ret* `<gnome-vfs-result>`) [Function]

Converts the system "h_errno" to a `<gnome-vfs-result>` (h_errno represents errors accessing and finding internet hosts)

ret a `<gnome-vfs-result>` equivalent to the current system "h_errno".

`gnome-vfs-result-from-h-errno-val` (*h_errno_code* `int`) [Function]
⇒ (*ret* `<gnome-vfs-result>`)

Converts the error code *h-errno-code* into a `<gnome-vfs-result>`.

h-errno-code
an integer representing the same error code as the system h_errno.

ret The `<gnome-vfs-result>` equivalent to the *h-errno-code*.

3 GnomeVFSURI

A uniform resource identifier.

3.1 Overview

A `<gnome-vfsuri>` is a semi-textual representation of a uniform resource identifier. It contains the information about a resource location encoded as canononicalized text, but also holds extra information about the context in which the URI is used.

3.2 Usage

`<gnome-vfsuri>` [Class]

Opaque pointer.

This class defines no direct slots.

`gnome-vfs-uri-new` (*text_uri* `mchars`) ⇒ (*ret* `<gnome-vfsuri>`) [Function]

Create a new uri from *text-uri*. Unsupported and unsafe methods are not allowed and will result in '`#f`' being returned. URL transforms are allowed.

The a text_uri must be an escaped URI string such as returned by `gnome-vfs-get-uri-from-local-path`, `gnome-vfs-make-uri-from-input`, or `gtk-file-chooser-get-uri`.

text-uri an escaped string representing a uri.

ret The new uri.

`gnome-vfs-uri-resolve-relative` (*self* `<gnome-vfsuri>`) [Function]
(*relative_reference* `mchars`) ⇒ (*ret* `<gnome-vfsuri>`)

Create a new uri from *relative-reference*, relative to *base*. The resolution algorithm in some aspects follows RFC 2396, section 5.2, but is not identical due to some extra assumptions GnomeVFS makes about URIs.

If *relative-reference* begins with a valid scheme identifier followed by ':', it is assumed to refer to an absolute URI, and a `<gnome-vfsuri>` is created from it using `gnome-vfs-uri-new`.

Otherwise, depending on its precise syntax, it inherits some aspects of the parent URI, but the parents' fragment and query components are ignored.

If *relative-reference* begins with `"//"`, it only inherits the *base* scheme, if it begins with '/' (i.e. is an absolute path reference), it inherits everything ecxept the *base* path. Otherwise, it replaces the part of *base* after the last '/'.

> This function should not be used by application authors unless they expect very distinct semantics. Instead, authors should use `gnome-vfs-uri-append-file-name`, `gnome-vfs-uri-append-path`, `gnome-vfs-uri-append-string` or `gnome-vfs-uri-resolve-symbolic-link`.

base base uri.

relative-reference
: a string representing a possibly relative uri reference.

ret
: A `<gnome-vfsuri>` referring to *relative-reference*, or '`#f`' if *relative-reference* was malformed.

`gnome-vfs-uri-resolve-symbolic-link` (*self* `<gnome-vfsuri>`) (*relative_reference* `mchars`) ⇒ (*ret* `<gnome-vfsuri>`) [Function]

Create a new uri from *symbolic-link*, relative to *base*.

If *symbolic-link* begins with a '/', it replaces the path of *base*, otherwise it is appended after the last '/' character of *base*.

base
: base uri.

relative-reference

ret
: A new `<gnome-vfsuri>` referring to *symbolic-link*.

Since 2.16

`gnome-vfs-uri-append-string` (*self* `<gnome-vfsuri>`) (*uri_fragment* `mchars`) ⇒ (*ret* `<gnome-vfsuri>`) [Function]

Create a new uri obtained by appending *uri-fragment* to *uri*. This will take care of adding an appropriate directory separator between the end of *uri* and the start of *uri-fragment* if necessary.

uri
: a `<gnome-vfsuri>`.

uri-fragment
: a piece of a uri (ie a fully escaped partial path).

ret
: The new uri obtained by combining *uri* and *uri-fragment*.

`gnome-vfs-uri-append-path` (*self* `<gnome-vfsuri>`) (*path* `mchars`) ⇒ (*ret* `<gnome-vfsuri>`) [Function]

Create a new uri obtained by appending *path* to *uri*. This will take care of adding an appropriate directory separator between the end of *uri* and the start of *path* if necessary as well as escaping *path* as necessary.

uri
: a `<gnome-vfsuri>`.

path
: a non-escaped file path.

ret
: The new uri obtained by combining *uri* and *path*.

`gnome-vfs-uri-append-file-name` (*self* `<gnome-vfsuri>`) (*filename* `mchars`) ⇒ (*ret* `<gnome-vfsuri>`) [Function]

Create a new uri obtained by appending *file-name* to *uri*. This will take care of adding an appropriate directory separator between the end of *uri* and the start of *file-name* if necessary. *file-name* might, for instance, be the result of a call to `g-dir-read-name`.

uri
: a `<gnome-vfsuri>`.

filename
: any "regular" file name (can include #, /, etc) in the file system encoding. This is not an escaped URI.

ret
: The new uri obtained by combining *uri* and *path*.

`gnome-vfs-uri-to-string` (*self* `<gnome-vfsuri>`) [Function]
(*hide_options* `<gnome-vfsuri-hide-options>`) ⇒ (*ret* `mchars`)

Translate *uri* into a printable string. The string will not contain the uri elements specified by *hide-options*.

A file: URI on Win32 might look like file:///x:/foo/bar.txt. Note that the part after file:// is not a legal file name, you need to remove the / in front of the drive letter. This function does that automatically if *hide-options* specifies that the toplevel method, user name, password, host name and host port should be hidden.

On the other hand, a file: URI for a UNC path looks like file:////server/share/foo/bar.txt, and in that case the part after file:// is the correct file name.

uri a `<gnome-vfsuri>`.

hide-options
bitmask specifying what uri elements (e.g. password, user name etc.) should not be represented in the returned string.

ret a malloc'd printable string representing *uri*.

`gnome-vfs-uri-dup` (*self* `<gnome-vfsuri>`) ⇒ (*ret* `<gnome-vfsuri>`) [Function]

Duplicate *uri*.

uri a `<gnome-vfsuri>`.

ret a pointer to a new uri that is exactly the same as *uri*.

`gnome-vfs-uri-is-local` (*self* `<gnome-vfsuri>`) ⇒ (*ret* `bool`) [Function]

Check if *uri* is a local URI. Note that the return value of this function entirely depends on the `<gnome-vfs-method>` associated with the URI. It is up to the method author to distinguish between remote URIs add URIs referring to entities on the local computer.

Warning, this can be slow, as it does i/o to detect things like NFS mounts.

uri a `<gnome-vfsuri>`.

ret '`#t`' if *uri* refers to a local entity, '`#f`' otherwise.

`gnome-vfs-uri-has-parent` (*self* `<gnome-vfsuri>`) ⇒ (*ret* `bool`) [Function]

Check if uri has a parent or not.

uri a `<gnome-vfsuri>`.

ret '`#t`' if *uri* has a parent, '`#f`' otherwise.

`gnome-vfs-uri-get-parent` (*self* `<gnome-vfsuri>`) [Function]
⇒ (*ret* `<gnome-vfsuri>`)

Retrieve *uri*'s parent uri.

uri a `<gnome-vfsuri>`.

ret a pointer to *uri*'s parent uri.

`gnome-vfs-uri-get-host-name` (*self* `<gnome-vfsuri>`) [Function]
⇒ (*ret* `mchars`)

Retrieve the host name for *uri*.

uri a `<gnome-vfsuri>`.

ret a string representing the host name.

`gnome-vfs-uri-get-scheme` (*self* `<gnome-vfsuri>`) ⇒ (*ret* `mchars`) [Function]
Retrieve the scheme used for *uri*.

uri a `<gnome-vfsuri>`.

ret a string representing the scheme.

`gnome-vfs-uri-get-host-port` (*self* `<gnome-vfsuri>`) [Function]
⇒ (*ret* `unsigned-int`)
Retrieve the host port number in *uri*.

uri a `<gnome-vfsuri>`.

ret The host port number used by *uri*. If the value is zero, the default port value for the specified toplevel access method is used.

`gnome-vfs-uri-get-user-name` (*self* `<gnome-vfsuri>`) [Function]
⇒ (*ret* `mchars`)
Retrieve the user name in *uri*.

uri a `<gnome-vfsuri>`.

ret a string representing the user name in *uri*.

`gnome-vfs-uri-get-password` (*self* `<gnome-vfsuri>`) [Function]
⇒ (*ret* `mchars`)
Retrieve the password for *uri*.

uri a `<gnome-vfsuri>`.

ret The password for *uri*.

`gnome-vfs-uri-set-host-name` (*self* `<gnome-vfsuri>`) [Function]
(*host_name* `mchars`)
Set *host-name* as the host name accessed by *uri*.

uri a `<gnome-vfsuri>`.

host-name a string representing a host name.

`gnome-vfs-uri-set-host-port` (*self* `<gnome-vfsuri>`) [Function]
(*host_port* `unsigned-int`)
Set the host port number in *uri*. If *host-port* is zero, the default port for *uri*'s toplevel access method is used.

uri a `<gnome-vfsuri>`.

host-port a TCP/IP port number.

`gnome-vfs-uri-set-user-name` (*self* `<gnome-vfsuri>`) [Function]
(*user_name* `mchars`)
Set *user-name* as the user name for *uri*.

uri a `<gnome-vfsuri>`.

user-name a string representing a user name on the host accessed by *uri*.

`gnome-vfs-uri-set-password` (*self* `<gnome-vfsuri>`) (*password* `mchars`) [Function]

Set *password* as the password for *uri*.

uri a `<gnome-vfsuri>`.

password a password string.

`gnome-vfs-uri-equal` (*self* `<gnome-vfsuri>`) (*b* `<gnome-vfsuri>`) ⇒ (*ret* `bool`) [Function]

Compare *a* and *b*.

FIXME: This comparison should take into account the possiblity that unreserved characters may be escaped. ...or perhaps `gnome-vfs-uri-new` should unescape unreserved characters?

a a `<gnome-vfsuri>`.

b a `<gnome-vfsuri>`.

ret '`#t`' if *a* and *b* are equal, '`#f`' otherwise.

`gnome-vfs-uri-is-parent` (*self* `<gnome-vfsuri>`) (*possible_child* `<gnome-vfsuri>`) (*recursive* `bool`) ⇒ (*ret* `bool`) [Function]

Check if *possible-child* is contained by *possible-parent*. If *recursive* is '`#f`', just try the immediate parent directory, else search up through the hierarchy.

possible-parent
a `<gnome-vfsuri>`.

possible-child
a `<gnome-vfsuri>`.

recursive a flag to turn recursive check on.

ret '`#t`' if *possible-child* is contained in *possible-parent*.

`gnome-vfs-uri-get-path` (*self* `<gnome-vfsuri>`) ⇒ (*ret* `mchars`) [Function]

Retrieve full path name for *uri*.

uri a `<gnome-vfsuri>`.

ret a pointer to the full path name in *uri*. Notice that the pointer points to the path name stored in *uri*, so the path name returned must not be modified nor freed.

`gnome-vfs-uri-extract-dirname` (*self* `<gnome-vfsuri>`) ⇒ (*ret* `mchars`) [Function]

Extract the name of the directory in which the file pointed to by *uri* is stored as a newly allocated string. The string will end with a '`GNOME_VFS_URI_PATH_CHR`'.

uri a `<gnome-vfsuri>`.

ret a pointer to the newly allocated string representing the parent directory.

`gnome-vfs-uri-extract-short-name` (*self* `<gnome-vfsuri>`) [Function]
⇒ (*ret* `mchars`)

Retrieve base file name for *uri*, ignoring any trailing path separators. This matches the XPG definition of basename, but not g_basename. This is often useful when you want the name of something that's pointed to by a uri, and don't care whether the uri has a directory or file form. If *uri* points to the root of a domain, returns the host name. If there's no host name, returns '`GNOME_VFS_URI_PATH_STR`'.

See also: `gnome-vfs-uri-extract-short-path-name`.

uri a `<gnome-vfsuri>`.

ret a pointer to the newly allocated string representing the unescaped short form of the name.

`gnome-vfs-uri-list-parse` (*uri_list* `mchars`) ⇒ (*ret* `glist-of`) [Function]

Extracts a list of `<gnome-vfsuri>` objects from a standard text/uri-list, such as one you would get on a drop operation. Use `gnome-vfs-uri-list-free` when you are done with the list.

uri-list string consists of `<gnome-vfsur-is>` and/or paths seperated by newline character.

ret a `<g-list>` of `<gnome-vfsur-is>`.

4 gnome-vfs-utils

various utilities functions to manipulate uris

4.1 Overview

4.2 Usage

`gnome-vfs-format-uri-for-display` (*uri* `mchars`) ⇒ (*ret* `mchars`) [Function]
Filter, modify, unescape and change *uri* to make it appropriate for display to users. The conversion is done such that the roundtrip to UTF-8 is reversible.

Rules: file: uri without fragments should appear as local paths. file: uri with fragments should appear as file:uri. All other uri appear as expected.

uri uri to be displayed.

ret a string which represents *uri* and can be displayed.

Since 2.2

`gnome-vfs-url-show` (*url* `mchars`) ⇒ (*ret* `<gnome-vfs-result>`) [Function]
Launches the default application or component associated with the given *url*.

url url to be shown.

ret '`GNOME_VFS_OK`' if the default action was launched, '`GNOME_VFS_ERROR_BAD_PARAMETERS`' for an invalid or non-existant *url*, '`GNOME_VFS_ERROR_NO_DEFAULT`' if no default action is associated with the *url*. Also error codes from `gnome-vfs-mime-action-launch` or `gnome-vfs-mime-action-launch-with-env`.

Since 2.4

`gnome-vfs-escape-string` (*string* `mchars`) ⇒ (*ret* `mchars`) [Function]
Escapes *string*, replacing any and all special characters with equivalent escape sequences.

string string to be escaped.

ret a newly allocated string equivalent to *string* but with all special characters escaped.

`gnome-vfs-escape-path-string` (*path* `mchars`) ⇒ (*ret* `mchars`) [Function]
Escapes *path*, replacing only special characters that would not be found in paths (so '/', '&', and '=' will not be escaped by this function).

path string to be escaped.

ret a newly allocated string equivalent to *path* but with non-path characters escaped.

`gnome-vfs-escape-slashes` (*string* `mchars`) ⇒ (*ret* `mchars`) [Function]

Escapes only '/' and '%' characters in *string*, replacing them with their escape sequence equivalents.

string string to be escaped.

ret a newly allocated string equivalent to *string*, but with no unescaped '/' or '%' characters.

`gnome-vfs-escape-set` (*string* `mchars`) (*match_set* `mchars`) ⇒ (*ret* `mchars`) [Function]

Escapes all characters in *string* which are listed in *match-set*.

string string to be escaped.

match-set a string containing all characters to be escaped in *string*.

ret a newly allocated string equivalent to *string* but with characters in *match-string* escaped.

`gnome-vfs-unescape-string` (*escaped_string* `mchars`) (*illegal_characters* `mchars`) ⇒ (*ret* `mchars`) [Function]

Decodes escaped characters (i.e. PERCENTxx sequences) in *escaped-string*. Characters are encoded in PERCENTxy form, where xy is the ASCII hex code for character 16x+y.

escaped-string
an escaped uri, path, or other string.

illegal-characters
a string containing a sequence of characters considered "illegal" to be escaped, '\0' is automatically in this list.

ret a newly allocated string with the unescaped equivalents, or '`#f`' if *escaped-string* contained an escaped encoding of one of the characters in *illegal-characters*.

`gnome-vfs-make-uri-canonical` (*uri* `mchars`) ⇒ (*ret* `mchars`) [Function]

Standardizes the format of the *uri*, so that it can be used later in other functions that expect a canonical uri.

uri an absolute or relative stringified uri. It might have scheme.

ret a newly allocated string that contains the canonical representation of *uri*.

Since 2.2

`gnome-vfs-make-path-name-canonical` (*path* `mchars`) ⇒ (*ret* `mchars`) [Function]

Calls `-gnome-vfs-canonicalize-pathname`, allocating storage for the result and providing for a cleaner memory management.

path a file path, relative or absolute.

ret a canonical version of *path*.

`gnome-vfs-make-uri-from-input` (*location* `mchars`) ⇒ (*ret* `mchars`) [Function]

Takes a user input path/uri and makes a valid uri out of it.

This function is the reverse of `gnome-vfs-format-uri-for-display` but it also handles the fact that the user could have typed arbitrary UTF-8 in the entry showing the string.

location a possibly mangled "uri", in UTF-8.

ret a newly allocated uri.

Since 2.2

`gnome-vfs-make-uri-from-shell-arg` (*uri* `mchars`) ⇒ (*ret* `mchars`) [Function]

Similar to `gnome-vfs-make-uri-from-input`, except that:

1) guesses relative paths instead of http domains. 2) doesn't bother stripping leading/trailing white space. 3) doesn't bother with ~ expansion–that's done by the shell.

uri path to make the uri from.

ret a newly allocated string representing *uri*.

Since 2.2

`gnome-vfs-expand-initial-tilde` (*path* `mchars`) ⇒ (*ret* `mchars`) [Function]

If *path* starts with a ~, representing the user's home directory, expand it to the actual path location.

path a local file path which may start with a '~'.

ret a newly allocated string with the initial tilde (if there was one) converted to an actual path.

`gnome-vfs-get-local-path-from-uri` (*uri* `mchars`) ⇒ (*ret* `mchars`) [Function]

Create a local path for a file:/// uri. Do not use with uris of other methods.

uri uri to convert to a local path.

ret a newly allocated string containing the local path. '`#f`' is returned on error or if the uri isn't a file: uri without a fragment identifier (or chained uri).

`gnome-vfs-get-uri-from-local-path` (*local_full_path* `mchars`) [Function]
⇒ (*ret* `mchars`)

Returns a file:/// URI for the local path *local-full-path*, such as a path provided by `gtk-file-chooser-get-filename`. The resulting URI may be provided, for instance, to `gnome-vfs-uri-new`.

On Windows *local-full-path* should be in the UTF-8 encoding, and can start with a drive letter, but doesn't have to.

local-full-path
a full local filesystem path (i.e. not relative).

ret a newly allocated string containing the uri corresponding to *local-full-path* ('`#f`' for some bad errors).

`gnome-vfs-icon-path-from-filename` (*filename* `mchars`) [Function]
⇒ (*ret* `mchars`)

filename path to a file. Could be relative or absolute path.

ret Returns the icon path for the *filename*. Example: gnome_vfs_icon_path_from_filename ("nautilus/nautilus-desktop.png") will return a string forming the full path of the file nautilus-desktop.png ie $PREFIX/share/pixmaps/nautilus/nautilus-desktop.png.

`gnome-vfs-is-primary-thread` ⇒ (*ret* `bool`) [Function]
Check if the current thread is the thread with the main glib event loop.

ret '`#t`' if the current thread is the thread with the main glib event loop.

`gnome-vfs-get-uri-scheme` (*uri* `mchars`) ⇒ (*ret* `mchars`) [Function]
Retrieve the scheme used in *uri*.

uri a stringified uri.

ret a newly allocated string containing the scheme, '`#f`' if *uri* doesn't contain a scheme.

Since 2.2

`gnome-vfs-uris-match` (*uri_1* `mchars`) (*uri_2* `mchars`) ⇒ (*ret* `bool`) [Function]
Compare two uris.

uri-1 stringified uri to compare with *uri-2*.

uri-2 stringified uri to compare with *uri-1*.

ret '`#t`' if they are the same, '`#f`' otherwise.

Since 2.2

5 GnomeVFSFileInfo

stores information about files, GnomeVFS equivalent of stat

5.1 Overview

5.2 Usage

`<gnome-vfs-file-info>` [Class]

Opaque pointer.

This class defines no direct slots.

6 Basic File Operations

Essential VFS operations. This includes creating, moving and deleting files, opening and closing file handles.

6.1 Overview

GnomeVFS file operations are, for the most part, patterned after their POSIX equivalents, with the systematic difference that they accept URIs rather than paths on the local filesystem. This makes them easy to learn as if you are already familiar with basic commands such as `open`, `seek`, `write`, etc you will feel right at home with GnomeVFS after learning a little about URIs.

6.2 Usage

`gnome-vfs-create` (*text_uri* `mchars`) [Function]
(*open_mode* `<gnome-vfs-open-mode>`) (*exclusive* `bool`)
(*perm* `unsigned-int`) ⇒ (*ret* `scm`)

Create *text-uri* according to mode *open-mode*. On return, *handle* will then contain a pointer to a handle for the open file.

handle pointer to a pointer to a `<gnome-vfs-handle>` object.

text-uri string representing the uri to create.

open-mode
mode to leave the file opened in after creation (or '`GNOME_VFS_OPEN_MODE_NONE`' to leave the file closed after creation).

exclusive whether the file should be created in "exclusive" mode. i.e. if this flag is nonzero, operation will fail if a file with the same name already exists.

perm bitmap representing the permissions for the newly created file (Unix style).

ret an integer representing the result of the operation.

`gnome-vfs-create-uri` (*uri* `<gnome-vfsuri>`) [Function]
(*open_mode* `<gnome-vfs-open-mode>`) (*exclusive* `bool`)
(*perm* `unsigned-int`) ⇒ (*ret* `scm`)

Create *uri* according to mode *open-mode*. On return, *handle* will then contain a pointer to a handle for the open file.

handle pointer to a pointer to a `<gnome-vfs-handle>` object.

uri uri for the file to create.

open-mode
open mode.

exclusive whether the file should be created in "exclusive" mode. i.e. if this flag is nonzero, operation will fail if a file with the same name already exists.

perm bitmap representing the permissions for the newly created file (Unix style).

ret an integer representing the result of the operation.

`gnome-vfs-open` (*text_uri* `mchars`) [Function]
(*open_mode* `<gnome-vfs-open-mode>`) ⇒ (*ret* `scm`)
Open *text-uri* according to mode *open-mode*. On return, *handle* will then contain a pointer to a handle for the open file.

handle pointer to a pointer to a `<gnome-vfs-handle>` object.

text-uri string representing the uri to open.

open-mode
open mode.

ret an integer representing the result of the operation.

`gnome-vfs-open-uri` (*uri* `<gnome-vfsuri>`) [Function]
(*open_mode* `<gnome-vfs-open-mode>`) ⇒ (*ret* `scm`)
Open *uri* according to mode *open-mode*. On return, *handle* will then contain a pointer to a handle for the open file.

handle pointer to a pointer to a `<gnome-vfs-handle>` object.

uri uri to open.

open-mode
open mode.

ret an integer representing the result of the operation.

`gnome-vfs-unlink` (*text_uri* `mchars`) ⇒ (*ret* `<gnome-vfs-result>`) [Function]
Unlink *text-uri* (i.e. delete the file).

text-uri uri of the file to be unlinked.

ret an integer representing the result of the operation.

`gnome-vfs-unlink-from-uri` (*uri* `<gnome-vfsuri>`) [Function]
⇒ (*ret* `<gnome-vfs-result>`)
Unlink *uri* (i.e. delete the file).

uri uri of the file to be unlinked.

ret an integer representing the result of the operation.

`gnome-vfs-move-uri` (*old_uri* `<gnome-vfsuri>`) [Function]
(*new_uri* `<gnome-vfsuri>`) (*force_replace* `bool`)
⇒ (*ret* `<gnome-vfs-result>`)
Move a file from uri *old-uri* to *new-uri*. This will only work if *old-uri* and *new-uri* are on the same file system. Otherwise, it is necessary to use the more general `gnome-vfs-xfer-uri` function.

old-uri source uri.

new-uri destination uri.

force-replace
if '`#t`', move *old-uri* to *new-uri* even if there is already a file at *new-uri*. If there is a file, it will be discarded.

ret an integer representing the result of the operation.

`gnome-vfs-move` (*old_text_uri* `mchars`) (*new_text_uri* `mchars`) [Function]
(*force_replace* `bool`) ⇒ (*ret* `<gnome-vfs-result>`)
Move a file from *old-text-uri* to *new-text-uri*. This will only work if *old-text-uri* and *new-text-uri* are on the same file system. Otherwise, it is necessary to use the more general `gnome-vfs-xfer-uri` function.

old-text-uri
string representing the source file location.

new-text-uri
string representing the destination file location.

force-replace
if '`#t`', perform the operation even if it unlinks an existing file at *new-text-uri*.

ret an integer representing the result of the operation.

`gnome-vfs-check-same-fs-uris` (*source_uri* `<gnome-vfsuri>`) [Function]
(*target_uri* `<gnome-vfsuri>`) ⇒ (*ret* `<gnome-vfs-result>`)
(*same_fs_return* `bool`)
Check if *source-uri* and *target-uri* are on the same file system.

source-uri a uri.

target-uri another uri.

same-fs-return
pointer to a boolean variable which will be set to '`#t`' on return if *source-uri* and *target-uri* are on the same file system.

ret an integer representing the result of the operation.

`gnome-vfs-check-same-fs` (*source* `mchars`) (*target* `mchars`) [Function]
⇒ (*ret* `<gnome-vfs-result>`) (*same_fs_return* `bool`)
Check if *source* and *target* are on the same file system.

source path to a file.

target path to another file.

same-fs-return
pointer to a boolean variable which will be set to '`#t`' on return if *source* and *target* are on the same file system.

ret an integer representing the result of the operation.

`gnome-vfs-uri-exists` (*self* `<gnome-vfsuri>`) ⇒ (*ret* `bool`) [Function]

Check if the uri points to an existing entity.

uri a uri.

ret '`#t`' if uri exists.

`gnome-vfs-create-symbolic-link` (*uri* `<gnome-vfsuri>`) [Function]
(*target_reference* `mchars`) ⇒ (*ret* `<gnome-vfs-result>`)

Creates a symbolic link, or eventually, a uri link (as necessary) at *uri* pointing to *target-reference*.

uri uri to create a link at.

target-reference
uri "reference" to point the link to (uri or relative path).

ret an integer representing the result of the operation.

7 Truncating Files

Force files to a particular length

7.1 Overview

Truncation of files is used to force them to a particular length. If a file longer than specified, the trailing bytes are discarded, if it is shorter than specified it is padded with zeros.

7.2 Usage

`gnome-vfs-truncate` (*text_uri* `mchars`) (*length* `unsigned-int64`) ⇒ (*ret* `<gnome-vfs-result>`) [Function]

Truncate the file at *text-uri* to *length* bytes.

text-uri	string representing the file to be truncated.
length	length of the new file at *text-uri*.
ret	an integer representing the result of the operation.

`gnome-vfs-truncate-uri` (*uri* `<gnome-vfsuri>`) (*length* `unsigned-int64`) ⇒ (*ret* `<gnome-vfs-result>`) [Function]

Truncate the file at *uri* to be only *length* bytes. Data past *length* bytes will be discarded.

uri	uri of the file to be truncated.
length	length of the new file at *uri*.
ret	an integer representing the result of the operation.

8 Getting and Setting File Information

Convenient high-level abstraction for obtaining and setting file information, including ACLs.

8.1 Overview

Applications can use the `gnome-vfs-get-file-info` family of operations to retrieve file information, as this operation can be quite costly in terms of time (specilly when sniffing the MIME type) applications can specify which information need at any time, reducing the performance impact.

All of these operations use a `<gnome-vfs-file-info>` data structure that holds the file information, there are several methods that can be used to manipulate this information. See `<gnome-vfs-file-info>` for more information.

8.2 Usage

`gnome-vfs-get-file-info` (*text_uri* `mchars`) [Function]
(*info* `<gnome-vfs-file-info>`)
(*options* `<gnome-vfs-file-info-options>`)
⇒ (*ret* `<gnome-vfs-result>`)

Retrieve information about *text-uri*. The information will be stored in *info*.

text-uri uri of the file for which information will be retrieved.

info pointer to a `<gnome-vfs-file-info>` object that will hold the information for the file on return.

options options for retrieving file information.

ret an integer representing the result of the operation.

`gnome-vfs-set-file-info` (*text_uri* `mchars`) [Function]
(*info* `<gnome-vfs-file-info>`)
(*mask* `<gnome-vfs-set-file-info-mask>`)
⇒ (*ret* `<gnome-vfs-result>`)

Set file information for *uri*; only the information for which the corresponding bit in *mask* is set is actually modified.

> *info*'s '`valid_fields`' is not required to contain the `<gnome-vfs-file-info-fields>` corresponding to the specified `<gnome-vfs-set-file-info-mask>` fields of *mask*. It is assumed that the *info* fields referenced by *mask* are valid.

text-uri string representing the file location.

info information that must be set for the file.

mask bit mask representing which fields of *info* need to be set.

ret an integer representing the result of the operation.

9 Basic Directory Operations

Creating and removing directories.

9.1 Overview

9.2 Usage

`gnome-vfs-make-directory` (*text_uri* `mchars`) (*perm* `unsigned-int`) [Function]
⇒ (*ret* `<gnome-vfs-result>`)

Create *text-uri* as a directory.

text-uri uri of the directory to be created.

perm Unix-style permissions for the newly created directory

ret an integer representing the result of the operation.

`gnome-vfs-make-directory-for-uri` (*uri* `<gnome-vfsuri>`) [Function]
(*perm* `unsigned-int`) ⇒ (*ret* `<gnome-vfs-result>`)

Create a directory at *uri*. Only succeeds if a file or directory does not already exist at *uri*.

uri uri of the directory to be created.

perm Unix-style permissions for the newly created directory.

ret an integer representing the result of the operation.

`gnome-vfs-remove-directory` (*text_uri* `mchars`) [Function]
⇒ (*ret* `<gnome-vfs-result>`)

Remove *text-uri*. *text-uri* must be an empty directory.

text-uri path of the directory to be removed.

ret an integer representing the result of the operation.

`gnome-vfs-remove-directory-from-uri` (*uri* `<gnome-vfsuri>`) [Function]
⇒ (*ret* `<gnome-vfs-result>`)

Remove *uri*. *uri* must be an empty directory.

uri uri of the directory to be removed.

ret an integer representing the result of the operation.

10 GnomeVFSVolume

Abstraction for a mounted file system or a network location.

10.1 Overview

10.2 Usage

`<gnome-vfs-volume>` [Class]

Derives from `<gobject>`.

This class defines no direct slots.

`gnome-vfs-volume-compare` (*self* `<gnome-vfs-volume>`) [Function]
(*b* `<gnome-vfs-volume>`) ⇒ (*ret* `int`)

`compare` [Method]

Compares two `<gnome-vfs-volume>` objects *a* and *b*. Two `<gnome-vfs-volume>` objects referring to different volumes are guaranteed to not return 0 when comparing them, if they refer to the same volume 0 is returned.

The resulting `<gint>` should be used to determine the order in which *a* and *b* are displayed in graphical user interfces.

The comparison algorithm first of all peeks the device type of *a* and *b*, they will be sorted in the following order:

-
-
-
-
-
-

Magnetic and opto-magnetic volumes (ZIP, floppy)

Optical volumes (CD, DVD)

External volumes (USB sticks, music players)

Mounted hard disks

Network mounts

Other volumes

Afterwards, the display name of *a* and *b* is compared using a locale-sensitive sorting algorithm, which involves `g-utf8-collate-key`.

If two volumes have the same display name, their unique ID is compared which can be queried using `gnome-vfs-volume-get-id`.

a a `<gnome-vfs-volume>`.

b a `<gnome-vfs-volume>`.

ret 0 if the volumes refer to the same *gnome-vfs-volume*, a negative value if *a* should be displayed before *b*, or a positive value if *a* should be displayed after *b*.

Since 2.6

`gnome-vfs-volume-get-activation-uri` (*self* `<gnome-vfs-volume>`) [Function]
⇒ (*ret* `mchars`)
`get-activation-uri` [Method]

Returns the activation URI of a `<gnome-vfs-volume>`.

The returned URI usually refers to a valid location. You can check the validity of the location by calling `gnome-vfs-uri-new` with the URI, and checking whether the return value is not '`#f`'.

volume a `<gnome-vfs-volume>`.

ret a newly allocated string for activation uri of *volume*.

Since 2.6

`gnome-vfs-volume-get-device-path` (*self* `<gnome-vfs-volume>`) [Function]
⇒ (*ret* `mchars`)
`get-device-path` [Method]

Returns the device path of a `<gnome-vfs-volume>`.

For HAL volumes, this returns the value of the volume's `"block.device"` key. For UNIX mounts, it returns the '`mntent`''s '`mnt_fsname`' entry.

Otherwise, it returns '`#f`'.

volume a `<gnome-vfs-volume>`.

ret a newly allocated string for device path of *volume*.

Since 2.6

`gnome-vfs-volume-get-device-type` (*self* `<gnome-vfs-volume>`) [Function]
⇒ (*ret* `<gnome-vfs-device-type>`)
`get-device-type` [Method]

Returns the `<gnome-vfs-device-type>` of the *volume*.

volume a `<gnome-vfs-volume>`.

ret the device type for *volume*.

Since 2.6

`gnome-vfs-volume-get-display-name` (*self* `<gnome-vfs-volume>`) [Function]
⇒ (*ret* `mchars`)
`get-display-name` [Method]

Returns the display name of the *volume*.

volume a `<gnome-vfs-volume>`.

ret a newly allocated string for display name of *volume*.

Since 2.6

`gnome-vfs-volume-get-drive` (*self* `<gnome-vfs-volume>`) [Function]
⇒ (*ret* `<gnome-vfs-drive>`)
`get-drive` [Method]

volume a `<gnome-vfs-volume>`.

ret the drive for the *volume*.

Since 2.6

`gnome-vfs-volume-get-hal-udi` (*self* `<gnome-vfs-volume>`) [Function]
⇒ (*ret* `mchars`)
`get-hal-udi` [Method]

Returns the HAL UDI of a `<gnome-vfs-volume>`.

For HAL volumes, this matches the value of the "info.udi" key, for other volumes it is '`#f`'.

volume a `<gnome-vfs-volume>`.

ret a newly allocated string for unique device id of *volume*, or '`#f`'.

Since 2.6

`gnome-vfs-volume-get-icon` (*self* `<gnome-vfs-volume>`) [Function]
⇒ (*ret* `mchars`)
`get-icon` [Method]

volume a `<gnome-vfs-volume>`.

ret a newly allocated string for the icon filename of *volume*.

Since 2.6

`gnome-vfs-volume-get-id` (*self* `<gnome-vfs-volume>`) [Function]
⇒ (*ret* `unsigned-long`)
`get-id` [Method]

Returns the ID of the *volume*,

Two `<gnome-vfs-volumes>` are guaranteed to refer to the same volume if they have the same ID.

volume a `<gnome-vfs-volume>`.

ret the id for the *volume*.

Since 2.6

`gnome-vfs-volume-get-volume-type` (*self* `<gnome-vfs-volume>`) [Function]
⇒ (*ret* `<gnome-vfs-volume-type>`)
`get-volume-type` [Method]

Returns the `<gnome-vfs-volume-type>` of the *volume*.

volume a `<gnome-vfs-volume>`.

ret the volume type for *volume*.

Since 2.6

`gnome-vfs-volume-handles-trash` (*self* `<gnome-vfs-volume>`) [Function]
⇒ (*ret* `bool`)

`handles-trash` [Method]

Returns whether the file system on a *volume* supports trashing of files.

If the *volume* has an AutoFS file system (i.e. `gnome-vfs-volume-get-device-type` returns `<gnome-vfs-device-type-autofs>`), or if the *volume* is mounted read-only (`gnome-vfs-volume-is-read-only` returns '`#t`'), it is assumed to not support trashing of files.

Otherwise, if the *volume* provides file system information, it is determined whether the file system supports trashing of files. See `gnome-vfs-volume-get-filesystem-type` for details which volumes provide file system information, and which file systems currently support a trash.

volume a `<gnome-vfs-volume>`.

ret '`#t`' if *volume* handles trash, '`#f`' otherwise.

Since 2.6

`gnome-vfs-volume-is-mounted` (*self* `<gnome-vfs-volume>`) [Function]
⇒ (*ret* `bool`)

`is-mounted` [Method]

Returns whether the file system on a *volume* is currently mounted.

For HAL volumes, this reflects the value of the "volume.is_mounted" key, for traditional UNIX mounts and connected servers, '`#t`' is returned, because their existence implies that they are mounted.

volume a `<gnome-vfs-volume>`.

ret '`#t`' if the *volume* is mounted, '`#f`' otherwise.

Since 2.6

`gnome-vfs-volume-is-read-only` (*self* `<gnome-vfs-volume>`) [Function]
⇒ (*ret* `bool`)

`is-read-only` [Method]

Returns whether the file system on a *volume* is read-only.

For HAL volumes, the "volume.is_mounted_read_only" key is authoritative, for traditional UNIX mounts it returns '`#t`' if the mount was done with the "ro" option. For servers, '`#f`' is returned.

volume a `<gnome-vfs-volume>`.

ret '`#t`' if the *volume* is read-only to the user, '`#f`' otherwise.

Since 2.6

`gnome-vfs-volume-is-user-visible` (*self* `<gnome-vfs-volume>`) [Function]
⇒ (*ret* `bool`)

`is-user-visible` [Method]

Returns whether the *volume* is visible to the user. This should be used by applications to determine whether it is included in user interfaces listing available volumes.

volume a `<gnome-vfs-volume>`.

ret '`#t`' if *volume* is visible to the user, '`#f`' otherwise.

Since 2.6

`gnome-vfs-connect-to-server` (*uri* `mchars`) (*display_name* `mchars`) (*icon* `mchars`) [Function]

This function adds a server connection to the specified *uri*, which is displayed in user interfaces with the specified *display-name* and *icon*.

If this function is invoked successfully, the created server shows up in the list of mounted volumes of the `<gnome-vfs-volume-monitor>`, which can be queried using `gnome-vfs-volume-monitor-get-mounted-volumes`.

This function does not have a return value. Hence, you can't easily detect whether the specified server was successfully created. The actual creation and consumption of the new server through the `<gnome-vfs-volume-monitor>` is done asynchronously.

uri, *display-name*, and *icon* can be freely chosen, but should be meaningful:

uri should refer to a valid location. You can check the validity of the location by calling `gnome-vfs-uri-new` with *uri*, and checking whether the return value is not '`#f`'.

The *display-name* should be queried from the user, and an empty string should not be considered valid.

icon typically references an icon from the icon theme. Some implementations currently use '`gnome-fs-smb`', '`gnome-fs-ssh`', '`gnome-fs-ftp`' and '`gnome-fs-share`', depending on the type of the server referenced by *uri*. The icon naming conventions might change in the future, though. Obeying the freedesktop.org Icon Naming Specification is suggested.

uri The string representation of the server to connect to.

display-name
The display name that is used to identify the server connection.

icon The icon that is used to identify the server connection.

Since 2.6

11 GnomeVFSDrive

Container for GnomeVFSVolume (floppy drive, CD reader, ...)

11.1 Overview

11.2 Usage

`<gnome-vfs-drive>` [Class]

Derives from `<gobject>`.

This class defines no direct slots.

`volume-mounted` (*arg0* `<gnome-vfs-volume>`) [Signal on `<gnome-vfs-drive>`]

This signal is emitted after the `<gnome-vfs-volume>`*volume* has been mounted.

When the *volume* is mounted, it is added to the *drive*'s list of mounted volumes, which can be queried using `gnome-vfs-drive-get-mounted-volumes`.

It is also added to the list of the `<gnome-vfs-volume-monitor>`'s list of mounted volumes, which can be queried using `gnome-vfs-volume-monitor-get-mounted-volumes`.

`volume-pre-unmount` [Signal on `<gnome-vfs-drive>`]
(*arg0* `<gnome-vfs-volume>`)

This signal is emitted when the `<gnome-vfs-volume>`*volume*, which has been present in the `<gnome-vfs-drive>`*drive*, is about to be unmounted.

When the *volume* is unmounted, it is removed from the *drive*'s list of mounted volumes, which can be queried using `gnome-vfs-drive-get-mounted-volumes`.

It is also removed from the `<gnome-vfs-volume-monitor>`'s list of mounted volumes, which can be queried using `gnome-vfs-volume-monitor-get-mounted-volumes`.

When a client application receives this signal, it must free all resources associated with the *volume*, for instance cancel all pending file operations on the *volume*, and cancel all pending file monitors using `gnome-vfs-monitor-cancel`.

`volume-unmounted` (*arg0* `<gnome-vfs-volume>`) [Signal on `<gnome-vfs-drive>`]

This signal is emitted after the `<gnome-vfs-volume>`*volume*, which had been present in the `<gnome-vfs-drive>`*drive*, has been unmounted.

When the *volume* is unmounted, it is removed from the *drive*'s list of mounted volumes, which can be queried using `gnome-vfs-drive-get-mounted-volumes`.

It is also removed from the `<gnome-vfs-volume-monitor>`'s list of mounted volumes, which can be queried using `gnome-vfs-volume-monitor-get-mounted-volumes`.

`gnome-vfs-drive-compare` (*self* `<gnome-vfs-drive>`) [Function]
(*b* `<gnome-vfs-drive>`) ⇒ (*ret* `int`)

`compare` [Method]

Compares two `<gnome-vfs-drive>` objects *a* and *b*. Two `<gnome-vfs-drive>` objects referring to different drives are guaranteed to not return 0 when comparing them, if they refer to the same drive 0 is returned.

The resulting `<gint>` should be used to determine the order in which *a* and *b* are displayed in graphical user interfces.

The comparison algorithm first of all peeks the device type of *a* and *b*, they will be sorted in the following order:

-
-
-
-
-

Magnetic and opto-magnetic drives (ZIP, floppy)

Optical drives (CD, DVD)

External drives (USB sticks, music players)

Mounted hard disks<

Other drives<

Afterwards, the display name of *a* and *b* is compared using a locale-sensitive sorting algorithm, which involves `g-utf8-collate-key`.

If two drives have the same display name, their unique ID is compared which can be queried using `gnome-vfs-drive-get-id`.

a a `<gnome-vfs-drive>`.

b a `<gnome-vfs-drive>`.

ret 0 if the drives refer to the same *gnome-vfs-drive*, a negative value if *a* should be displayed before *b*, or a positive value if *a* should be displayed after *b*.

Since 2.6

`gnome-vfs-drive-get-activation-uri` (*self* `<gnome-vfs-drive>`) [Function]
⇒ (*ret* `mchars`)
`get-activation-uri` [Method]

Returns the activation URI of a `<gnome-vfs-drive>`.

The returned URI usually refers to a valid location. You can check the validity of the location by calling `gnome-vfs-uri-new` with the URI, and checking whether the return value is not '`#f`'.

drive a `<gnome-vfs-drive>`.

ret a newly allocated string for the activation uri of the `<drive>`.

Since 2.6

`gnome-vfs-drive-get-device-path` (*self* `<gnome-vfs-drive>`) [Function]
⇒ (*ret* `mchars`)
`get-device-path` [Method]

Returns the device path of a `<gnome-vfs-drive>`.

For HAL drives, this returns the value of the drives's "block.device" key. For UNIX mounts, it returns the '`mntent`''s '`mnt_fsname`' entry.

Otherwise, it returns '`#f`'.

drive a `<gnome-vfs-drive>`.

ret a newly allocated string for the device path of the `<drive>`.

Since 2.6

`gnome-vfs-drive-get-device-type` (*self* `<gnome-vfs-drive>`) [Function]
⇒ (*ret* `<gnome-vfs-device-type>`)

`get-device-type` [Method]

drive a `<gnome-vfs-drive>`.

ret device type, a `<gnome-vfs-device-type>` value.

Since 2.6

`gnome-vfs-drive-get-display-name` (*self* `<gnome-vfs-drive>`) [Function]
⇒ (*ret* `mchars`)

`get-display-name` [Method]

drive a `<gnome-vfs-drive>`.

ret a newly allocated string for the display name of the *drive*.

Since 2.6

`gnome-vfs-drive-get-hal-udi` (*self* `<gnome-vfs-drive>`) [Function]
⇒ (*ret* `mchars`)

`get-hal-udi` [Method]

Returns the HAL UDI of a `<gnome-vfs-drive>`.

For HAL drives, this matches the value of the "info.udi" key, for other drives it is '`#f`'.

drive a `<gnome-vfs-drive>`.

ret a newly allocated string for the unique device id of the *drive*, or '`#f`'.

Since 2.6

`gnome-vfs-drive-get-icon` (*self* `<gnome-vfs-drive>`) [Function]
⇒ (*ret* `mchars`)

`get-icon` [Method]

drive a `<gnome-vfs-drive>`.

ret a newly allocated string for the icon filename of the *drive*.

Since 2.6

`gnome-vfs-drive-get-id` (*self* `<gnome-vfs-drive>`) [Function]
⇒ (*ret* `unsigned-long`)

`get-id` [Method]

drive a `<gnome-vfs-drive>`.

ret drive id, a `<gulong>` value.

Since 2.6

`gnome-vfs-drive-get-mounted-volumes` (*self* `<gnome-vfs-drive>`) [Function]
⇒ (*ret* `glist-of`)
`get-mounted-volumes` [Method]

drive a `<gnome-vfs-drive>`.

ret list of mounted volumes for the *drive*.

Since 2.8

`gnome-vfs-drive-is-connected` (*self* `<gnome-vfs-drive>`) [Function]
⇒ (*ret* `bool`)
`is-connected` [Method]

drive a `<gnome-vfs-drive>`.

ret '`#t`' if the *drive* is connected, '`#f`' otherwise.

Since 2.6

`gnome-vfs-drive-is-mounted` (*self* `<gnome-vfs-drive>`) [Function]
⇒ (*ret* `bool`)
`is-mounted` [Method]

drive a `<gnome-vfs-drive>`.

ret '`#t`' if the *drive* is mounted, '`#f`' otherwise.

Since 2.6

`gnome-vfs-drive-is-user-visible` (*self* `<gnome-vfs-drive>`) [Function]
⇒ (*ret* `bool`)
`is-user-visible` [Method]

Returns whether the *drive* is visible to the user. This should be used by applications to determine whether it is included in user interfaces listing available drives.

drive a `<gnome-vfs-drive>`.

ret '`#t`' if the *drive* is visible to the user, '`#f`' otherwise.

Since 2.6

12 GnomeVFSVolumeMonitor

Monitors volume mounts and unmounts

12.1 Overview

12.2 Usage

`<gnome-vfs-volume-monitor>` [Class]

Derives from `<gobject>`.

This class defines no direct slots.

`volume-mounted` [Signal on `<gnome-vfs-volume-monitor>`]
(*arg0* `<gnome-vfs-volume>`)

This signal is emitted after the `<gnome-vfs-volume>`*volume* has been mounted.

When the *volume* is mounted, it is present in the *volume-monitor*'s list of mounted volumes, which can be queried using `gnome-vfs-volume-monitor-get-mounted-volumes`.

If the *volume* has an associated `<gnome-vfs-drive>`, it also appears in the drive's list of mounted volumes, which can be queried using `gnome-vfs-drive-get-mounted-volumes`.

`volume-pre-unmount` [Signal on `<gnome-vfs-volume-monitor>`]
(*arg0* `<gnome-vfs-volume>`)

This signal is emitted when the `<gnome-vfs-volume>`*volume* is about to be unmounted.

When the *volume* is unmounted, it is removed from the *volume-monitor*'s list of mounted volumes, which can be queried using `gnome-vfs-volume-monitor-get-mounted-volumes`.

If the *volume* has an associated `<gnome-vfs-drive>`, it is also removed from in the drive's list of mounted volumes, which can be queried using `gnome-vfs-drive-get-mounted-volumes`.

When a client application receives this signal, it must free all resources associated with the *volume*, for instance cancel all pending file operations on the *volume*, and cancel all pending file monitors using `gnome-vfs-monitor-cancel`.

`volume-unmounted` [Signal on `<gnome-vfs-volume-monitor>`]
(*arg0* `<gnome-vfs-volume>`)

This signal is emitted after the `<gnome-vfs-volume>`*volume* has been unmounted.

When the *volume* is unmounted, it is removed from the *volume-monitor*'s list of mounted volumes, which can be queried using `gnome-vfs-volume-monitor-get-mounted-volumes`.

If the *volume* has an associated `<gnome-vfs-drive>`, it is also removed from in the drive's list of mounted volumes, which can be queried using `gnome-vfs-drive-get-mounted-volumes`.

`drive-connected` [Signal on `<gnome-vfs-volume-monitor>`]

(*arg0* `<gnome-vfs-drive>`)

This signal is emitted when the `<gnome-vfs-drive>`*drive* has been connected.

When the *drive* is connected, it is present in the *volume-monitor*'s list of connected drives, which can be queried using `gnome-vfs-volume-monitor-get-connected-drives`.

`drive-disconnected` [Signal on `<gnome-vfs-volume-monitor>`]

(*arg0* `<gnome-vfs-drive>`)

This signal is emitted after the `<gnome-vfs-drive>`*drive* has been disconnected.

When the *drive* is disconnected, it is removed from the *volume-monitor*'s list of connected drives, which can be queried using `gnome-vfs-volume-monitor-get-connected-drives`.

`gnome-vfs-get-volume-monitor` [Function]

⇒ (*ret* `<gnome-vfs-volume-monitor>`)

Returns a pointer to the `<gnome-vfs-volume-monitor>` singleton. `<gnome-vfs-volume-monitor>` is a singleton, this means it is guaranteed to exist and be valid until `gnome-vfs-shutdown` is called. Consequently, it doesn't need to be refcounted since gnome-vfs will hold a reference to it until it is shut down.

ret a pointer to the `<gnome-vfs-volume-monitor>` singleton.

Since 2.6

13 MIME typing

functions to get a mime-type for a file using its name or its content

13.1 Overview

13.2 Usage

`gnome-vfs-mime-type-from-name` (*filename* `mchars`) ⇒ (*ret* `mchars`) [Function]

'`gnome_vfs_mime_type_from_name`' is deprecated and should not be used in newly-written code. This function is deprecated, use `gnome-vfs-get-mime-type-for-name` instead.

Determine the mime type for *filename*.

Returns:

filename a filename (the file does not necessarily exist).

ret the mime-type for this filename. Will return '`GNOME_VFS_MIME_TYPE_UNKNOWN`' if mime-type could not be found.

`gnome-vfs-get-mime-type-common` (*uri* `<gnome-vfsuri>`) [Function]
⇒ (*ret* `mchars`)

Tries to guess the mime type of the file represented by *uri*. Favors using the file data to the *uri* extension. Handles *uri* of a non-existent file by falling back on returning a type based on the extension. If cant find the mime-type based on the extension also then returns 'application/octet-stream'.

FIXME: This function will not necessarily return the same mime type as doing a get file info on the text uri.

uri a real file or a non-existent uri.

ret the mime-type for *uri*.

`gnome-vfs-get-mime-type-from-uri` (*uri* `<gnome-vfsuri>`) [Function]
⇒ (*ret* `mchars`)

Tries to guess the mime type of the file *uri* by checking the file name extension. Works on non-existent files.

uri a file uri.

ret the mime-type for file at *uri*.

`gnome-vfs-mime-type-is-supertype` (*mime_type* `mchars`) [Function]
⇒ (*ret* `bool`)

mime-type
a const char * identifying a mime type.

ret Whether *mime-type* is of the form "foo'/'*".

`gnome-vfs-mime-info-cache-reload` (*dir* `mchars`) [Function]
Reload the mime information for the *dir*.

dir directory path which needs reloading.

`gnome-vfs-mime-reload` [Function]
Reload the MIME database.

`gnome-vfs-mime-shutdown` [Function]
'`gnome_vfs_mime_shutdown`' is deprecated and should not be used in newly-written code. This function doesn't have to be called as the operating system automatically cleans up resources when exiting.

Unload the MIME database from memory.

14 gnome-vfs-mime-utils

14.1 Overview

14.2 Usage

`gnome-vfs-mime-type-is-equal` (*a* `mchars`) (*b* `mchars`) ⇒ (*ret* `bool`) [Function]

Compares two mime types to determine if they are equivalent. They are equivalent if and only if they refer to the same mime type.

a a const char * containing a mime type, e.g. `"image/png"`.

b a const char * containing a mime type, e.g. `"image/png"`.

ret '`#t`', if a and b are equivalent mime types.

`gnome-vfs-get-mime-type-for-name` (*filename* `mchars`) [Function]
⇒ (*ret* `mchars`)

Determine the mime type for *filename*. The file *filename* may not exist, this function does not access the actual file. If the mime-type cannot be determined, '`GNOME_VFS_MIME_TYPE_UNKNOWN`' is returned.

filename a filename.

ret the mime-type for this filename or '`GNOME_VFS_MIME_TYPE_UNKNOWN`' if mime-type could not be determined.

Since 2.14

`gnome-vfs-get-mime-type` (*text_uri* `mchars`) ⇒ (*ret* `mchars`) [Function]

Determine the mime type of *text-uri*. The mime type is determined in the same way as by `gnome-vfs-get-file-info`. This is meant as a convenient function for times when you only want the mime type.

text-uri path of the file for which to get the mime type.

ret The mime type, or '`#f`' if there is an error reading the file.

`gnome-vfs-get-slow-mime-type` (*text_uri* `mchars`) ⇒ (*ret* `mchars`) [Function]

Determine the mime type of *text-uri*. The mime type is determined in the same way as by `gnome-vfs-get-file-info`. This is meant as a convenience function for times when you only want the mime type.

text-uri URI of the file for which to get the mime type

ret The mime type, or NULL if there is an error reading the file.

Since 2.14

15 MIME Database

functions for getting information about applications and components associated with MIME types.

15.1 Overview

This API can be used to query the applications and components associated with particular MIME types, and to collect extra information about them, and also provides a convenience function for launching them. Applications can register themselves for particular MIME types by adhering to the Freedesktop.Org Desktop Entry Specification.

15.2 Usage

`<gnome-vfs-mime-application>` [Class]

Opaque pointer.

This class defines no direct slots.

`gnome-vfs-mime-get-all-applications` (*mime_type* `mchars`) [Function]
⇒ (*ret* `glist-of`)

Return an alphabetically sorted list of `<gnome-vfs-mime-application>` data structures representing all applications in the MIME database registered to handle files of MIME type *mime-type* (and supertypes).

mime-type
: a const char * containing a mime type, e.g. "image/png".

ret
: a `<g-list>` * where the elements are `<gnome-vfs-mime-application>` * representing applications that handle MIME type *mime-type*.

`gnome-vfs-mime-get-all-components` (*mime_type* `mchars`) [Function]
⇒ (*ret* `glist-of`)

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

mime-type
ret

`gnome-vfs-mime-get-description` (*mime_type* `mchars`) [Function]
⇒ (*ret* `mchars`)

Query the MIME database for a description of the *mime-type*.

mime-type
: the mime type.

ret
: description of MIME type *mime-type*.

`gnome-vfs-mime-can-be-executable` (*mime_type* `mchars`) [Function]
⇒ (*ret* `bool`)

Check whether files of *mime-type* might conceivably be executable. Default for known types if '`#f`'. Default for unknown types is '`#t`'.

mime-type
: a const char * containing a mime type.

ret
: '`#t`' if files of *mime-type* can be executable, '`#f`' otherwise.

`gnome-vfs-mime-application-launch` [Function]
(*self* `<gnome-vfs-mime-application>`) (*uris* `glist-of`)
⇒ (*ret* `<gnome-vfs-result>`)

Launches the given mime application with the given parameters. Command line parameters will be expanded as required by the application. The application will also be launched in a terminal if that is required. If the application only supports one argument per instance then multiple instances of the application will be launched.

app
: the `<gnome-vfs-mime-application>` to launch.

uris
: parameters for the `<gnome-vfs-mime-application>`.

ret
: '`GNOME_VFS_OK`' if the application was launched. '`GNOME_VFS_ERROR_NOT_SUPPORTED`' if the uri protocol is not supported by the application. '`GNOME_VFS_ERROR_PARSE`' if the application command can not be parsed. '`GNOME_VFS_ERROR_LAUNCH`' if the application command can not be launched. '`GNOME_VFS_ERROR_INTERNAL`' for other internal and GConf errors.

Since 2.4

`gnome-vfs-mime-application-get-name` [Function]
(*self* `<gnome-vfs-mime-application>`) ⇒ (*ret* `mchars`)

Returns the name of the application *app*

app
: a `<gnome-vfs-mime-application>`.

ret
: the name of the application.

Since 2.10

`gnome-vfs-mime-application-get-icon` [Function]
(*self* `<gnome-vfs-mime-application>`) ⇒ (*ret* `mchars`)

Returns an icon representing the specified application.

app
: a `<gnome-vfs-mime-application>`.

ret
: the filename of the icon usually without path information, e.g. "gedit-icon.png", and sometimes does not have an extension, e.g. "gnome-pdf" if the icon is supposed to be image type agnostic between icon themes. Icons are generic, and not theme specific.

Since 2.10

`gnome-vfs-mime-application-get-exec` [Function]
(*self* `<gnome-vfs-mime-application>`) ⇒ (*ret* `mchars`)

Returns the program to execute, possibly with arguments and parameter variables, as specified by the Desktop Entry Specification.

app
: a `<gnome-vfs-mime-application>`.

ret the command line to execute.

Since 2.10

`gnome-vfs-mime-application-equal` [Function]

(*self* `<gnome-vfs-mime-application>`)
(*app_b* `<gnome-vfs-mime-application>`) ⇒ (*ret* `bool`)

Compare *app-a* and *app-b*.

app-a a `<gnome-vfs-mime-application>`.

app-b a `<gnome-vfs-mime-application>`.

ret '`#t`' if *app-a* and *app-b* are equal, '`#f`' otherwise.

Since 2.10

16 Undocumented

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

16.1 (gnome gw gnome-vfs)

gnome-vfs-ace-add-perm [Variable]
gnome-vfs-ace-check-perm [Variable]
gnome-vfs-ace-copy-perms [Variable]
gnome-vfs-ace-del-perm [Variable]
gnome-vfs-ace-equal [Variable]
gnome-vfs-ace-get-id [Variable]
gnome-vfs-ace-get-inherit [Variable]
gnome-vfs-ace-get-kind [Variable]
gnome-vfs-ace-get-negative [Variable]
gnome-vfs-ace-set-id [Variable]
gnome-vfs-ace-set-inherit [Variable]
gnome-vfs-ace-set-kind [Variable]
gnome-vfs-ace-set-negative [Variable]
gnome-vfs-acl-clear [Variable]
gnome-vfs-acl-get-ace-list [Function]
gnome-vfs-acl-kind-to-string [Variable]
gnome-vfs-acl-new [Variable]
gnome-vfs-acl-perm-to-string [Variable]
gnome-vfs-acl-set [Variable]
gnome-vfs-acl-unset [Variable]
gnome-vfs-context-check-cancellation-current [Variable]
gnome-vfs-drive-needs-eject [Variable]
gnome-vfs-escape-host-and-path-string [Variable]
gnome-vfs-format-file-size-for-display [Variable]
gnome-vfs-get-default-browse-domains [Function]
gnome-vfs-get-mime-type-from-file-data [Variable]
gnome-vfs-get-supertype-from-mime-type [Variable]
gnome-vfs-init [Variable]
gnome-vfs-initialized [Variable]

Type Index

Function Index

C

D

G

H

I

V