This manual documents GNU G-Golf version 0.1.0.

Copyright (C) 2016 - 2022 Free Software Foundation, Inc.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled “GNU Free Documentation License.”
Table of Contents

Preface ................................................................. 1
  Contributors to this Manual ........................................ 1
  Join the GNU Project ............................................. 1
  The G-Golf License ............................................... 1

I. Introduction ......................................................... 1
  About G-Golf ....................................................... 1
  Obtaining and installing G-Golf ............................... 2
  Contact Information .............................................. 4
  Reporting Bugs ................................................... 5

II. Using G-Golf ......................................................... 5
  Before you start .................................................. 5
    Naming Conventions ............................................. 5
    GOOPS Notes and Conventions ................................ 7
    Configuring Guile for G-Golf ................................ 9
    Customizing G-Golf ............................................ 10
  Getting Started with G-Golf .................................... 11
    Hello World! ................................................... 11
    Selective Import ............................................... 13
    Scripting ....................................................... 14
    Building Applications ......................................... 18
    G-Golf on Mobile Devices ..................................... 18
  Working with GNOME ............................................. 18
    Import ........................................................ 19
    Events ........................................................ 20
    GObject ....................................................... 26
  G-Golf Valley .................................................... 31
    Cache Park ..................................................... 31
    Customization Square ......................................... 32
    VFunc Alley .................................................... 38

III. G-Golf Core Reference .......................................... 41
  Overview ........................................................ 41
    Structure and Naming Conventions ............................ 41
  Glib .............................................................. 43
    Memory Allocation ............................................. 43
    The Main Event Loop .......................................... 43
    IO Channels ................................................... 47
    Miscellaneous Utility Functions ............................. 48
    UNIX-specific utilities and integration ....................... 50
  Doubly-Linked Lists ............................................ 50
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Singly-Linked Lists</td>
<td>51</td>
</tr>
<tr>
<td>Quarks</td>
<td>53</td>
</tr>
<tr>
<td>GObject</td>
<td>53</td>
</tr>
<tr>
<td>Type Information</td>
<td>54</td>
</tr>
<tr>
<td>GObject</td>
<td>58</td>
</tr>
<tr>
<td>Enumeration and Flag Types</td>
<td>60</td>
</tr>
<tr>
<td>Boxed Types</td>
<td>60</td>
</tr>
<tr>
<td>Generic Values</td>
<td>61</td>
</tr>
<tr>
<td>Parameters and Values</td>
<td>62</td>
</tr>
<tr>
<td>GParamSpec</td>
<td>66</td>
</tr>
<tr>
<td>Closures</td>
<td>69</td>
</tr>
<tr>
<td>Signals</td>
<td>71</td>
</tr>
<tr>
<td>GObject Introspection</td>
<td>74</td>
</tr>
<tr>
<td>Repository</td>
<td>74</td>
</tr>
<tr>
<td>Typelib</td>
<td>76</td>
</tr>
<tr>
<td>Common Types</td>
<td>77</td>
</tr>
<tr>
<td>Version Information</td>
<td>79</td>
</tr>
<tr>
<td>Base Info</td>
<td>79</td>
</tr>
<tr>
<td>Callable Info</td>
<td>82</td>
</tr>
<tr>
<td>Signal Info</td>
<td>84</td>
</tr>
<tr>
<td>Function Info</td>
<td>84</td>
</tr>
<tr>
<td>Registered Type Info</td>
<td>87</td>
</tr>
<tr>
<td>Enum Info</td>
<td>88</td>
</tr>
<tr>
<td>Struct Info</td>
<td>90</td>
</tr>
<tr>
<td>Union Info</td>
<td>92</td>
</tr>
<tr>
<td>Object Info</td>
<td>93</td>
</tr>
<tr>
<td>Interface Info</td>
<td>98</td>
</tr>
<tr>
<td>Arg Info</td>
<td>101</td>
</tr>
<tr>
<td>Constant Info</td>
<td>104</td>
</tr>
<tr>
<td>Field Info</td>
<td>105</td>
</tr>
<tr>
<td>Property Info</td>
<td>106</td>
</tr>
<tr>
<td>Type Info</td>
<td>107</td>
</tr>
<tr>
<td>FFI Interface</td>
<td>109</td>
</tr>
<tr>
<td>Utilities</td>
<td>110</td>
</tr>
<tr>
<td>Support</td>
<td>114</td>
</tr>
<tr>
<td>Module</td>
<td>114</td>
</tr>
<tr>
<td>Goops</td>
<td>115</td>
</tr>
<tr>
<td>Enum</td>
<td>116</td>
</tr>
<tr>
<td>Flags</td>
<td>117</td>
</tr>
<tr>
<td>Struct</td>
<td>118</td>
</tr>
<tr>
<td>Union</td>
<td>120</td>
</tr>
<tr>
<td>Utilities</td>
<td>122</td>
</tr>
<tr>
<td>G-Golf High Level API</td>
<td>126</td>
</tr>
<tr>
<td>Closure</td>
<td>126</td>
</tr>
<tr>
<td>Function</td>
<td>127</td>
</tr>
<tr>
<td>Import</td>
<td>133</td>
</tr>
<tr>
<td>Utilities</td>
<td>137</td>
</tr>
</tbody>
</table>
Appendix A  GNU Free Documentation License . 137

Concept Index ............................................. 146

Procedure Index ......................................... 147

Variable Index ........................................... 152

Type Index ................................................ 153

List of Examples ......................................... 154
Preface

This manual describes how to use G-Golf. It relates particularly to G-Golf version 0.1.0.

Contributors to this Manual

Like G-Golf itself, the G-Golf reference manual is a living entity. Right now, the contributor to this manual is:

• David Pirotte

who is also the author and maintainer of G-Golf.

You are most welcome to join and help. Visit G-Golf’s web site at http://www.gnu.org/software/g-golf/ to find out how to get involved.

Join the GNU Project

GNU G-Golf is part of the GNU Operating System, developed by the GNU Project.

If you are the author of an awesome program and want to join us in writing Free (libre) Software, please consider making it an official GNU program and become a GNU Maintainer. You can find instructions on how to do this here (https://www.gnu.org/help/evaluation.html).

You don’t have a program to contribute? Look at all the other ways you may help (https://www.gnu.org/help/help.html).

To learn more about Free (libre) Software, you can read and please share this page (https://gnu.org/philosophy/free-sw.html).

The G-Golf License

GNU G-Golf is Free Software. GNU G-Golf is copyrighted, not public domain, and there are restrictions on its distribution or redistribution:

• GNU G-Golf and supporting files are published under the terms of the GNU Lesser General Public License version 3 or later. See the file LICENSE.

• This manual is published under the terms of the GNU Free Documentation License (see Appendix A [GNU Free Documentation License], page 137).

You must be aware there is no warranty whatsoever for GNU G-Golf. This is described in full in the license.

I. Introduction

About G-Golf

GNU G-Golf
GNOME: (Guile Object Library for).
Description

G-Golf is a Guile\(^1\) Object Library for GNOME (https://www.gnome.org/).

G-Golf low level API comprises a binding to - (most of) the GObject Inter- 
trospection (https://developer.gnome.org/stable/gi) and (some of) 
the GObject (https://developer.gnome.org/gobject/stable/) and Glib 
(https://developer.gnome.org/glib/stable/) libraries, as well as additional (G-Golf) 
utilities - used to import GObject libraries and build their corresponding G-Golf high level 
API.

**Note:** to be precise, G-Golf imports (and depends on the existence of) a Typelib 
(https://gi.readthedocs.io/en/latest) - a binary, readonly, memory-mappable 
database containing reflective information about a GObject library.

G-Golf high level API makes (imported) GObject classes and methods available using 
GOOPS, the Guile Object Oriented System (see Section “GOOPS” in The GNU Guile Reference Manual).

G-Golf is a tool to develop modern graphical applications.

Savannah

GNU G-Golf also has a project page on Savannah (https://savannah.gnu.org/projects/g-golf).

Obtaining and installing G-Golf

GNU G-Golf can be obtained from the following archive site http://ftp.gnu.org/gnu/g-golf/.
The file will be named g-golf-version.tar.gz. The current version is 0.1.0, so the file you 
should grab is:

http://ftp.gnu.org/gnu/g-golf/g-golf-0.1.0.tar.gz

Dependencies

GNU G-Golf needs the following software to run:

- Autoconf $\geq 2.69$
- Automake $\geq 1.14$
- Makeinfo $\geq 6.6$
- Guile (http://www.gnu.org/software/guile) $\geq 2.0.14$ [allows 2.2 3.0]
- Guile-Lib (http://www.nongnu.org/guile-lib) $\geq 0.2.5$
- Glib-2.0 (https://developer.gnome.org/glib/stable/) $\geq 2.48.0$
- GObject-2.0 (https://developer.gnome.org/gobject/stable/) $\geq 2.48.0$
- GObject-Introspection-1.0 (https://developer.gnome.org/stable/gi) $\geq 1.48.0$

Install from the tarball

Assuming you have satisfied the dependencies, open a terminal and proceed with the fol-
lowing steps:

```
cd <download-path>
```

\(^1\) GNU Guile (http://www.gnu.org/software/guile)
an interpreter and compiler for the Scheme (http://schemers.org) programming language.
I. Introduction

```
tar zxf g-golf-0.1.0.tar.gz
cd g-golf-0.1.0
./configure [--prefix=/your/prefix] [--with-guile-site=yes]
make
make install
```

Happy G-Golf (http://www.gnu.org/software/g-golf/)!

### Install from the source

G-Golf (http://www.gnu.org/software/g-golf/) uses Git (https://git-scm.com/) for revision control, hosted on Savannah (https://savannah.gnu.org/projects/g-golf), you may browse the sources repository here (http://git.savannah.gnu.org/cgit/g-golf.git).

There are currently 2 [important] branches: master and devel. G-Golf (http://www.gnu.org/software/g-golf/) stable branch is master, developments occur on the devel branch.

So, to grab, compile and install from the source, open a terminal and:

```
git clone git://git.savannah.gnu.org/g-golf.git
cd g-golf
./autogen.sh
./configure [--prefix=/your/prefix] [--with-guile-site=yes]
make
make install
```

The above steps ensure you’re using G-Golf (http://www.gnu.org/software/g-golf/) bleeding edge stable version. If you wish to participate to developments, checkout the devel branch:

```
git checkout devel
```

Happy hacking!

### Notes:

1. The **default** and **--prefix** installation locations for source modules and compiled files (in the absence of **--with-guile-site=yes**) are:

   $(datadir)/g-golf
   $(libdir)/g-golf/guile/$(GUILE_EFFECTIVE_VERSION)/site-ccache

   If you pass **--with-guile-site=yes**, these locations become the Guile global site and site-ccache directories, respectively.

   The configure step reports these locations as the content of the **sitedir** and **siteccachedir** variables, respectively the source modules and compiled files install locations. After installation, you may consult these variables using pkg-config:

   ```
pkg-config g-golf-1.0 --variable=sitedir
pkg-config g-golf-1.0 --variable=siteccachedir
```

   You will need - unless you have used **--with-guile-site=yes**, or unless these locations are already 'known' by Guile - to define or augment your GUILE_LOAD_PATH and GUILE_COMPILED_PATH environment variables with these locations, respectively (or `%load-path` and

2. G-Golf also installs its `libg-golf.*` library files, in `$\text{libdir}$`. The configure step reports its location as the content of the `libdir` variable, which depends on the content of the `prefix` and `exec_prefix` variables (also reported). After installation, you may consult these variables using `pkg-config`:

\begin{verbatim}
    pkg-config g-golf-1.0 --variable=prefix	pkg-config g-golf-1.0 --variable=exec_prefix
    pkg-config g-golf-1.0 --variable=libdir
\end{verbatim}

You will need - unless the `$\text{libdir}$` location is already 'known' by your system - to either define or augment your `$\text{LD_LIBRARY_PATH}$` environment variable, or alter the `/etc/ld.so.conf` (or add a file in `/etc/ld.so.conf.d`) and run (as root) `ldconfig`, so that G-Golf finds its `libg-golf.*` library files\(^3\).

3. To install G-Golf, you must have write permissions to the default or `$\text{prefix}$` directory and its subdirs, as well as to both Guile's site and site-ccache directories if `--with-guile-site=yes` was passed.

4. Like for any other GNU Tool Chain compatible software, you may install the documentation locally using `make install-info`, `make install-html` and/or `make install-pdf`.

5. Last but not least :)\(^4\), G-Golf comes with a test-suite, which we recommend you to run (especially before [Reporting Bugs], page 5):

\begin{verbatim}
    make check
\end{verbatim}

Contact Information

Mailing list

G-Golf uses Guile's mailing lists:

- `guile-user@gnu.org` is for general user help and discussion.
- `guile-devel@gnu.org` is used to discuss most aspects of G-Golf, including development and enhancement requests.

Please use ‘G-Golf ~ ’ to preceed the subject line of G-Golf related emails, thanks!

You can (un)subscribe to the one or both of these mailing lists by following instructions on their respective list information page (https://lists.gnu.org/mailman/listinfo/).

\(^2\) In this case, you may as well decide to either alter your `$\text{HOME}/.guile$` personal file, or, if you are working in a multi-user environment, you may also opt for a global configuration. In this case, the file must be named `init.scm` and placed it here (evaluate the following expression in a terminal): `guile -c "(display (%global-site-dir))(newline)"`.

\(^3\) Contact your administrator if you opt for the second solution but don't have write privileges on your system.
IRC
Most of the time you can find me on irc, channel #guile, #guix and #scheme on irc.libera.chat, #clutter and #introspection on irc.gnome.org, under the nickname david.

Reporting Bugs
G-Golf uses a bug control and manipulation mailserver. You may send your bugs report here:

- bug-g-golf@gnu.org

You can (un)subscribe to the bugs report list by following instructions on the list information page (https://lists.gnu.org/mailman/listinfo/bug-g-golf).

Further information and a list of available commands are available here (https://debbugs.gnu.org/server-control.html).

II. Using G-Golf

Before you start

Naming Conventions
G-Golf is, or at least tries to be, consistent in the way ‘things’ are being named, whether the functionality being ‘exposed’ is from an imported GNOME library or is part of a G-Golf’s core reference module.

 GNOME Libraries
When G-Golf imports a GNOME library, its classes, properties, methods, functions, types and constant are renamed, which is achieved by calling [g-name-class-name], page 123, and [g-name-name], page 123, appropriately.

As described in their respective documentation entry, as well as in the [Customizing G-Golf], page 10, section, G-Golf offers a way to either ignore or partially customize the renaming process.

- Classes

GNOME libraries classes are imported as GOOPS classes (the Guile Object Oriented System, see Section “GOOPS” in The GNU Guile Reference Manual), and their respective name is given by the result of calling [g-name-class-name], page 123, for example:

- GtkWidget ⇒ <gtk-window>
- ClutterActor ⇒ <clutter-actor>
- WebKitWebView ⇒ <webkit-web-view>\footnote{By default, G-Golf sets WebKit as a renaming exception token, otherwise, the class name would be <web-kit-web-view>.}
II. Using G-Golf

- Properties

GNOME libraries class properties are imported as GOOPS class slots, and their respective name is given by calling [g-name->name], page 123. Each property slot defines an init-keyword and an accessor, following G-Golf’s accessors naming conventions (See [GOOPS Notes and Conventions], page 7).

As an example, the <gtk-label> class has a label slot, with the #:label init-keyword and !label accessor.

- Methods

GNOME libraries methods are imported as GOOPS methods, the name of which is obtained by calling [g-name->name], page 123.

Unless otherwise specified (see [Customization Square], page 32, - GI Method Short Name Skip), as it imports a GI typelib, G-Golf creates a method short name for each imported method, obtained by dropping the container name (and its trailing hyphen) from the GI typelib method long name.

For example, the <gtk-label> class, which defines a gtk-label-get-text method, would also define, using G-Golf’s default settings, an get-text method.

- Functions

GNOME libraries functions are imported as procedures, renamed by calling [g-name->name], page 123. For example:

```r
  gtk_window_new => gtk-window-new
  clutter_actor_new => clutter-actor-new
  ...
```

- Enums, Flags and Boxed types

GNOME libraries enums, flags and boxed types are renamed by calling [g-name->name], page 123, (and cached, See [Cache Park], page 31, section).

Enum and flag type members are renamed by calling [g-name->name], page 123. To illustrate, here is an example:

```r
,use (g-golf)

(gi-import-by-name "Gtk" "WindowPosition")
⇒ $2 = #<<gi-enum> 5618c7a18090>

(describe $2)
⇒ #<<gi-enum> 5618c7a18090> is an instance of class <gi-enum>
⇒ Slots are:
  - enum-set = ((none . 0) (center . 1) (mouse . 2) (center-always . 3) (center-on-parent . 4))
  - g-type = 94664428197600
  - g-name = "GtkWindowPosition"
  - name = gtk-window-position
```

G-Golf Core Reference

- Procedures and Variables
G-Golf procedure names that bind a Glib, GObject or GObject Introspection functions (always) use the ‘original’ name, except that every _ (underscore) occurrence is replaced by a - (hyphens). For example:

```
    g_main_loop_new
⇒ [g-main-loop-new], page 44
```

```
    g_irepository_get_loaded_namespaces
⇒ [g-irepository-get-loaded-namespaces], page 75
```

G-Golf also comes with its own set of procedures, syntax and variables, aimed at not just reading a typelib, but making its functionality available from Guile (http://www.gnu.org/software/guile). Naming those, whenever possible, is done following the ‘traditional way’ scheme name its procedures, syntax and variables. For example:

- procedure names that start with call-with-input-, call-with-output- followed by a Glib, GObject. Gdk or GI type, such as:
  
  [call-with-input-typelib], page 77

- syntax names that start as with- followed by a Glib, GObject, Gdk or GI type, such as:
  
  [with-gerror], page 112

When an ‘obvious’ name can’t be find ‘on its own’, or to avoid possible conflict outside G-Golf, then the name starts using a g- prefix (when the procedure context is GNOME in general) or gi- prefix (when the procedure context is GI more specifically), and equally for variables, using %g- or %gi-.

- Types and Values

G-Golf variables that bind Glib, GObject and GI types and values use the same convention as for procedures, except that they always start with % and their original type names are transformed by the same rules that those applied when calling [g-study-caps-expand], page 123.

For example, from the GIBaseInfo section:

```
    GIIInfoType
⇒ [%gii-info-type], page 82
```

### GOOPS Notes and Conventions

G-Golf extensively uses GOOPS, the Guile Object Oriented System (see Section “GOOPS” in The GNU Guile Reference Manual), in a way that is largely inspired by Guile-Gnome (https://www.gnu.org/software/guile-gnome).

Here are some notes and the GOOPS conventions used by G-Golf.

- Slots are not Immutable

---

5 As an example, it would not be a good idea to use (the name) import for the G-Golf procedure that reads and build the interface for a GIR library, since it is an R6RS reserved word.
Except for virtual slots, there is currently no way to effectively prohibit (block) a user to mutate a goops class instance (one can always use `slot-set! instance slot-name value`).

However, you will find a few places in this manual using phrase excerpts like ‘instances of this <class> are immutable’, or ‘this <slot> is immutable’. In these contexts, what is actually meant is that these (instances or slots) are not meant to be mutated. Doing so is not only at your own risks, but likely to cause a crash.

- **Merging Generics**

In G-Golf, generic functions are always merged (see Section “Merging Generics” in *The GNU Guile Reference Manual*).

Users are (highly) recommended to do the same, in their repl, application/library modules and script(s). In its modules - those that import (oop goops) - G-Golf uses the following duplicate binding handler set:

```scheme
#:duplicates (merge-generics replace warn-override-core warn last)
```

In a repl or in scripts, these maybe set - after importing (oop goops) - by calling `default-duplicate-binding-handler`:

```scheme
(use-modules (oop goops))

(default-duplicate-binding-handler
 '(merge-generics replace warn-override-core warn last))
```

G-Golf regular users should consider adding the above lines to their `$HOME/.guile` or, when working in a multi-user environment, should consider adding those lines the file named `init.scm` in the so-called Guile global site directory, here (evaluate the following expression in a terminal): `guile -c "(display (%global-site-dir))(newline)"`.

- **Accessors Naming Convention**

In G-Golf, all slots define an accessor (and no getter, no setter), the name of which is the `slot-name` prefixed using `!`. For example:

```scheme
(define-class <gtype-class> (<class>)
  (info #:accessor !info
       #:init-keyword #:info)
  ...
)
```

The principal reasons are (not in any particular order):

---

6 Actually, to be complete, there is a way, which is to define the slot using `#:class <read-only-slot>`, but (a) it is undocumented and (b), it requires the use use of libguile to initialize the slot value, something that I don’t want to do in G-Golf. If you are interested by this (undocumented) feature for your own project though, I suggest you look for some examples in the Guile-Gnome (https://www.gnu.org/software/guile-gnome), source tree, where it is extensively used.

7 You need write privileges to add or modify this file, contact your system administrator if you’re not in charge of the system you are working on.
II. Using G-Golf

- It is a good idea, we think, to be able to visually (and somehow immediately) spot and distinct accessors from the rest of the scheme code you are looking at or working on.
- Accessors are exported, and with this convention, we almost certainly avoid all 'name clashes' with user namespaces, that otherwise would be extremelly frequent.
- Users quite often want or even need to cash slot values in a closure. By using this ! prefixing convention, we leave users with the (quite usefull) possibility to name their local variables using the respective slot names.
- Accessors may always be used to mutate a slot value (except for virtual slots, for which you can 'block' that feature), like in (set! (!name an-actor) "Mike"). In scheme, it is a tradition to signal mutability by postfixing the procedure name using the ! character.
- Accessors are not procedures though, there are methods, and to effectively mutate a slot value, one must use set!. Therefore, prefixing makes sense (and preserves the first reason announced here, where posfixing would break it).
- We should also add that we are well aware that Java also prefixes its accessors, using a . as its prefix character, but GOOPS is radically different from Java in its design, and therefore, we really wanted another character.

Configuring Guile for G-Golf

The following description and content is shared and identical to the 'Merging Generics' heading of the previous section.

It is repeated it here, under its own section entry, so that it appears in the table of content and grab all users attention - those who do not follow our recommendation may void their warranty or poison their cat.

- Merging Generics

In G-Golf, generic functions are always merged (see Section "Merging Generics" in The GNU Guile Reference Manual).

Users are (highly) recommended to do the same, in their repl, application/library modules and script(s). In its modules - those that import (oop goops) - G-Golf uses the following duplicate binding handler set:

```scheme
#:duplicates (merge-generics
replace
warn-override-core
warn
last)
```

In a repl or in scripts, these maybe set - after importing (oop goops) - by calling default-duplicate-binding-handler:

```scheme
(use-modules (oop goops))
(default-duplicate-binding-handler
```

---

8 Slot names tends to be extremelly common, like name, color, ... and naming their respective accessor using the slot name would very likely provoke numerous name clashes with user variables, procedures and methods names.
II. Using G-Golf

'(merge-generics replace warn-override-core warn last))

G-Golf regular users should consider adding the above lines to their 
$HOME/.guile or, when working in a multi-user environment, should consider adding those lines the file named init.scm in the so-called Guile global site directory, here (evaluate the following expression in a terminal): guile -c "(display (%global-site-dir))(newline)".

Customizing G-Golf

G-Golf offers a series of customization interfaces for the following domains: (●) Name Transformation - how things are being named as they are being imported; (●) Strip Boolean Result - should G-Golf elude (some) function and method call returned value when it is #t and raise an exception if the returned value is #f; (●) Method Short Name - should G-Golf create them or not; (●) Syntax Name Protect - how G-Golf should address syntax name 'clash' against method short name.

- Name Transformation

When G-Golf imports a GNOME library, its classes, properties, methods, functions, types and constants are renamed (See [Naming Conventions], page 5), mainly to (a) avoid ‘Camel Case (https://en.wikipedia.org/wiki/Camel_case), (b) surround class names by ‘<’ ‘>’ and (c) replace '_' (underscore) occurrences using the '-' (hyphen) character instead.

G-Golf offers - through a series of interfaces to get, check, add, remove and reset two (distinct) associative lists - a way to either ignore or partially customize the renaming process.

See [Customization Square], page 32, - GI Name Transformation.

- Strip Boolean Result

Some GI typelib functions and methods that (1) have at least one 'inout or 'out argument(s) and (2) return either #t or #f, solely to indicate that the function or method call was successful or not.

G-Golf offers - through a series of interfaces to get, check, add, remove and reset a list of such function or methods names - to instead elude the function or method returned value when it is #t and raise an exception if the returned value is #f.

See [Customization Square], page 32, - GI Strip Boolean Result.

- Method Short Name

By default, as it imports a GI typelib, G-Golf creates a method short name for each imported method, obtained by dropping the container name (and its trailing hyphen) from the GI typelib method full/long name.

Users may change this default and skip the method short name creation step, either individually or for all GI imported methods.

See [Customization Square], page 32, - GI Method Short Name Skip.

- Syntax Name Protect

---

9 You need write privileges to add or modify this file, contact your system administrator if you’re not in charge of the system you are working on.
II. Using G-Golf

When G-Golf creates a method short name, obtained by dropping the container name (and its trailing hyphen) from the GI typelib method full/long name, it may lead to a ‘name clash’, with an already defined procedure or syntax.

Both type of ‘name clash’ need to be addressed, which G-Golf does, automatically, but special care must be taken when that happens against a syntax name, a process that you may custom to your own taste.

See [Customization Square], page 32, - GI Syntax Name Protect.

Getting Started with G-Golf

G-Golf will let you import and work with any GObject-Introspectable GNOME library\(^{10}\). Since we need to make a choice among so many, to guide new comers and get them started with G-Golf, let’s pick-up Gtk (https://developer.gnome.org/gtk4/stable/), and show how to Create interfaces that users just love (https://gtk.org/).

Please note that in the entire course of the G-Golf manual, unless otherwise specified, examples are based on and use Gtk-4.0 (https://developer.gnome.org/gtk4/stable/), Gdk-4.0 (https://developer.gnome.org/gdk4/stable/) and Gsk-4.0 (https://developer.gnome.org/gsk4/stable) - which is new and only available with Gtk-4.0.


We shall complete this brief introduction mentioning that the GNOME team wrote a guide to help Migrating from GTK 3.x to GTK 4 (https://developer.gnome.org/gtk4/stable/gtk-migrating-3-to-4.html).

Hello World!

Following the tradition, let’s first see how the often seen ‘Hello World!’ familiar, minimal, friendly greeting program looks like in G-Golf:

```g-golf
;; Load Gtk
(use-modules (g-golf))
(gi-import "Gtk")

;; When the application is launched..
(define (activate app)
  ;; - Create a new window and a new button
  (let ((window (make <gtk-application-window>
                   #:title "Hello"
                   #:application app))
        (button (make <gtk-button>
                   #:label "Hello, World!")
        )))

  ;; - Which closes the window when clicked
  (connect button
             'clicked
             (lambda (b
            
```

\(^{10}\) In its compiled form, a GObject-Introspectable GNOME library is called a Typelib (https://gi.readthedocs.io/en/latest) - a binary, readonly, memory-mappable database containing reflective information about a GObject library.
II. Using G-Golf

```lisp
(close window))
(set-child window button)
(show window))

;; Create a new application
(let ((app (make <gtk-application>
#:application-id "org.example.GtkApplication")))
  (connect app 'activate activate)
;; Run the application
(run app 0 '()))
```

Providing you successfully installed G-Golf, you may run the above code in a Guile REPL (Read Evaluate Print Loop)\(^\text{11}\), which as described in its comments, starts the application, resulting in opening a (small) window named ‘Hello’, with one button named ‘Hello, World!’, that will close the window when clicked.

Example 1

Wonderful! But you probably rightfully think that it was a bit slow. This is not because G-Golf nor Guile are slow, but because the Gtk namespace is absolutely huge, and although we only use a few components, we asked to import the all namespace. We will see how to only selectively import the namespace components we need in the next section, but let’s first try the following, (a) close the window and (b) re-evaluate the last expression:

```lisp
(let ((app (make <gtk-application>
#:application-id "com.example.GtkApplication")))
  (connect app 'activate activate)
(run app 0 '()))
```

Great! Now, the application was launched instantaneously. Since everything it needs was already imported, the time it takes to execute the code is nearly identical to the time it would take to execute the same code from C - if you accurately measure the execution time in both situation, you would see a difference in the results, but small enough that it is safe to declare it imperceptible.

It would be beyond the scope of this introduction to describe the `<gtk-application>` / `g-application-run` instance creation and run mechanism in detail, for this, please consult and carefully read their respective entries in the Gtk (https://developer.gnome.org/gtk4/stable/GtkApplication.html) and Gio (https://developer.gnome.org/gio/stable/GApplication.html) reference manuals.

The GNOME team also maintains a wiki called HowDoI (https://wiki.gnome.org/HowDoI), and two pages are dedicated to this subject: HowDoI GtkApplication (https://wiki.gnome.org/HowDoI/GtkApplication) and HowDoI GtkApplication/CommandLine (https://wiki.gnome.org/HowDoI/GtkApplication/CommandLine).

This said, let’s just make a few hopefully usefull comments to newcomers:

\(^\text{11}\) If you haven’t done so, please read the [Configuring Guile for G-Golf], page 9, *Merging Generics* and configure your repl as proposed, before to run the example.
II. Using G-Golf

- as you can see, we do not need to call gtk-init, it is done automatically (more on this in the GtkApplication (https://developer.gnome.org/gtk4/stable/GtkApplication.html) section of the Gtk Reference Manual);

- the #:application-id init-keyword is optional, although recommended, and when passed, the application ID must be valid (more on this below).

  - Is your application ID valid?


In G-Golf, you may check if your application ID is valid by calling g-application-id-is-valid\textsuperscript{12}, for example:

\[
\begin{align*}
\text{(g-application-id-is-valid "com.example.GtkApplication")} & \Rightarrow \text{#t} \\
\text{(g-application-id-is-valid "RedBear")} & \Rightarrow \text{#f}
\end{align*}
\]

If you pass an invalid application ID to a <gtk-application> instance creation, you’ll be noted with a message similar to this:

\[(\text{process:30818): GLib-GIO-CRITICAL **: 21:58:52.700: g_application_set_application_id: assertion 'application_id == NULL || g_application_id_is_valid (application_id)' failed}\]

- Great, but could we speed things up a little?

Yes we can! In the next section, as promised above, we will walk you through [Selective Import], page 13, used to reduce the time G-Golf has to spend importing the typelib(s) that your application requires.

Selective Import

To selectively import namespace components, use [gi-import-by-name], page 19, which takes two arguments, a namespace and a (component) name. Let’s try on our minimal ‘Hello World!’ example and see how it goes. All we need to do, is to substitute the (gi-import "Gtk") call by the following expression:

\[(\text{for-each (lambda (name) (gi-import-by-name "Gtk" name))} \text{'("Application" "ApplicationWindow" "Button")})\]

With this change, everything else kept equal, if you (quit and) restart Guile, evaluate the updated ‘Hello World!’ example code, you will notice how the elapse time before the application window appears is now substantially reduced, compared to the version that

\textsuperscript{12} After you at least import either directly (gi-import-by-name "Gio" "Application"), or (gi-import-by-name "Gtk" "Application"), which triggers the appropriate Gio imports, as described in the next section
II. Using G-Golf

imports the all Gtk namespace. Substantially reduced but ... not instantaneous: well, that
is expected!

Although we only import a few Gtk namespace components, three GObject classes in this
example, G-Golf will import those classes, their interface(s) if any, methods, enums, flags ...
and do the same for their parent class, recursively. For those three classes only, G-Golf
actually has to import (and dynamically define) tens of classes, interfaces, enums, flags . . .
as well as hundreds of methods and procedures.

G-Golf will also import classes, interfaces and their dependencies (enums, flags . . . recur-
sively as well . . .) from other namespace if necessary. We already have an illustration of
this, both with the original example and the change we just made: although we do not
explicitly import the GApplication class from the Gio namespace, G-Golf did that for us,
and so we may call run - which is the short method name for g-application-run - as if
we did manually import it.

Both the namespace and name arguments are case sensitive. The name argument is used
to retrieve the typelib [Base Info], page 79, that holds the metadata of the introspectable
library element it represents. Although there are a some exceptions, it is generally derived
from and obtained by dropping the namespace prefix (without its version number if any)
out of the original name. Here are a few more examples, organized by namespace:

    Gtk        GtkWindow -> Window
               gtk_init -> init
               gtk_main -> main
               gtk_main_quit -> main_quit
               ...

    WebKit2    WebKitWebView -> WebView
               WebKitLoadEvent -> LoadEvent
               ...

- Cool, selective import, but what about scripting?

Right! The 'Hello World!' example we have presented so far can only be run interactively.
In the next section, we will see how we may turn it - and any other example or application
- so it can be run as a script.

Scripting

A Guile script is simply a file of Scheme code with some ‘extra information at the
beginning’ which tells the OS (operating system) how to invoke Guile, and then tells
Guile how to handle the Scheme code.

- Invoking Guile

It would be beyond the scope of this manual to expose the numerous ways one can define
and invoke a Guile script, for a complete description of the subject, see Section “Guile

In G-Golf, both provided examples and in this manual, we use the so called ‘for maximum
portability’ scripting technique, which is to invoke the shell to execute guile with specified
command line arguments.
II. Using G-Golf

Here is what we do:

```bash
#!/bin/sh
# -*- mode: scheme; coding: utf-8 -*-
exec guile -e main -s "$0" "$@

"$@
```

In the above, the first line is to specify which shell will be used to interpret the (OS part of the) `extra information at the beginning` of the script.

The second line is optional (and a comment from a shell point of view), that we use it to inform emacs (should you use emacs to edit the file) that despite the `extra information at the beginning` (and the possible lack of filename extension in the script name), it should use the `scheme` mode as the script editing buffer mode.

The third line tells the shell to execute guile, with the following arguments:

- `-e main` after reading the script, apply `main` to command line arguments
- `-s "$0"` load the source code from "$0" (which by shell rules, is bound to the fullname of the script itself)

Note that the top level script lines may contain other declaration(s), like environment variable definitions. Suppose you would like to be warned if your script uses any deprecated guile functionality. In this case, you add the following `export GUILE_WARN_DEPRECATED="detailed"` declaration, before the `exec guile ...` call, like this:

```bash
#!/bin/sh
# -*- mode: scheme; coding: utf-8 -*-
export GUILE_WARN_DEPRECATED="detailed"
exec guile -e main -s "$0" "$@

"$@
```

- Extra Guile information

Within the context of a G-Golf script, two other things must be taken care of - in addition to the `(use-modules (g-golf))` step - so that the script runs fine: (1) set-up Guile so that generic functions are merged; (2) import (all) typelib element(s) at expand load eval time.

In a repl or in scripts, (1) is achieved by importing the `(oop goops)` module and calling `default-duplicate-binding-handler`\(^{13}\).

In Guile, (2) is achieved by calling the `eval-when` syntax\(^{14}\).

Now, bear with us :) , since (2) will define generic functions and/or add methods to existing generic functions, we must make sure the (1) not only preceeds (2), but also happens at expand load eval time.

---

\(^{13}\) As seen in [Configuring Guile for G-Golf], page 9, (and in [GOOPS Notes and Conventions], page 7, - 'Merging Generics').

With all the above in mind, here is how the extra Guile information looks like, for our ‘Hello World!’ script example:

```scheme
(eval-when (expand load eval)
  (use-modules (oop goops))

  (default-duplicate-binding-handler
   '(merge-generics replace warn-override-core warn last))

  (use-modules (g-golf))

  (for-each (lambda (name)
             (gi-import-by-name "Gtk" name))
    '("Application"
      "ApplicationWindow"
      "Button")))

- A Hello World! script

Let’s put all this together, and while doing this, enhance a little our original example.

Here is what we propose to do: (a) add a GtkLabel, (b) use a GtkBox and see how to declare its margins and orientation, (c) specify a default width and height for our application window, and (d) see how we can tell the label to horizontally and vertically expand, so it occupies the extra vertical space, while keeping the button to its minimal vertical size.

Joining (1), (2) and the small enhancement, our ‘Hello World!’ script now looks like this:

```bash
#!/bin/sh
#
# -*- mode: scheme; coding: utf-8 -*-
exec guile -e main -s "$0" "$@
!#
```

```scheme
(eval-when (expand load eval)
  (use-modules (oop goops))

  (default-duplicate-binding-handler
   '(merge-generics replace warn-override-core warn last))

  (use-modules (g-golf))

  (for-each (lambda (name)
             (gi-import-by-name "Gtk" name))
    '("Application"
      "ApplicationWindow"
      "Box"
      "Label"
      "Button")))
```
II. Using G-Golf

(define (activate app)
  (let ((window (make <gtk-application-window>
                   #:title "Hello"
                   #:default-width 320
                   #:default-height 240
                   #:application app))
      (box (make <gtk-box>
             #:margin-top 6
             #:margin-start 12
             #:margin-bottom 6
             #:margin-end 6
             #:orientation 'vertical))
      (label (make <gtk-label>
              #:label "Hello, World!"
              #:hexpand #t
              #:vexpand #t))
      (button (make <gtk-button>
               #:label "Close")))

      (connect button
                 'clicked
                 (lambda (b)
                           (close window)))

      (set-child window box)
      (append box label)
      (append box button)
      (show window)))

(define (main args)
  (let ((app (make <gtk-application>
               #:application-id "org.gtk.example")))
    (connect app 'activate activate)
    (let ((status (run app 0 '())))
      (exit status))))

If you save the above in a file, say hello-world, then chmod a+x hello-world and launch the script, ./hello-world, here is what you’ll get on the screen:

Example 2

- A last few comments

We need to make a last few comments, that also applies and will be further addressed in the next section.

Desktop Entry
II. Using G-Golf

If you are running a GNOME desktop, you probably noticed that in the GNOME menu bar, the application menu entry for our ‘Hello World!’ script is `org.gtk.example` (not Hello). This is because we’re missing a Desktop Entry. We will see how to create and install a Desktop Entry in the next section.

Command Line Arguments

As described in the first part of this section, we use the so called ‘for maximum portability’ scripting technique, and more precisely, the following incantation:

```
exec guile -e main -s "$0" "$@
```

In the above, the last argument refers to the the command line arguments. It is actually optional, but when used, they are passed to the `main` (entry point) script procedure.

However, as you may have noticed, we do not pass those (if any) to the Gtk application, which we launch using `(run app 0 '())`.

This is intentional: (a) we (want to) always use the same incantation to invoke Guile - and sometimes, may quickly hack something using additional debug args on the scheme side only . . .; (b) you may only pass those arguments to the Gtk application if you have defined the signal callback(s) to handle them.

If you pass the command line arguments to a Gtk application that does not define the appropriate signal callback procedure to handle them, you’ll get an error message in the terminal (and the application won’t be launched).

To illustrate, let’s change the `g-application-run` call of our script, so it becomes `(run app (length args) args)`, then try to launch it, passing a few (fake) arguments, here is what happens:

```
./hello-world 1 2 3
```

```
(./hello-world:216198): GLib-GIO-CRITICAL **: 22:26:41.135: This application can not open files.
```

And as mentioned above, the application is not launched.

Although scripts may (also) accept and pass command line argument(s) to the Gtk application or dialog they define, we will see how to handle those in the next section, [Building Applications], page 18.

Building Applications

G-Golf on Mobile Devices

Working with GNOME

Working with GNOME exposes, grouped by theme, the user interfaces to import and work with GObject-Introspectable GNOME libraries.

Please note that within the scope of the G-Golf manual in general, in the sections presented here in particular, we simply (as in merely and in the simplest possible way) exposes the scheme representation and G-Golf interfaces of the elements that are being addressed. For a deep(er) understanding of the original concepts, components and interfaces, you must refer to the upstream library documentation itself.

This is particularly true for the GLib Object System related sections. For a thorough understanding of the GLib Object System - its background, design goals, dynamic
II. Using G-Golf

Using G-Golf 19 type system, base class instantiation, memory management, properties, closures and signals messaging system - please consult the GObject - Type System Concepts (https://docs.gtk.org/gobject/concepts.html) of the GObject reference manual.

Import

G-Golf Import interfaces.
Importing GNOME libraries.

Procedures

- [gi-import], page 19
- [gi-import-by-name], page 19

Description

The G-Golf GIR namespace (Typelib) import interfaces.

Procedures

**gi-import** namespace [#:version #f]

[Procedure]

Returns nothing.
Imports the namespace GIR Typelib and exports its interface. For example:

```lisp
,use (g-golf
gi-import "Clutter")
```

The namespace is a case sensitive string. It is an error to call this procedure using an invalid namespace.

The optional #:version keyword argument may be used to require a specific namespace version, otherwise, the latest will be used.

This procedure is certainly one of the first thing you will want to try and use, but it has a cost: you will not ‘feel it’ if the number of objects in namespace is relatively small, but importing the "Gtk" namespace, on a laptop equipped with an i5-2450M CPU 2.50GHz × 4 and 6GB of memory takes nearly 2 seconds.

So, either early in the development cycle, or when your application is more stable, at your best convenience, you may consider making a series of selective import instead, see [gi-import-by-name], page 19, here below.

**gi-import-by-name** namespace name [#:version #f] [#:with-method #t]

[Procedure]

Returns the object or constant returned by [gi-import-info], page 134, called upon the GIBaseInfo info named name in namespace.

Obtains and imports the GIBaseInfo info named name in namespace. The namespace and name arguments are case sensitive. It is an error to call this procedure using an invalid namespace or name.

The optional #:version keyword argument may be used to require a specific namespace version, otherwise, the latest will be used.

The optional keyword #:with-method argument - which is #t by default - is passed to the gi-import-enum, gi-import-flags and gi-import-struct. When #:with-
Using G-Golf

If the method is `#f`, then the enum, flags or struct info will be imported without their respective methods. This is likely to only be the case if/when you intend to selectively import an enum, flags or struct from GLib or GObject, which is what G-Golf itself does, for example, in the top level (g-golf) module:

```
(gi-import-by-name "GLib" "IOChannel" #:with-method #f)
```

Events

G-Golf Events interfaces.
Handling events from the window system.

- Special Note -

Most of the numerous, important and sometimes radical changes in between Gtk-3.0 (https://developer.gnome.org/gtk3/stable) and Gdk-3.0 (https://developer.gnome.org/gdk3/stable) and Gtk-4.0 (https://developer.gnome.org/gtk4/stable), Gdk-4.0 (https://developer.gnome.org/gdk4/stable) and Gsk-4.0 (https://developer.gnome.org/gsk4/stable) have had no impact on G-Golf. And by most, we actually mean all but one: the GdkEvent and its API.

For this reason, this section is split/organized in two subheading, namely ‘In Gdk-3.0’ and ‘In Gdk-4.0’, how creative :), that expose their respective G-Golf interfaces.

- In Gdk-3.0

In Gdk-3.0 (https://developer.gnome.org/gdk3/stable), a GdkEvent contains a union of all of the event types. Data fields may be accessed either directly, direct access to GdkEvent structs, or using accessors (but not all data fields have an accessor).

In G-Golf however GdkEvent is a class, with an event slot - holding a pointer the Gdk event - all other slots are virtual and define an accessor, which is the only way users may retrieve data fields.

When G-Golf detects it is leading with GdkEvent from Gdk-3.0, while dynamically implementing the above, in addition, when applicable, it will also add some of the upstream GdkEvent accessor name to the GI Strip Boolean Result list. This is further detailed below, at the end of the section.

Class

[<gdk-event>], page 21
II. Using G-Golf

Accessors

[!event], page 24
[!axis], page 24
[!button], page 24
[!click-count], page 24
[!coords], page 24
[!device], page 24
[!device-tool], page 24
[!event-sequence], page 24
[!event-type], page 24
[!keycode], page 24
[!keyval], page 24
[!pointer-emulated], page 24
[!root-coords], page 24
[!scancode], page 24
[!screen], page 24
[!scroll-deltas], page 24
[!scroll-direction], page 24
[!seat], page 24
[!source-device], page 24
[!state], page 24
[!time], page 24
[!window], page 24
[!keyname], page 25
[!x], page 25
[!y], page 25
[!root-x], page 25
[!root-y], page 25

Class

Class `<gdk-event>`

It is an instance of `<class>`.

Superclasses are:

`<object>`

Class Precedence List:

`<gdk-event>
<object>
<top>`

Direct slots are:

```lisp
(event #:accessor !event #:init-keyword #:event)
```
II. Using G-Golf

A pointer to a GdkEvent.

axis

#:accessor !axis
#:allocation #:virtual

button

#:accessor !button
#:allocation #:virtual

click-count

#:accessor !click-count
#:allocation #:virtual

coords

#:accessor !coords
#:allocation #:virtual

device

#:accessor !device
#:allocation #:virtual

device-tool

#:accessor !device-tool
#:allocation #:virtual

event-sequence

#:accessor !event-sequence
#:allocation #:virtual

event-type

#:accessor !event-type
#:allocation #:virtual

keycode

#:accessor !keycode
#:allocation #:virtual

keyval

#:accessor !keyval
II. Using G-Golf

#:allocation #:virtual

pointer-emulated

#:accessor !pointer-emulated
#:allocation #:virtual

root-coords

#:accessor !root-coords
#:allocation #:virtual

scancode

#:accessor !scancode
#:allocation #:virtual

screen

#:accessor !screen
#:allocation #:virtual

scroll-deltas

#:accessor !scroll-deltas
#:allocation #:virtual

scroll-direction

#:accessor !scroll-direction
#:allocation #:virtual

seat

#:accessor !seat
#:allocation #:virtual

source-device

#:accessor !source-device
#:allocation #:virtual

state

#:accessor !state
#:allocation #:virtual
II. Using G-Golf

```lisp
(time #:accessor !time #:allocation #:virtual

(window #:accessor !window #:allocation #:virtual

(keyname #:accessor !keyname #:allocation #:virtual

(x #:accessor !x #:allocation #:virtual

(y #:accessor !y #:allocation #:virtual

(root-x #:accessor !root-x #:allocation #:virtual

(root-y #:accessor !root-y #:allocation #:virtual

(event (inst <gdk-event>) [Accessor]
  Returns the content of the event slot for inst, a pointer to a GdkEvent.

!axis (inst <gdk-event>) [Accessor]
!button (inst <gdk-event>) [Accessor]
!click-count (inst <gdk-event>) [Accessor]
!coords (inst <gdk-event>) [Accessor]
!device (inst <gdk-event>) [Accessor]
!device-tool (inst <gdk-event>) [Accessor]
!event-sequence (inst <gdk-event>) [Accessor]
!event-type (inst <gdk-event>) [Accessor]
!keycode (inst <gdk-event>) [Accessor]
!keyval (inst <gdk-event>) [Accessor]
!pointer-emulated (inst <gdk-event>) [Accessor]
!root-coords (inst <gdk-event>) [Accessor]
!scancode (inst <gdk-event>) [Accessor]
```
II. Using G-Golf

Respectively returns the scheme representation of the content of the \texttt{inst} event (struct) element - referred to by its name. It is an error to call an accessor on a \texttt{inst} for which the event (struct) does not deliver the element.

Internally, each of the above \texttt{<gdk-event>} accessor calls the corresponding \texttt{GdkEvent} accessor, passing the content of the \texttt{event} slot. For example, let's see what happens when a user performs a left button (single) click upon a widget that tracks the \texttt{'button-press-event} signal callback:

\[
\begin{align*}
&(!\texttt{button} \ \texttt{inst}) \\
&\mapsto \ (gdk-event-get-button (!\texttt{event} \ \texttt{inst})) \\
&\mapsto 1
\end{align*}
\]

\[
\begin{align*}
&(!\texttt{click-count} \ \texttt{inst}) \\
&\mapsto \ (gdk-event-get-click-count (!\texttt{event} \ \texttt{inst})) \\
&\mapsto 1
\end{align*}
\]


To complete the above listed \texttt{<gdk-event>} virtual slots and accessors automatically provided by introspecting \texttt{GdkEvent}, G-Golf also defines a few additional rather convenient virtual slots and accessors:

\texttt{!keyname (inst <gdk-event>)}

Returns the key (symbol) name that was pressed or released.

Note that there is actually no such element in any (gdk) event. This accessor calls \texttt{gdk-keyval-name} on the keyval of the event). Here is what happens if a user press the ‘a’ keyboard key in a widget that tracks the \texttt{'key-press-event} signal callback:

\[
\begin{align*}
&(!\texttt{keyname} \ \texttt{inst}) \\
&\mapsto \ (gdk-keyval-name (!\texttt{keyval} \ \texttt{inst})) \\
&\mapsto \ (gdk-keyval-name (gdk-event-get-keyval \ \texttt{inst})) \\
&\mapsto \text{a}
\end{align*}
\]

\texttt{!x (inst <gdk-event>)}

\texttt{!y (inst <gdk-event>)}

\texttt{!root-x (inst <gdk-event>)}

\texttt{!root-y (inst <gdk-event>)}

Respectively returns the x, y, root-x and root-y coordinate for \texttt{inst}.
II. Using G-Golf

The result is simply obtained by destructuring and selecting one of the [!coords], page 24, and [!root-coords], page 24, list values, respectively.

**Strip Boolean Result**

If you are not (yet) familiar with the concept we are dealing with here, make sure you visit and read the [Customization Square], page 32, - *GI Strip Boolean Result* section of the manual.

When G-Golf detects it is leading with GdkEvent from Gdk-3.0, while dynamically implementing the [<!gdk-event>], page 21, class and its accessors, it will add the following names to the *GI Strip Boolean Result* list:

```
gdk-event-get-axis
gdk-event-get-button
gdk-event-get-click-count
gdk-event-get-coords
gdk-event-get-keycode
gdk-event-get-keyval
gdk-event-get-root-coords
gdk-event-get-scroll-deltas
gdk-event-get-scroll-direction
gdk-event-get-state
```

- **In Gdk-4.0**

In Gdk-4.0 ([https://developer.gnome.org/gdk4/stable/](https://developer.gnome.org/gdk4/stable/)), GdkEvent is a class\(^{15}\). GdkEvent structs are opaque and immutable. Direct access to GdkEvent structs is no longer possible in GTK 4. All event fields have accessors.

In G-Golf - as in Gdk-4.0 GdkEvent is a class - no special treatment is performed anymore. In particular, no virtual slot is defined and users must access the GdkEvent structs data fields using the accessors provided by Gdk-4.0.

**GObject**

G-Golf GObject interfaces.

The G-Golf integration with the GLib Object System.

- **Special Note** -

For completion, this section exposes the definition of the classes and metaclasses involved in the G-Golf integration of the GLib Object System. From a (strict) user point of view however, these are actually G-Golf internals and, unless you are interested of course, might be ignored.

What you actually really need to know, as a G-Golf user, is mostly (a) the upstream reference manual of the GNOME library(ies) you intend to use, (b) how to program in Guile Scheme of course, and (c) the basics of the Guile Object Oriented System.

\(^{15}\) From a GI point of view - internally, it is a C struct.
It doesn’t hurt if you are, or if you are willing to become one, but we would like to emphasize that you do not need to be a Guile Object Oriented System expert to use G-Golf. What you need to know, with that respect, is somehow largely covered by the [Getting Started with G-Golf], page 11, sections, the description of this (and related) sections and in the examples that come with G-Golf.

Classes

- `<gobject>`, page 28
- `<ginterface>`, page 28
- `<object-class>`, page 28
- `<gtype-class>`, page 29
- `<gtype-instance>`, page 30

Procedures, Accessors and Methods

- `<object-class?>`, page 30
- `[!info]`, page 29
- `[!derived]`, page 29
- `[!namespace]`, page 29
- `[g-type]`, page 29
- `[g-name (2)]`, page 29
- `[g-class]`, page 29
- `[g-inst]`, page 30
- `[unref]`, page 30

Description

GObject\(^{16}\) is the GLib Object System.


All the GNOME libraries that use the GLib type system inherit from GObject ([https://developer.gnome.org/gobject/stable/gobject-The-Base-Object-Type.html](https://developer.gnome.org/gobject/stable/gobject-The-Base-Object-Type.html)), the base object type, which provides methods for object construction and destruction, property access methods, and signal support.

---

\(^{16}\) The name GObject, depending on the context, can actually be used and refer to the GLib Object System ([https://developer.gnome.org/gobject/stable/](https://developer.gnome.org/gobject/stable/)) language system as a whole, or be used and refer to the fundamental type implementation, the base object type ([https://developer.gnome.org/gobject/stable/gobject-The-Base-Object-Type.html](https://developer.gnome.org/gobject/stable/gobject-The-Base-Object-Type.html)), upon which GNOME libraries object hierarchies are based.

\(^{17}\) The name GType, depending on the context, can actually be used and refer to the The GLib Dynamic Type System ([https://developer.gnome.org/gobject/stable/chapter-gtype.html](https://developer.gnome.org/gobject/stable/chapter-gtype.html)), or be used and refer to the type it denotes, a unique ID (Identifier) - an `unsigned-long` to be precise.
G-Golf uses GOOPS\textsuperscript{18} and defines the [\texttt{gobject}], page 28, class, from which all imported GNOME libraries inherit, as their class hierarchy is being built in Guile Scheme.

**Classes**

\texttt{<gobject>}

[Class]

The base class of the GLib Object System.

It is an instance of [\texttt{gobject-class}], page 28.

Superclasses are:

\texttt{<gtype-instance>}

Class Precedence List:

\texttt{<gobject>}

\texttt{<gtype-instance>}

\texttt{<object>}

\texttt{<top>}

(No direct slot)

\texttt{<ginterface>}

[Class]

The base class for GLib’s interface types. Not derivable in Scheme.

It is an instance of [\texttt{gobject-class}], page 28.

Superclasses are:

\texttt{<gtype-instance>}

Class Precedence List:

\texttt{<ginterface>}

\texttt{<gtype-instance>}

\texttt{<object>}

\texttt{<top>}

(No direct slot)

\texttt{<object-class>}

[Class]

The metaclass of the [\texttt{gobject}], page 28, and [\texttt{ginterface}], page 28, classes.

It is an instance of [\texttt{class}].

Superclasses are:

\texttt{<gtype-class>}

Class Precedence List:

\texttt{<object-class>}

\texttt{<gtype-class>}

\texttt{<class>}

\textsuperscript{18} The Guile Object Oriented System (see Section “GOOPS” in *The GNU Guile Reference Manual*). If you haven’t done so already, please make sure you read both the [Naming Conventions], page 5, and [GOOPS Notes and Conventions], page 7, sections.
The metaclass of all GType classes. Ensures that GType classes have an `info` slot, holding a pointer to either a `GIObjectInfo` or a `GIInterfaceInfo`.

It is an instance of `<class>`.

Superclasses are:

```
<class>
```

Class Precedence List:

```
<gtype-class>
<class>
<object>
<top>
```

Direct slots are:

```
info #:accessor !info
#:init-keyword #:info

derived #:accessor !derived
#:init-keyword #:derived
#:init-value #f

A class is derived when it is user defined (not imported), and inherit a `<gobject>`, page 28, subclass.

```
namespace #:accessor !namespace

g-type #:accessor !g-type

g-name #:accessor !g-name

g-class #:accessor !g-class
```

The `#:info` `#:init-keyword` is mandatory, other slots are initialized automatically. All slots are immutable (to be precise, they are not meant to be mutated, see [GOOPS Notes and Conventions], page 7, 'Slots are not Immutable').

```
!info (inst <gtype-class>) [Accessor]
!derived (inst <gtype-class>) [Accessor]
```
II. Using G-Golf

!namespace (inst <gtype-class>)  [Accessor]
!g-type (inst <gtype-class>)  [Accessor]
!g-name (inst <gtype-class>)  [Accessor]
!g-class (inst <gtype-class>)  [Accessor]

Returns the content of their respective slot for inst.

<gtype-instance>  [Class]
The root class of all instantiable GType classes. Adds a slot, g-inst, to instances, which holds a pointer to the C value.

It is an instance of [gtype-class], page 29.

Superclasses are:
<object>

Class Precedence List:

<gtype-instance>
<object>
<top>

Direct slots are:

$g-inst$  #:accessor !g-inst

The $g-inst$ slot is initialized automatically and immutable (to be precise, it is not meant to be mutated, see [GOOPS Notes and Conventions], page 7, 'Slots are not Immutable').

!g-inst (inst <gtype-instance>)  [Accessor]

Returns the content of the $g-inst$ slot for instance.

unref (inst <gtype-instance>)  [Method]

Returns nothing.

This method calls [g-object-unref], page 59, on the $g-inst$ of instance.

When the reference count for the $g-inst$ reaches 0 (zero), it sets the $g-inst$ slot value for instance to #f and removes instance from the %g-inst-cache.

Note that it used to be mandatory to call this method upon unreachable instances, so that their memory could be freed by the next gc (garbage collector) occurrence, but this is not the case anymore, as auto gc of unreachable <gobject> instances is a now feature [since August 2021].

Procedures

gobject-class? val  [Procedure]

Returns #t if val is a class and if [gobject], page 28, is a member of its class precedence list. Otherwise, it returns #f.
G-Golf Valley

Cache Park
Cache Park - Accessing G-Golf caches.

Procedures

[gi-cache-show], page 31
[gi-cache-ref], page 31

Variables

[%gi-cache], page 32

Description

G-Golf has and uses a cache ‘mechanism’ - actually several, but only one is (partially) exposed to users (and with reserves, see below), also referred to as G-Golf main cache - not only for internal needs, but also to avoid reconstructing things ‘on-the-fly’ unnecessarily, such as already imported [gi-enum], page 116, [gi-flags], page 118, and [gi-struct], page 119, instances.

G-Golf main cache exposed functionality is ‘access only’ - users should not (never) attempt to change its content - and its design is not (yet) ‘set in stone’, so interfaces here exposed, may (have to be) change(d).

So, keeping the above reserves in mind, G-Golf main cache current data structure is composed of two nested association lists, to which we refer using m-key (main key) and s-key (secondary key).

Procedures

gi-cache-show [m-key #f] [Procedure]
Returns nothing.
Displays the content of G-Golf main cache. If m-key (main key) is #f (the default), it displays the list of the main keys present in the cache. Otherwise, it retrieves the content of the main cache for m-key and displays its content if any, or -- is empty -- if none.

gi-cache-ref m-key s-key [Procedure]
Returns a [%gi-cache], page 32, entry or #f.
Obtains and returns the [%gi-cache], page 32, entry for m-key and s-key, or #f if none is found.

Remember that you may (always) view the list of main and secondary key names (which is ‘dynamic’, depending on what you have imported) by calling [gi-cache-show], page 31, (without or with an m-key arg appropriately), but as a user, the two most important m-key are ‘enum’ and ‘flags’, so you may check their member names, or bind their instance locally.
Main key names are given by G-Golf. Secondary key names are always the result of calling [g-name->name], page 123, upon the ‘object’ original name.
II. Using G-Golf

For example, let’s import, then retrieve and visualize the content of the GtkPositionType (enum) type:

```lisp
(use (g-golf))
(gi-import-by-name "Gtk" "PositionType")
⇒ $2 = #"<<gi-enum> 7ff938938b40>

(gi-cache-ref 'enum 'gtk-position-type)
⇒ $3 = #"<<gi-enum> 7ff938938b40>

(describe $3)
⊣ #"<<gi-enum> 7ff938938b40> is an instance of class <gi-enum>
⊣ Slots are:
⊣   enum-set = ((left . 0) (right . 1) (top . 2) (bottom . 3))
⊣   g-type = 94673466933568
⊣   g-name = "GtkPositionType"
⊣   name = gtk-position-type
```

Variables

%gi-cache

Holds a reference the the G-Golf main cache, which as said earlier, currently is composed of two nested association lists.

Customization Square

Customization Square - G-Golf customization functionality.
II. Using G-Golf

Procedures and Syntax

[g-name-transform-exception], page 34
[g-name-transform-exception?], page 34
[g-name-transform-exception-add], page 34
[g-name-transform-exception-remove], page 34
[g-name-transform-exception-reset], page 34
[g-studly-caps-expand-token-exception], page 34
[g-studly-caps-expand-token-exception?], page 35
[g-studly-caps-expand-token-exception-add], page 35
[g-studly-caps-expand-token-exception-remove], page 35
[g-studly-caps-expand-token-exception-reset], page 35
[gi-strip-boolean-result], page 36
[gi-strip-boolean-result?], page 36
[gi-strip-boolean-result-add], page 36
[gi-strip-boolean-result-remove], page 36
[gi-strip-boolean-result-reset], page 36
[gi-method-short-name-skip], page 37
[gi-method-short-name-skip?], page 37
[gi-method-short-name-skip-all], page 37
[gi-method-short-name-skip-add], page 37
[gi-method-short-name-skip-remove], page 37
[gi-method-short-name-skip-reset], page 37
[syntax-name-protect-prefix], page 38
[syntax-name-protect-prefix-set], page 38
[syntax-name-protect-prefix-reset], page 38
[syntax-name-protect-postfix], page 38
[syntax-name-protect-postfix-set], page 38
[syntax-name-protect-postfix-reset], page 38
[syntax-name-protect-renamer], page 38
[syntax-name-protect-renamer-set], page 38
[syntax-name-protect-renamer-reset], page 38
[syntax-name-protect-reset], page 38

Description

Welcome to the G-Golf Customization Square.

This section is organized per customization theme: (-) GI Name Transformation; (-) GI Strip Boolean Result; (-) GI Method Short Name Skip and (-) GI Syntax Name Protect.

GI Name Transformation

In this corner of the square, we expose how you may customize G-Golf with respect to GI Name Transformation that occurs when importing GNOME libraries.

When G-Golf imports a GNOME library, its classes, properties, methods, functions, types and constants are renamed (See [Naming Conventions], page 5), mainly to (a) avoid ‘Camel Case (https://en.wikipedia.org/wiki/Camel_case), (b) surround class names by ‘<’ ‘>’ and (c) replace ‘_’ (underscore) occurrences using the ‘-’ (hyphen) character.
As the context of name transformation is GNOME in general, as opposed to GI more specifically, (all) procedures involved are named using a \texttt{g-} prefix.

Here is a summary of how the name transformation happens:

- Class names are obtained by calling \texttt{[g-name->class-name], page 123}, which calls \texttt{[g-name->name], page 123};

- \texttt{[g-name->name], page 123}, first calls \texttt{[g-name-transform-exception?], page 34}, and returns its value if it found one, otherwise, it calls \texttt{[g-studly-caps-expand], page 123};

- \texttt{[g-studly-caps-expand], page 123}, which does the core of the job, uses \texttt{[g-studly-caps-expand-token-exception?], page 35}, to specially treat its listed token exceptions.

\texttt{g-name-transform-exception}  
\begin{verbatim}
[Procedure]
\end{verbatim}  
\begin{itemize}
\item Returns an alist.
\item Obtains and returns the list of GI name transform exception \((key \ . \ value)\) pairs. Both \texttt{key} and \texttt{value} are strings.
\item The GI name transform exception alist is never empty, as it is initialized and always kept to at least contain the '"GObject" . "gobject") pair\textsuperscript{19}.
\item As a consequence \texttt{[<gobject>], page 28}, (as opposed to \texttt{<g-object>}) is the G-Golf class name for the base class of the GLib Object System.
\item This only affects the class name though - any procedure or method name that comes from the "GObject" namespace is transformed using the \texttt{g-object} prefix, as the upstream library prefix is \texttt{g_object}.
\end{itemize}

\texttt{g-name-transform-exception? key}  
\begin{verbatim}
[Procedure]
\end{verbatim}  
\begin{itemize}
\item Returns \#t if \texttt{key} is a key member of the GI name transform exception alist. Otherwise, it returns \#f.
\end{itemize}

\texttt{g-name-transform-exception-add key value}  
\begin{verbatim}
[Procedure]
\end{verbatim}  
\begin{itemize}
\item Returns nothing.
\item Add (remove) a \((key \ . \ value)\) pair to (from) the GI name transform exception alist.
\end{itemize}

\texttt{g-name-transform-exception-reset}  
\begin{verbatim}
[Procedure]
\end{verbatim}  
\begin{itemize}
\item Returns nothing.
\item This procedure resets the GI name transform exception alist to its default value - which is to contain the single '"GObject" . "gobject") pair.
\end{itemize}

\texttt{g-studly-caps-expand-token-exception}  
\begin{verbatim}
[Procedure]
\end{verbatim}  
\begin{itemize}
\item Returns an alist.
\end{itemize}

\textsuperscript{19} This is the only name for which G-Golf maintains compatibility with Guile-GNOME (which has a long list of exceptions).
II. Using G-Golf

Obtains and returns the list of GI studly caps expand token exception (key . value) pairs. Both key and value are strings.

The GI studly caps expand token exception alist is never empty, as it is initialized and always kept to at least contain the '("WebKit" . "webkit") pair.

**g-studly-caps-expand-token-exception? key**  
Returns #t if key is a key member of the GI studly caps expand token exception alist. Otherwise, it returns #f.

**g-studly-caps-expand-token-exception-add key value**  
**g-studly-caps-expand-token-exception-remove key**  
**g-studly-caps-expand-token-exception-reset**

These procedures allow you to add, remove, or reset the GI studly caps expand token exception alist.

**GI Strip Boolean Result**

In this corner of the square, we expose how you may customize G-Golf with respect to *GI Strip Boolean Result*, which addresses the problem of typelib functions and methods that (1) have at least one ’inout or ’out argument(s) and (2) return either #t or #f, solely to indicate that the function or method call was successful or not.

The default G-Golf behavior, when there is at least one ’inout or ’out argument(s), is to return multiple values. The first returned value is the function or method result, followed by the ’inout and ’out values, in order of appearance in the function or method call.

G-Golf also offers - through a series of interfaces to get, check, add, remove and reset a list of such function or methods names - to instead elude the function or method returned value when it is #t and raise an exception if the returned value is #f.

Here is a concrete example, for the "Clutter" namespace and the clutter-color-from-string procedure:

```lisp
c(use (g-golf))
(gi-import "Clutter")

(clutter-color-from-string "Blue")
⇒ $2 = #t
⇒ $3 = (0 0 255 255)

And call it with an undefined color name:

(clutter-color-from-string "Bluee")
⇒ $4 = #f
⇒ $5 = (0 0 0 0)
```
Now, let’s add `clutter-color-from-string` to the list of GI functions and methods for which we wish to elude the result of the call from the returned value(s), then experiment the above calls and see how G-Golf changed the way it handles the results:

```
(gi-strip-boolean-result-add clutter-color-from-string)
```

```
(clutter-color-from-string "Blue")
⇒ $7 = (0 0 255 255)
```

As expected, if we call it with an undefined color name, it will raise an exception:

```
(clutter-color-from-string "Bluee")
⊣ ice-9/boot-9.scm:1686:16: In procedure raise-exception:
⊣   clutter-color-from-string " failed."
⊣
⊣ Entering a new prompt. Type ‘,bt’ for a backtrace or ‘,q’ to continue.
```

G-Golf default is that the list of GI functions and methods for which to elude the result of the call from the returned value(s) is empty. It is a user responsibility to fill it appropriately, for each namespace they are importing.

### gi-strip-boolean-result

[Procedure]

Returns a (possibly empty) list of (symbol) name(s).

Obtains and returns the list of GI functions and methods for which G-Golf will elude the result of the call from the returned value(s).

### gi-strip-boolean-result? name

[Procedure]

Returns `#t` if name is a member of the list of GI functions and methods for which G-Golf will elude the result of the call from the returned value(s). Otherwise, it returns `#f`.

### gi-strip-boolean-result-add name . . .

[Syntax]

Add (remove) the names to (from) the list of GI functions and methods for which G-Golf will elude the result of the call from the returned value(s).

### gi-strip-boolean-result-reset

[Procedure]

Resets the list of GI functions and methods for which G-Golf will elude the result of the call from the returned value(s) to the empty list.

### GI Method Short Name Skip

In this corner of the square, we expose how you may customize G-Golf with respect to GI Method Short Name, more specifically, whether you wish to skip the method short name creation, and doing so individually or for all GI imported methods.

By default, as it imports a GI typelib, G-Golf creates a method short name for each imported method, obtained by dropping the container name (and its trailing hyphen) from the GI typelib method full/long name.

---

20 Note that the raised exception message and formatting depends on the version of guile you are using. Fwiw, this example was produced using GNU Guile 3.0.8.
II. Using G-Golf

For example, the <gtk-label> class, which defines the gtk-label-get-text method, would also define, using G-Golf’s default settings, the get-text method. To be more precise, G-Golf would create (if it does not exist) or reuse (if it exists) the get-text generic function, make and add a method with its specializer(s), in this case <gtk-label>.

Now, let’s add gtk-label-get-text to the list of the GI methods for which we wish to skip the short name creation step. In this case, as G-Golf imports the GtLabel class, it would only create the gtk-label-get-text method, but not the get-text method anymore.

```lisp
(gi-method-short-name-skip)  [Procedure]
  Returns a (possibly empty) list of (symbol) name(s).
  Obtains and returns the list of GI method long name for which G-Golf will skip the method short name creation step.

(gi-method-short-name-skip? name)  [Procedure]
  Returns #t if name is a member of the list of GI method long name for which G-Golf will skip the method short name creation step. Otherwise, it returns #f.

(gi-method-short-name-skip-all)  [Procedure]
  Returns nothing.
  Sets the GI method short name skip creation step to ’all.

(gi-method-short-name-skip-add name ...)  [Syntax]
  (gi-method-short-name-skip-remove name ...)  [Syntax]
  Add (remove) the names to (from) the list of GI method long name for which G-Golf will skip the method short name creation step.

(gi-method-short-name-skip-reset)  [Procedure]
  Resets the list of GI method long name for which G-Golf will skip the method short name creation step to the empty list.

GI Syntax Name Protect

In this corner of the square, we expose how you may customize G-Golf with respect to GI Syntax Name Protect.

When G-Golf creates a method short name, obtained by dropping the container name (and its trailing hyphen) from the GI typelib method full/long name, it may lead to a ’name clash’, with an already defined procedure or syntax.

GI methods are added to their respective generic function, which is created if it does not already exist. When a generic function is created, G-Golf checks if the name is used, and when it is bound to a procedure, the procedure is ’captured’ into an unspecialized method, which is added to the newly created generic function.

However, when the name is used but its variable value is a syntax, the above can’t be done and the name must be ‘protected’, which is what [syntax-name->method-name], page 124, does21, using a renamer, or by adding a prefix, a postfix or both to its (symbol) name argument.

21 Users should normally not call this procedure - except for testing purposes, if/when they customize its default settings - it is appropriately and automatically called by G-Golf when importing a GI typelib.
G-Golf defines the following interfaces to get, set and reset the syntax name protect prefix, postfix and renamer, of which at least one must be set.

- **syntax-name-protect-prefix** [Procedure]
- **syntax-name-protect-prefix-set** prefix [Procedure]
- **syntax-name-protect-prefix-reset** [Procedure]

  Respectively get, set and reset the syntax name protect prefix. Its default value is #f.

- **syntax-name-protect-postfix** [Procedure]
- **syntax-name-protect-postfix-set** postfix [Procedure]
- **syntax-name-protect-postfix-reset** [Procedure]

  Respectively get, set and reset the syntax name protect postfix. Its default value is ’_ (the symbol _).

- **syntax-name-protect-renamer** [Procedure]
- **syntax-name-protect-renamer-set** renamer [Procedure]
- **syntax-name-protect-renamer-reset** [Procedure]

  Respectively get, set and reset the syntax name protect renamer. Its default value is ’_ (the symbol _).

  The syntax name protect renamer, unless set to #f, must be a procedure that takes a (symbol) name as its single argument, and return a ‘none clashing’ (symbol) name.

- **syntax-name-protect-reset** [Procedure]

  This procedure will conveniently reset all three syntax name protect prefix, postfix and renamer to their default value, which are:

  - [syntax-name-protect-prefix], page 38, #f
  - [syntax-name-protect-postfix], page 38, ’_ (the symbol _)
  - [syntax-name-protect-renamer], page 38, #f

**VFunc Alley**

VFunc Alley - VFunc G-Golf support.

--- Special Notes ---

For completion, this section exposes the definition of the [vfunc], page 40, class and [vfunc], page 41, syntax, involved in the G-Golf integration of the (GLib Object System) VFunc. From a (strict) user point of view however, these are actually G-Golf internals and, unless you are interested of course, might be ignored.

In the GObject documentation, the terminology (mostly) used is virtual public/private method or simply virtual method. In the GI (GObject Introspection) documentation how-
ever, the structure representing a virtual method is named a GI
FuncInfo and the description says it represents a virtual function. The GI core functionality also uses the vfunc or
vfunc-info prefix, infix or postfix terms, depending on the context.

Class

[<vfunc>], page 40

Syntaxes and Accessors

[define-vfunc], page 40
[vfunc], page 41
![vf-class], page 41
![vf-name], page 41
![vf-info], page 41
![vf-long-name-prefix], page 41
![vf-long-name?], page 41

Description

Welcome to the VFunc G-Golf Alley.

Let’s first recap :-) GObject (the GLib Object System) offers different ways to define object and interface methods and extend them, well introduced and described in the GObject Tutorial (https://docs.gtk.org/gobject/tutorial.html):

- non-virtual public methods
- virtual public methods
- virtual private methods
- non-virtual private methods

Of those four, virtual public methods and virtual private methods maybe overridden, through the use of a mechanism that involves the creation of a C closure and the setting of its pointer in the corresponding GObject or Interface class struct.

In G-Golf, this is implemented by the [define-vfunc], page 40, syntax, which must be used to define a VFunc (virtual method). From a user perspective, define-vfunc is very much like define-method (See Section “Methods and Generic Functions” in The GNU Guile Reference Manual).

Here is an example, which defines a GObject subclass that inherits the GdkPaintable interface, then overrides the get_flags VFunc, one of its numerous virtual methods:

(define-class <solitaire-peg> (<gobject> <gdk-paintable>))
    (i #:accessor !i #:init-keyword #:i)
    (j #:accessor !j #:init-keyword #:j))
    (define-vfunc (get-flags-vfunc (self <solitaire-peg>))
      '(size contents))

The only difference, from a user point of view and as you can see in the example above, is that define-vfunc imposes one (or two, depending on the context) additional constraint(s) to the VFunc name, fully described in the [define-vfunc], page 40, definition.
II. Using G-Golf

Class

<\texttt{vfunc}> 

The base class of all virtual method. 

It is an instance of \texttt<class>. 

Superclasses are:

<\texttt{method}>

Class Precedence List:

<\texttt{vfunc}>
<\texttt{method}>
<\texttt{object}>
<\texttt{top}>

Direct slots are:

\begin{itemize}
\item \texttt{vf-class} #:accessor \texttt{!vf-class}
\item \texttt{vf-name} #:accessor \texttt{!vf-name}
\item \texttt{vf-info} #:accessor \texttt{!vf-info}
\item \texttt{vf-long-name-prefix} #:accessor \texttt{!vf-long-name-prefix}
\item \texttt{vf-long-name?} #:accessor \texttt{!vf-long-name?}
\end{itemize}

All direct slots are initialized automatically and immutable (to be precise, they are not meant to be mutated, see [GOOPS Notes and Conventions], page 7, 'Slots are not Immutable').

Syntaxes and Accessors

\texttt{define-vfunc (generic parameter ...)} \texttt{body ...}

Defines a \texttt{vfunc} (a specialized method) for the generic function \texttt{generic} with parameters \texttt{parameters} and body \texttt{body} . . . .

\texttt{generic} is a generic function, and the following constraints apply to the generic function name:

- the \texttt{generic} function name is valid if it is the scheme representation of a VFunc (name) that exists for at least one of the instance specializer superclasses, followed by the \texttt{-vfunc} postfix\footnote{This is because most of the cases, in the upstream lib, the VFunc is a virtual public method, that is, both a method and a VFunc exist that use the same name. When that happens, the upstream lib method normally has the same arity and definition (spec), and it 'just' calls the VFunc - however, it is (unfortunately) not guaranteed to always be the case, hence all GI lang bindings impose a specific VFunc naming convention. Pygobject for example imposes to use a \texttt{do-} prefix. In G-Golf, we opted for a \texttt{-vfunc} postfix.}. 

\footnote{\texttt{-vfunc} postfixed to the function name is used by most GI lang bindings to denote a virtual function.}
• if more then one instance specializer superclasses has a VFunc name, then the 
scheme name must be a so-called long name\textsuperscript{23}, followed by the \texttt{-vfunc} postfix\textsuperscript{24}.

If generic is a variable which is not yet bound to a generic function object, the 
extension of \texttt{define-vfunc} will include a call to \texttt{define-generic}.

Each \texttt{parameter} must be either a symbol or a two-element list (\texttt{symbol class}). The 
symbols refer to variables in the body forms that will be bound to the parameters 
supplied by the caller when calling this method. The \texttt{classes}, if present, specify the 
possible combinations of parameters to which this method can be applied.

\texttt{body . . . } are the bodies of the vfunc definition.

\texttt{vfunc (parameter . . .) body . . .} \hspace{1cm} \textbf{[Syntax]}
\begin{quote}
Makes a vfunc (a specialized method) whose specializers are defined by the classes 
in \texttt{parameters} and whose procedure definition is constructed from the \texttt{parameter} 
symbols and \texttt{body} forms.
\end{quote}

The \texttt{parameter} and \texttt{body} parameters should be as for [\texttt{define-vfunc}], page 40.

\begin{verbatim}
!vf-class inst \hspace{1cm} \texttt{[Accessor]} 
!vf-name inst \hspace{1cm} \texttt{[Accessor]} 
!vf-info inst \hspace{1cm} \texttt{[Accessor]} 
!vf-long-name-prefix inst \hspace{1cm} \texttt{[Accessor]} 
!vf-long-name? inst \hspace{1cm} \texttt{[Accessor]}
\end{verbatim}

Returns the content of their respective slot for \texttt{inst} (a \texttt{<vfunc>} instance).

\section*{III. G-Golf Core Reference}

\subsection*{Overview}

\subsection*{Structure and Naming Conventions}

G-Golf Core Reference modules and documentation structure and naming conventions are 
based, whenever it is possible, on the structure and naming conventions of the corresponding 
GNOME library.

To illustrate, let’s look at a few GLib, GObject and GObject Introspection sections and 
corresponding G-Golf sections and modules naming examples:

\textbf{Glib}

\textit{It must be prefixed using the scheme representation name of the GObject or Interface that owns the Vfunc, 
followed by - (hyphen), i.e. gdk-paintable-get-flags-vfunc is the valid define-vfunc long name for the get_flags 
virtual method of the GdkPaintable interface.}

\textit{Otherwise, it would be impossible to determine which iface or object class struct the *-vfunc user code is 
meant to override. Consider (define-class <foo> (<gobject> <bar> <baz>)), with both <bar> and <baz> defining 
a get_flags VFuc: in this context (define-vfunc (get-flags-vfunc (self <foo>))...) is an invalid definition, as 
it is not possible for G-Golf to determine if it is the <bar> or the <baz> iface class struct VFuc that must 
be overridden. In such cases, the user must pass a method long name, i.e. (define-vfunc (bar-get-flags-vfunc 
(self <foo>)) ...) or (define-vfunc (baz-get-flags-vfunc (self <foo>)) ...).}
Memory Allocation
[Memory Allocation], page 43,
(g-golf glib mem-alloc)

The Main Event Loop
[The Main Event Loop], page 43,
(g-golf glib main-event-loop)

...

GObject

Type Information
(https://developer.gnome.org/gobject/stable/gobject-Type-Information.html)
>Type Information], page 54,
(g-golf gobject type-info)

GObject
[GOobject], page 58,
(g-golf gobject gobject)

Enumeration and Flag Types
[Enumeration and Flag Types], page 60,
(g-golf gobject enum-flags)

...

GObject Introspection

[Repository], page 74,
(g-golf gi repository)

common types
[Common Types], page 77,
(g-golf gi common-types)

[Base Info], page 79,
(g-golf gi base-info)

...

Support to the G-Golf Core Reference modules themselves, or additional functionality to G-Golf as a all, is organized and located in other (none GNOME library based) modules, such as (g-golf support ...), g-golf override ...)}
Glib

G-Golf Glib modules are defined in the `glib` subdirectory, such as `(g-golf glib main-event-loop)`.

Where you may load these modules individually, the easiest way to use G-Golf Glib is to import its main module, which imports and re-exports the public interface of (oop goops), (system foreign), all G-Golf support and G-Golf Glib modules:

```
(use-modules (g-golf glib))
```

G-Golf Glib low level API modules correspond to a Glib section, though they might be some exception in the future.

Memory Allocation

G-Golf Glib Memory Allocation low level API.

Memory Allocation — general memory-handling

Procedures

```
[g-malloc], page 43
[g-malloc0], page 43
[g-free], page 43
[g-memdup], page 43
```

Description

These functions provide support for allocating and freeing memory.


Procedures

```
g-malloc n-bytes
[g-malloc0] n-bytes
Returns a pointer to the allocated memory, or #f.
Allocates `n-bytes` of memory. If `n-bytes` is 0 it returns #f. When using `g-malloc0`, the allocated memory is initialized to 0.

g-free mem
Returns nothing.
Frees the memory pointed to by `mem`.

g-memdup mem n-bytes
Returns a pointer to the allocated memory, or #f.
Allocates `n-bytes` of memory and copies `n-bytes` into it from `mem`. If `mem` is the %null-pointer or `n-bytes` is 0 it returns #f.
```

The Main Event Loop

G-Golf Glib Main Event Loop low level API.

The Main Event Loop — manages all available sources of events
III. G-Golf Core Reference

Procedures

[g-main-loop-new], page 44
[g-main-loop-run], page 45
[g-main-loop-ref], page 44
[g-main-loop-unref], page 45
[g-main-loop-quit], page 45
[g-main-context-new], page 45
[g-main-context-default], page 45
[g-timeout-source-new], page 45
[g-timeout-source-new-seconds], page 45
[g-idle-source-new], page 45
[g-source-ref-count], page 46
[g-source-ref], page 46
[g-source-unref], page 46
[g-source-free], page 46
[g-source-attach], page 46
[g-source-destroy], page 46
[g-source-is-destroyed?], page 46
[g-source-set-priority], page 46
[g-source-get-priority], page 47
[g-source-remove], page 47

Description

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


Procedures

Note: in this section, the loop, context and source arguments are [must be] pointers to a GMainLoop, a GMainContext and a GSource respectively.

[g-main-loop-new] [context #f] [is-running? #f] [Procedure]

Returns a pointer to a new GMainLoop.

Creates a new GMainLoop structure.

The context must be a pointer to a GMainContext of #f, in which case the default context is used. When is-running? is #t, it indicates that the loop is running. This is not very important since calling g-main-loop-run will set this to #t anyway.

[g-main-loop-ref] loop [Procedure]

Returns loop.

Increases the loop reference count by one.
III. G-Golf Core Reference

**g-main-loop-unref loop**
Returns nothing.
Decreases the loop reference count by one. If the result is zero, free the loop and free all associated memory.

**g-main-loop-run loop**
Returns nothing.
Runs a main loop until [g-main-loop-quit], page 45, is called on the loop. If this is called for the thread of the loop’s GMainContext, it will process events from the loop, otherwise it will simply wait.

**g-main-loop-quit loop**
Returns nothing.
Stops a GMainLoop from running. Any calls to [g-main-loop-run], page 45, for the loop will return.
Note that sources that have already been dispatched when g-main-loop-quit is called will still be executed.

**g-main-context-new**
Returns a pointer.
Creates and returns a (pointer to a) new GMainContext structure.

**g-main-context-default**
Returns a pointer.
Returns the global default main context. This is the main context used for main loop functions when a main loop is not explicitly specified, and corresponds to the ‘main’ main loop.

**g-timeout-source-new interval**
Returns a pointer.
Creates and returns (a pointer to) a new (timeout) GSource.
The source will not initially be associated with any GMainContext and must be added to one with [g-source-attach], page 46, before it will be executed.
The timeout interval is in milliseconds.

**g-timeout-source-new-seconds interval**
Returns a pointer.
Creates and returns (a pointer to) a new (timeout) GSource.
The source will not initially be associated with any GMainContext and must be added to one with [g-source-attach], page 46, before it will be executed.
The timeout interval is in seconds.

**g-idle-source-new**
Returns a pointer.
Creates and returns (a pointer to) a new (idle) GSource.
The source will not initially be associated with any `GMainContext` and must be added to one with `[g-source-attach]`, page 46, before it will be executed. Note that the default priority for idle sources is 200, as compared to other sources which have a default priority of 300.

**g-source-ref-count** *source*  
Returns an integer.  
Obtains and returns the reference count of *source*.

**g-source-ref** *source*  
Returns *source*.  
Increases the *source* reference count by one.

**g-source-unref** *source*  
Returns nothing.  
Decreases the *source* reference count by one. If the resulting reference count is zero the source and associated memory will be destroyed.

**g-source-free** *source*  
Returns nothing.  
Calls `[g-source-destroy]`, page 46, and decrements the reference count of *source* to 0 (so *source* will be destroyed and freed).

**g-source-attach** *source* *context*  
Returns an integer.  
Adds *source* to *context* so that it will be executed within that context.  
Returns the ID (greater than 0) for the *source* within the *context*.  
Remove it by calling `[g-source-destroy]`, page 46.

**g-source-destroy** *source*  
Returns nothing.  
Removes *source* from its `GMainContext`, if any, and mark it as destroyed. The source cannot be subsequently added to another context. It is safe to call this on sources which have already been removed from their context.  
This does not unref *source*: if you still hold a reference, use `g-source-unref` to drop it.

**g-source-is-destroyed?** *source*  
Returns #t if *source* has been destroyed. Otherwise, it returns #f.  
Once a source is destroyed it cannot be un-destroyed.

**g-source-set-priority** *source* *priority*  
Returns nothing.  
Sets the *source* priority. 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.
A child source always has the same priority as its parent. It is not permitted to change the priority of a source once it has been added as a child of another source.

**g-source-get-priority source priority**  
Returns an integer.  
Obtains and returns the source priority.

**g-source-remove id**  
Returns #t.  
Removes the source with the given id from the default main context. You must use [g-source-destroy], page 46, for sources added to a non-default main context.  
It is an error to attempt to remove a non-existent source.  
Source IDs can be reissued after a source has been destroyed. This could lead to the removal operation being performed against the wrong source, unless you are cautious.  
For historical reasons, this procedure always returns #t.

**IO Channels**  
G-Golf Glib IO Channels low level API.  
IO Channels — portable support for using files, pipes and sockets

**Procedures**

[g-io-channel-unix-new], page 47  
[g-io-channel-ref], page 48  
[g-io-channel-unref], page 48  
[g-io-create-watch], page 48

**Types and Values**

[%g-io-condition], page 48

**Description**

The GIOMChannel 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.  

**Procedures**

Note: in this section, the **fd**, channel and condition arguments are respectively an integer (a ‘valid’ file descriptor), a pointer to a GIOMChannel and a list of one or more [%g-io-condition], page 48, flags.

**g-io-channel-unix-new fd**  
Returns a pointer.  
Creates and returns a pointer to a new GIOMChannel for fd (file descriptor). On UNIX systems this works for plain files, pipes, and sockets.
The newly created GIOChannel has a reference count of 1.

The default encoding for GIOChannel is UTF-8. If your application is reading output from a command using via pipe, you may need to set the encoding to the encoding of the current locale (FIXME - still missing a binding to g_io_channel_set_encoding).

g-io-channel-ref channel
   Returns channel.
   Increments the channel reference count.

[g-get-prgname], page 49
[g-set-prgname], page 49
[g-get-system-data-dirs], page 49
[g-get-system-config-dirs], page 49
Description
These are portable utility functions.

Procedures

\textbf{g-get-prgname}\quad [Procedure]

Returns the name of the program, or \texttt{##}if it has not been set yet.

Obtains and returns the name of the program. This name should not be localized, in contrast to \texttt{g-get-application-name}.

If you are using \texttt{GApplication}, the program name is set in \texttt{g-application-run}.

\textbf{g-set-prgname name}\quad [Procedure]

Returns nothing.

Sets the name of the program to \texttt{name}. This name should not be localized, in contrast to \texttt{g-set-application-name}.

If you are using \texttt{GApplication}, the program name is set in \texttt{g-application-run}.

Note that for thread-safety reasons this function can only be called once.

\textbf{g-get-system-data-dirs}\quad [Procedure]

Returns an ordered list of base directories in which to access system-wide application data.

On UNIX platforms this is determined using the mechanisms described in the XDG Base Directory Specification (http://www.freedesktop.org/Standards/basedir-spec). In this case the list of directories retrieved will be \texttt{XDG\_DATA\_DIRS}.

On Windows it follows XDG Base Directory Specification if \texttt{XDG\_DATA\_DIRS} is defined. If \texttt{XDG\_DATA\_DIRS} is undefined, the first elements in the list are the Application Data and Documents folders for All Users. (These can be determined only on Windows 2000 or later and are not present in the list on other Windows versions.) See documentation for \texttt{CSIDL\_COMMON\_APPDATA} and \texttt{CSIDL\_COMMON\_DOCUMENTS}.

Then follows the "share" subfolder in the installation folder for the package containing the DLL that calls this function, if it can be determined.

Finally the list contains the "share" subfolder in the installation folder for GLib, and in the installation folder for the package the application’s .exe file belongs to.

The installation folders above are determined by looking up the folder where the module (DLL or EXE) in question is located. If the folder’s name is "bin", its parent is used, otherwise the folder itself.

Note that on Windows the returned list can vary depending on where this function is called.

\textbf{g-get-system-config-dirs}\quad [Procedure]

Returns an ordered list of base directories in which to access system-wide configuration information.
On UNIX platforms this is determined using the mechanisms described in the XDG Base Directory Specification (http://www.freedesktop.org/Standards/basedir-spec). In this case the list of directories retrieved will be XDG_CONFIG_DIRS.

On Windows it follows XDG Base Directory Specification if XDG_CONFIG_DIRS is defined. If XDG_CONFIG_DIRS is undefined, the directory that contains application data for all users is used instead. A typical path is C:\Documents and Settings\All Users\Application Data. This folder is used for application data that is not user specific. For example, an application can store a spell-check dictionary, a database of clip art, or a log file in the CSIDL_COMMON_APPDATA folder. This information will not roam and is available to anyone using the computer.

**UNIX-specific utilities and integration**

G-Golf Glib UNIX-specific utilities and integration low level API. UNIX-specific utilities and integration — pipes, signal handling.

**Procedures**

[g-unix-fd-source-new], page 50

**Description**

Most of GLib is intended to be portable; in contrast, this set of functions is designed for programs which explicitly target UNIX, or are using it to build higher level abstractions which would be conditionally compiled if the platform matches G_OS_UNIX.

**Procedures**

Note: in this section, the *fd* and *condition* arguments are [must be] respectively an integer (a ‘valid’ file descriptor) and a list of one or more [%g-io-condition], page 48, flags.

\[
g\text{-unix-fd-source-new} \quad \text{*fd} \quad \text{*condition} \\
\text{Returns a pointer.} \\
\text{Creates and returns a pointer to a new GSource to watch for a particular IO condition on *fd.} \\
\text{The source will never close the file descriptor, you must do it yourself.}
\]

**Doubly-Linked Lists**

G-Golf Glib Doubly-Linked Lists low level API. Doubly-Linked Lists — linked lists that can be iterated over in both directions

**Procedures**

[g-list-data], page 51
[g-list-next], page 51
[g-list-prev], page 51
[g-list-free], page 51
[g-list-length], page 51
[g-list-nth-data], page 51
Description
The **GList** structure and its associated functions provide a standard doubly-linked list data structure.

Each element in the list contains a piece of data, together with pointers which link to the previous and next elements in the list. Using these pointers it is possible to move through the list in both directions (unlike the singly-linked **GSList**, which only allows movement through the list in the forward direction).


Procedures

**g-list-data**  
`g-list`  
Returns a pointer.  
Obtains and returns a pointer to the data in `g-list`, or any integer value, in which case, it is the responsibility of the caller to apply the appropriate type conversion procedure.

**g-list-next**  
`g-list`  
Returns a pointer or #f.  
Obtains and returns the next element in `g-list`, or #f if there are no more elements.

**g-list-prev**  
`g-list`  
Returns a pointer or #f.  
Obtains and returns the previous element in `g-list`, or #f if there are no previous element.

**g-list-free**  
`g-list`  
Returns nothing.  
Frees all of the memory used by `g-list`.

**g-list-length**  
`g-list`  
Returns an integer.  
Obtains and returns the number of elements in `g-list`. This function iterates over the whole list to count its elements.

**g-list-nth-data**  
`g-list`  
Returns a pointer or #f.  
Obtains and returns a pointer to the data of the `n`-th element of `g-list`. This iterates over the list until it reaches the n-th position. If `n` is off the end of `g-list`, it returns #f.

Singly-Linked Lists

G-Golf Glib Singly-Linked Lists low level API.  
Singly-Linked Lists — Linked lists that can be iterated over in one direction
Procedures

[g-slist-data], page 52
[g-slist-next], page 52
[g-slist-append], page 52
[g-slist-prepend], page 52
[g-slist-free], page 53
[g-slist-length], page 53
[g-slist-nth-data], page 53

Description

The GSList structure and its associated functions provide a standard singly-linked list data structure.

Each element in the list contains a piece of data, together with a pointer which links to the next element in the list. Using this pointer it is possible to move through the list in one direction only (unlike the [Doubly-Linked Lists], page 50, which allow movement in both directions).


Procedures

g-slist-data g-slist

Returns a pointer.

Obtains and returns a pointer to the data in g-slist, or any integer value, in which case, it is the responsibility of the caller to apply the appropriate type conversion procedure.

g-slist-next g-slist

Returns a pointer or #f.

Obtains and returns the next element in g-slist, or #f if there are no more elements.

g-slist-append g-slist data

Returns a pointer.

Adds data - which is (must be) a pointer - to the end of g-slist and returns a pointer to the (possibly new) start of the list (so make sure you store the new value).

Note that [g-slist-append], page 52, has to traverse the entire list to find the end, which is inefficient when adding multiple elements. A common idiom to avoid the inefficiency is to prepend the elements and reverse the list when all elements have been added.

g-slist-prepend g-slist data

Returns a pointer.

Adds data - which is (must be) a pointer - to the start of g-slist and returns a pointer to the (possibly new) start of the list (so make sure you store the new value).
g-slist-free g-slist
   [Procedure]
   Returns nothing.
   Frees all of the memory used by g-slist.

g-slist-length g-slist
   [Procedure]
   Returns an integer.
   Obtains and returns the number of elements in g-slist. This function iterates over the
   whole list to count its elements.

g-slist-nth-data g-slist n
   [Procedure]
   Returns a pointer or #f.
   Obtains and returns a pointer to the data of the n-th element of g-slist. This iterates
   over the list until it reaches the n-th position. If n is off the end of g-slist, it returns
   #f.

Quarks
G-Golf Glib Quarks low level API.
Quarks — a 2-way association between a string and a unique integer identifier.

Procedures

   [g-quark-from-string], page 53
   [g-quark-to-string], page 53

Description
Quarks are associations between strings and integer identifiers. Given either the string or
the GQuark identifier it is possible to retrieve the other.

Procedures

g-quark-from-string str
   [Procedure]
   Returns an integer.
   Obtains and returns the GQuark identifying the string given by str. If the string does
   not currently have an associated GQuark, a new GQuark is created, using a copy of
   the string.

g-quark-to-string g-quark
   [Procedure]
   Returns a string.
   Obtains and returns the string associated with the GQuark given by g-quark.

GObject
G-Golf GObject modules are defined in the gobject subdirectory, such as (g-golf gobject
enum-flags).
Where you may load these modules individually, the easiest way to use G-Golf is to import
its main module, which imports and re-exports the public interface of (oop goops), (system
foreign), all G-Golf support and G-Golf GObject modules:
   (use-modules (g-golf gobject))
G-Golf GObject low level API modules correspond to a GObject section, though they might be some exception in the future.

**Type Information**

G-Golf GObject Type Information low level API.
Type Information — The GLib Runtime type identification and management system

**Procedures**

- [g-type->symbol], page 54
- [symbol->g-type], page 55
- [g-type-name], page 55
- [g-type-from-name], page 55
- [g-type-parent], page 55
- [g-type-is-a], page 55
- [g-type-class-ref], page 55
- [g-type-class-peek], page 55
- [g-type-class-unref], page 55
- [g-type-interface-peek], page 56
- [g-type-query], page 56
- [g-type-register-static-simple], page 56
- [g-type-add-interface-static], page 56
- [g-type-fundamental], page 56
- [g-type-ensure], page 56

**Types and Values**

- [%g-type-fundamental-flags], page 56
- [%g-type-fundamental-types], page 57

**Object Hierarchy**

```
gpointer +— GType
```

**Description**

The GType API is the foundation of the GObject system. It provides the facilities for registering and managing all fundamental data types, user-defined object and interface types.

Please read the Type Information (https://developer.gnome.org/gobject/stable/gobject-Type-Information.html) section from the GObject reference manual for a complete description.

**Procedures**

*Procedure*

**g-type->symbol**

*Procedure*

<table>
<thead>
<tr>
<th>Returns a symbol.</th>
</tr>
</thead>
</table>

Get the symbol that correspond to the type ID `g-type`. Note that this function (like all other GType API) cannot cope with invalid type IDs. It accepts validly registered
type ID, but randomized type IDs should not be passed in and will most likely lead to a crash.

symbol->g-type symbol
Returns a type ID.
Get the type ID for symbol. Note that this function (like all other GType API) cannot cope with invalid type ID symbols. It accepts validly registered type ID symbol, but randomized type IDs should not be passed in and will most likely lead to a crash.

g-type-name g-type
Returns a string.
Get the unique name that is assigned to g-type, a type ID. Note that this function (like all other GType API) cannot cope with invalid type IDs. It accepts validly registered type ID, but randomized type IDs should not be passed in and will most likely lead to a crash.

g-type-from-name name
Returns a type ID or #f.
Obtains and returns the type ID for the given type name, or #f if no type has been registered under this name (this is the preferred method to find out by name whether a specific type has been registered yet).

g-type-parent g-type
Returns a GType.
Returns the direct parent type for g-type. If g-type has no parent, i.e. is a fundamental type, 0 is returned.

g-type-is-a g-type is-a-g-type
Returns #t if g-type is a is-a-g-type.
If is-a-g-type is a derivable type, check whether g-type is a descendant of is-a-g-type.
If is-a-g-type is an interface, check whether g-type conforms to it.

g-type-class-ref g-type
Returns a pointer.
Obtains and returns a pointer to the GTypeClass structure for g-type (a GObject class GType). The reference count of the class is incremented, and the class is 'created' (instanciated) if/when it doesn’t exist already.

g-type-class-peek g-type
Returns a pointer.
Obtains and returns a pointer to the GTypeClass structure for g-type (a GObject class GType). The reference count of the class isn’t incremented. As a consequence, this function may return #f - if the class of the type passed in does not currently exist (hasn’t been referenced before).

g-type-class-unref g-class
Returns nothing.
Decrrents the reference count for \textit{g-class} (a pointer to a \texttt{GTypeClass} structure). Once the last reference count of a class has been released, it may be finalized by the type system. Attempting to further dereference a finalized class is invalid.

\textbf{g-type-interface-peek} \texttt{g-class iface-type} \hspace{1em} [Procedure]

Returns a pointer of \texttt{#f}.

Obtains and returns the (a pointer to) \texttt{GTypeInterface} structure for \texttt{iface-type} if implemented by \texttt{g-class}, Otherwise, it returns \texttt{#f}.

\textbf{g-type-query} \texttt{g-type} \hspace{1em} [Procedure]

Returns a list.

Obtains and returns the (\texttt{g-type type-name class-size instance-size}) list for \texttt{g-type}.

\textbf{g-type-register-static-simple} \texttt{parent-type type-name class-size} \hspace{1em} [Procedure]

\hspace{1em} class-init-func instance-size instance-init-func flags

Returns a newlyer type ID.

Registers \texttt{type-name} as the name of a new static type derived from \texttt{parent-type}. The value of \texttt{flags} determines the nature (e.g. abstract or not) of the type. It works by filling a \texttt{GTypeInfo} struct and calling \texttt{g_type_register_static}.

\textbf{g-type-add-interface-static} \texttt{g-type iface-type iface-info} \hspace{1em} [Procedure]

Returns nothing.

Adds \texttt{iface-type} to the static \texttt{g-type}. The information contained in the \texttt{GInterfaceInfo} structure pointed to by \texttt{iface-info} is used to manage the relationship.

If \texttt{iface-info} is \texttt{#f}, a new \texttt{GInterfaceInfo} structure is made, with \texttt{iface-init-func} and \texttt{iface-finalize-func} set to no-op procedures, and \texttt{iface-data} set to the \texttt{%null-pointer} (this is only meant to be used for testing and debugging purposes).

\textbf{g-type-fundamental} \texttt{g-type} \hspace{1em} [Procedure]

Returns a type ID.

Extracts the fundamental type ID portion for \texttt{g-type}.

\textbf{g-type-ensure} \texttt{g-type} \hspace{1em} [Procedure]

Returns nothing.

Ensures that the indicated \texttt{g-type} has been registered with the type system, and that its _class_init method has been run.

**Types and Values**

\%\texttt{g-type-fundamental-flags} \hspace{1em} [Instance Variable of \texttt{<gi-enum>}]  

Bit masks used to check or determine specific characteristics of a fundamental type.

An instance of \texttt{<gi-enum>}, who’s members are the scheme representation of the \texttt{GTypeFundamentalFlags}:
III. G-Golf Core Reference

\textit{g-name}: GTypeFundamentalFlags
\textit{name}: g-type-fundamental-flags
\textit{enum-set}:

\begin{itemize}
  \item \texttt{classed} Indicates a classed type
  \item \texttt{instantiable} Indicates an instantiable type (implies classed)
  \item \texttt{derivable} Indicates a flat derivable type
  \item \texttt{deep-derivable} Indicates a deep derivable type (implies derivable)
\end{itemize}

\texttt{%g-type-fundamental-types} [Instance Variable of \texttt{<gi-enum>}]  
An instance of \texttt{<gi-enum>}, who’s members are the scheme representation of the GType obtained from the fundamental types defined using \texttt{G_TYPE_MAKE_FUNDAMENTAL}, which starts with \texttt{G_TYPE_INVALID} and ends with \texttt{G_TYPE_OBJECT}.

\textit{g-name}: \#f\footnote{There is no corresponding \texttt{enum} in GObject. These fundamental types (in GObject) are defined using a macro, \texttt{G_TYPE_MAKE_FUNDAMENTAL}, that applies bitwise arithmetic shift given by \texttt{G_TYPE_FUNDAMENTAL_SHIFT} (which we also have to apply, to get to the type ID for the fundamental number \texttt{x}).}

\textit{name}: g-type-fundamental-types
\textit{enum-set}:

\begin{itemize}
  \item \texttt{invalid} An invalid GType used as error return value in some functions which return a GType.
  \item \texttt{none} A fundamental type which is used as a replacement for the C void return type.
  \item \texttt{interface} The fundamental type from which all interfaces are derived.
  \item \texttt{char} The fundamental type corresponding to gchar. It is unconditionally an 8-bit signed integer. This may or may not be the same type a the C type "gchar".
  \item \texttt{uchar} The fundamental type corresponding to guchar.
  \item \texttt{boolean} The fundamental type corresponding to gboolean.
  \item \texttt{int} The fundamental type corresponding to gint.
  \item \texttt{uint} The fundamental type corresponding to guint.
  \item \texttt{long} The fundamental type corresponding to glong.
  \item \texttt{ulong} The fundamental type corresponding to gulong.
  \item \texttt{int64} The fundamental type corresponding to gint64.
  \item \texttt{uint64} The fundamental type corresponding to guint64.
\end{itemize}
### enum
The fundamental type from which all enumeration types are derived.

### flags
The fundamental type from which all flags types are derived.

### float
The fundamental type corresponding to gfloat.

### double
The fundamental type corresponding to gdouble.

### string
The fundamental type corresponding to nul-terminated C strings.

### pointer
The fundamental type corresponding to gpointer.

### boxed
The fundamental type from which all boxed types are derived.

### param
The fundamental type from which all [GParamSpec], page 66, types are derived.

### object
The fundamental type for [GObject*], page 58.

**GObject**

G-Golf GObject low level API.

**GObject** — The base object type

**Procedures**

- [g-object-class-find-property], page 59
- [g-object-new], page 59
- [g-object-new-with-properties], page 59
- [g-object-ref], page 59
- [g-object-unref], page 59
- [g-object-ref-sink], page 59
- [g-object-ref-count], page 60
- [g-object-is-floating], page 60
- [g-object-type], page 60
- [g-object-type-name], page 60
- [g-object-get-property], page 60
- [g-object-set-property], page 60

**Object Hierarchy**

```
GObject
  +— GBinding
  +— GInitiallyUnowned
  +— GTypeModule
```

**Description**

**GObject** is the fundamental type providing the common attributes and methods for all object types in GTK+, Pango and other libraries based on GObject. The GObject class provides methods for object construction and destruction, property access methods, and signal support.

Procedures

Note: in this section, unless otherwise specified, the object argument is [must be] a pointer to a GObject (instance).

**g-object-class-find-property**

*Procedure*

\[\text{Procedure}\]

\[\text{g-object-class-find-property \hspace{1em} g-class name}\]

Returns a pointer or \#f.

Obtains and returns (a pointer to) the GParamSpec for name, or \#f if g-class (a pointer to a GObjectClass) doesn’t have a property of that name.

**g-object-new**

*Procedure*

\[\text{Procedure}\]

\[\text{g-object-new \hspace{1em} gtype}\]

Returns a pointer.

Creates and returns a (pointer to) a new instance of a GObject subtype gtype. All properties are set to their default values.

**g-object-new-with-properties**

*Procedure*

\[\text{Procedure}\]

\[\text{g-object-new-with-properties \hspace{1em} gtype n-prop names g-values}\]

Returns a pointer.

Creates and returns a (pointer to) a new instance of a GObject subtype gtype. The other arguments are n-prop the number of properties, names a pointer to an array of pointers to strings with the names of each property to be set and values an array of GValue containing the values of each property to be set.

Properties that are not explicitly specified are set to their default values.

**g-object-ref**

*Procedure*

\[\text{Procedure}\]

\[\text{g-object-ref \hspace{1em} object}\]

Returns a pointer.

Increases the reference count of object.

**g-object_unref**

*Procedure*

\[\text{Procedure}\]

\[\text{g-object_unref \hspace{1em} object}\]

Returns nothing.

Decreases the reference count of object. When its reference count drops to 0, the object is finalized (i.e. its memory is freed).

If the pointer to the GObject may be reused in future (for example, if it is an instance variable of another object), it is recommended to clear the pointer to NULL rather than retain a dangling pointer to a potentially invalid GObject instance. Use g-clear-object for this.

**g-object-ref-sink**

*Procedure*

\[\text{Procedure}\]

\[\text{g-object-ref-sink \hspace{1em} object}\]

Returns a pointer.

If object has a floating reference, then this call ‘assumes ownership’ of the floating reference, converting it to a normal reference by clearing the floating flag while leaving the reference count unchanged.

If object is not floating, then this call adds a new normal reference increasing the reference count by one.
**III. G-Golf Core Reference**

**g-object-ref-count object**

Returns an integer.

Obtains and returns the (public GObject struct field) `ref_count` value for `object`.

**g-object-is-floating object**

Returns `#t` if `object` has a floating reference, otherwise it returns `#f`.

**g-object-type object**

Returns the `GType` (the type id) for `object`.

**g-object-type-name object**

Returns the `GType` name for `object`.

**g-object-get-property object property [g-type #f]**

Returns the `property` value for `object`.

The `property` argument is (must be) a pointer to a valid `GIPROPERTYINFO` (`property` must point to one of the properties infos of the class of `object`). The `g-type` argument must be a valid `GType` value. If `#f`, which is the default, [gi-property-g-type], page 107, is called.

**g-object-set-property object property value [g-type #f]**

Returns `value`.

Sets the `object property` to `value`. The `property` argument is (must be) a pointer to a valid `GIPROPERTYINFO` (`property` must point to one of the properties infos of the class of `object`). The `g-type` argument must be a valid `GType` value. If `#f`, which is the default, [gi-property-g-type], page 107, is called.

**Enumeration and Flag Types**

G-Golf GObject Enumeration and Flag Types low level API.

**Description**

The GLib type system provides fundamental types for enumeration and flags types. (Flags types are like enumerations, but allow their values to be combined by bitwise or). A registered enumeration or flags type associates a name and a nickname with each allowed value. When an enumeration or flags type is registered with the GLib type system, it can be used as value type for object properties.

**Boxed Types**

G-Golf GObject Boxed Types low level API.

**Procedures**

[g-boxed-free], page 61
[g-strv-get-type], page 61
Description

GBoxed is a generic wrapper mechanism for arbitrary C structures. The only thing the type system needs to know about the structures is how to copy them (a GBoxedCopyFunc) and how to free them (a GBoxedFreeFunc) — beyond that they are treated as opaque chunks of memory.


Procedures

**g-boxed-free**  
* g-type pointer  
[Procedure]  

Returns nothing.  
Frees the boxed structure at *pointer*, which is of type *g-type*.

**g-strv-get-type**  
[Procedure]  

Returns a GType.  
Registers (unless already registered) the GStrv GLib type in GObject and returns its GType, the GType for a boxed type holding a NULL-terminated array of strings. This procedure must have been called at least once before (*g-type-from-name* "GStrv") calls may be honoured.

Generic Values

G-Golf GObject Generic Values low level API.  
Generic values — A polymorphic type that can hold values of any other type.

Procedures

[g-value-size], page 62  
[g-value-new], page 62  
[g-value-init], page 62  
[g-value-unset], page 62

Object Hierarchy

GBoxed  
+— GValue

Description

The GValue structure is basically a variable container that consists of a type identifier and a specific value of that type. The type identifier within a GValue structure always determines the type of the associated value. To create a undefined GValue structure, simply call [g-value-new], page 62, which create a zero-filled GValue structure. To create and initialize a GValue, use the [g-value-init], page 62, procedure. A GValue cannot be used until it is initialized. The basic type operations (such as freeing and copying) are determined by the GTypeValueTable associated with the type ID stored in the GValue.

Procedures

**g-value-size**
Returns an integer.

Obtains and returns the size of a GValue.

**g-value-new**
Returns a pointer to a GValue.

Creates and returns (a pointer to) an empty (uninitialized) GValue.

**g-value-init**
Returns a pointer to a GValue.

Creates and initializes a GValue with the default value for g-type, which can either be an integer - a GType static or dynamic value, or a symbol - a member of the [%g-type-fundamental-types], page 57.

**g-value-unset**
Returns nothing.

Clears the current value in g-value (if any) and ‘unsets’ the type. This releases all resources associated with g-value. An unset GValue is the same as an uninitialized (zero-filled) GValue structure.

Parameters and Values
G-Golf GObject Parameters and Values low level API.
Parameters and Values — Standard Parameter and Value Types
Procedures

[g-value-type], page 63
[g-value-type-tag], page 63
[g-value-type-name], page 63
[g-value-ref], page 64
[g-value-set!], page 64
[g-value-get-boolean], page 64
[g-value-set-boolean], page 64
[g-value-get-int], page 64
[g-value-set-int], page 64
[g-value-get-uint], page 64
[g-value-set-uint], page 64
[g-value-get-float], page 64
[g-value-set-float], page 65
[g-value-get-double], page 65
[g-value-set-double], page 65
[g-value-get-enum], page 65
[g-value-set-enum], page 65
[g-value-get-flags], page 65
[g-value-set-flags], page 65
[g-value-get-string], page 65
[g-value-set-string], page 65
[g-value-get-param], page 65
[g-value-set-param], page 66
[g-value-get-boxed], page 66
[g-value-set-boxed], page 66
[g-value-get-pointer], page 66
[g-value-set-pointer], page 66
[g-value-get-object], page 66
[g-value-set-object], page 66
[g-value-get-variant], page 66

Description

GValue provides an abstract container structure which can be copied, transformed and compared while holding a value of any (derived) type, which is registered as a GType with a GTypeValueTable in its GTypeInfo structure. Parameter specifications for most value types can be created as GParamSpec derived instances, to implement e.g. GObject properties which operate on GValue containers.

Parameter names need to start with a letter (a-z or A-Z). Subsequent characters can be letters, numbers or a '-' character. All other characters are replaced by a '-' during construction.

Procedures and Methods

Note: in this section, the g-value argument is [must be] a pointer to a GValue.

g-value-type g-value [Procedure]
g-value-type-tag g-value [Procedure]
**g-value-type-name** *g-value*  
- Returns an integer, a symbol or a string, respectively.  
- Obtains and returns the **GType**, the **GType** tag (see [%g-type-fundamental-types], page 57) or the **GType** name (see [g-type-name], page 55, for *g-value*, respectively.

**g-value-ref** *g-value*  
- Returns the content of *g-value*.  
- Obtains and returns the content of *g-value*. Supported **GType** (their scheme representation) for *g-value* are: boolean, uint, int, float, double, enum, flags, string, boxed, pointer, object, interface.

**g-value-set!** *g-value* *value*  
- Returns nothing.  
- Sets the content of *g-value* to *value*. Supported **GType** (their scheme representation) for *g-value* are: boolean, uint, int, float, double, enum, flags, string, boxed, pointer, object, interface.  
- Note that this procedure cannot cope with invalid values (the type of *value* must correspond to the **GType** for *g-value*, otherwise it will most likely lead to a crash.

**g-value-get-boolean** *g-value*  
- Returns #t or #f.  
- Obtains the content of *g-value* and returns #f if it is 0, otherwise it returns #t.

**g-value-set-boolean** *g-value* *val*  
- Returns nothing.  
- Sets the content of *g-value* to 0 if *val* is #f, otherwise sets the content to 1.

**g-value-get-int** *g-value*  
- Returns an integer.  
- Obtains and returns the content of *g-value*.

**g-value-set-int** *g-value* *int*  
- Returns nothing.  
- Sets the content of *g-value* to *int*.

**g-value-get-uint** *g-value*  
- Returns an unsigned integer.  
- Obtains and returns the content of *g-value*.

**g-value-set-uint** *g-value* *uint*  
- Returns nothing.  
- Sets the content of *g-value* to *uint*.

**g-value-get-float** *g-value*  
- Returns a float.  
- Obtains and returns the content of *g-value*. 
III. G-Golf Core Reference

**g-value-set-float**  
`g-value float`  
Returns nothing.  
Sets the content of `g-value` to `float`.

**g-value-get-double**  
`g-value`  
Returns a double.  
Obtains and returns the content of `g-value`.

**g-value-set-double**  
`g-value double`  
Returns nothing.  
Sets the content of `g-value` to `double`.

**g-value-get-enum**  
`g-value`  
Returns a symbol.  
Obtains and returns the (registered) enum type info symbol for `g-value`.

**g-value-set-enum**  
`g-value (id <integer>)`  
`g-value-set-enum`  
`g-value (sym <symbol>)`  
Returns nothing.  
Sets the content of `g-value` to `id`, or to the id corresponding to `sym` respectively. The `id` or the `sym` must be valid (as in being a valid member of the (registered) enum type info for `g-value`), otherwise an exception is raised.

**g-value-get-flags**  
`g-value`  
Returns a list.  
Obtains and returns the (registered) list of flags for `g-value`.

**g-value-set-flags**  
`g-value (val <integer>)`  
`g-value-set-flags`  
`g-value (flags <list>)`  
Returns nothing.  
Sets the content of `g-value` to `val`, or to the value given by calling `[flags->integer]`, page 118, upon the list of `flags`, respectively. The `val` or the `flags` must be valid (as in being a valid member of the (registered) gi-flags type for `g-value`), otherwise an exception is raised.

**g-value-get-string**  
`g-value`  
Returns a string or `#f`.  
Obtains and returns the content of `g-value`, a string or `#f` if the `g-value` content is the `%null-pointer`.

**g-value-set-string**  
`g-value str`  
Returns nothing.  
Sets the content of `g-value` to `str`.

**g-value-get-param**  
`g-value`  
Returns a (pointer to) `GParamSpec` or `#f`.  

[Procedure]  
[Procedure]  
[Procedure]  
[Procedure]  
[Procedure]  
[Procedure]  
[Method]  
[Method]  
[Method]  
[Procedure]  
[Procedure]  
[Procedure]
Obtains and returns the content of g-value, a (pointer to) GParamSpec or #f if the g-value content is the %null-pointer.

**g-value-set-param** g-value param

Returns nothing.

Sets the content of g-value to param.

**g-value-get-boxed** g-value

Returns either a list of values, or a pointer.

Obtains and returns the content of g-value. If the boxed type ![is-opaque?], page 120, or ![is-semi-opaque?], page 120, it ‘blindingly’ returns the boxed instance g-value pointer. Otherwise, the boxed instance is ‘decoded’, and a list of its field values is returned.

**g-value-set-boxed** g-value boxed

Returns nothing.

Sets the content of g-value to boxed. If the boxed type ![is-opaque?], page 120, or ![is-semi-opaque?], page 120, then boxed is (supposed to be) a pointer, used to ‘blindingly’ set g-value. Otherwise, the boxed instance is (supposed to be) a list of values, that are ‘encoded’, and its (newly created) pointer is used to set g-value.

**g-value-get-pointer** g-value

Returns a pointer.

Obtains and returns the content of g-value.

**g-value-set-pointer** g-value pointer

Returns nothing.

Sets the content of g-value to pointer.

**g-value-get-object** g-value

Returns a pointer.

Obtains and returns the content of g-value.

**g-value-set-object** g-value object

Returns nothing.

Sets the content of g-value to object (a pointer to a GObject instance) and increases the object reference count.

**g-value-get-variant** g-value

Returns a pointer or #f.

Obtains and returns content of a variant g-value, or #f (may be NULL).

**GParamSpec**

G-Golf GObject GParamSpec low level API.

GParamSpec — Metadata for parameter specifications.
III. G-Golf Core Reference

Procedures

[gi-g-param-spec-show], page 67
[g-param-spec-type], page 68
[g-param-spec-type-name], page 68
[g-param-spec-get-default-value], page 68
[g-param-spec-get-name], page 68
[g-param-spec-get-nick], page 68
[g-param-spec-get-blurb], page 68
[g-param-spec-get-flags], page 68

Types and Values

[%g-param-flags], page 68

Description

GParamSpec is an object structure that encapsulates the metadata required to specify parameters, such as e.g. GObject properties.

Procedures

Note: in this section, the p-spec argument is [must be] a pointer to a GParamSpec.

\textit{gi-g-param-spec-show} \textit{p-spec} \hspace{1cm} [Procedure]

Returns nothing.

Obtains and displays the following informations about the interface pointed to by \textit{p-spec}:

\begin{verbatim}
,use (g-golf)
(g-irepository-require "Gtk" #:version "4.0")
⇒ $2 = #<pointer 0x55ae43d74a60>

(gi-import-by-name "Gtk" "Label")
⇒ $3 = #<<gobject-class> <gtk-label> 7f1a75436a50>

(!g-class <gtk-label>)
⇒ $4 = #<pointer 0x55ae43deb0c0>

(g-object-class-find-property $4 "css-classes")
⇒ $5 = #<pointer 0x55ae43d9d510>

(gi-g-param-spec-show $5)
-
⇒ #<pointer 0x55ae43d9d510> is a (pointer to a) GParamSpec:
-
⇒ name: "css-classes"
⇒ nick: "CSS Style Classes"
⇒ blurb: "List of CSS classes"
⇒ g-type: 94206951022032
⇒ g-type-name: "GStrv"
\end{verbatim}
Note that the last item, type-name: g-strv is not part of the GParamSpec structure. It is obtained (and used by G-Golf internally by calling ([g-name->name], page 123, g-type-name)).

- g-param-spec-type p-spec
  Returns an integer or a (symbol) name, respectively.
  Obtains and returns the GType or the GType (symbol) name for p-spec, respectively.

- g-param-spec-type-name p-spec
  Returns a pointer.
  Obtains and returns the p-spec default value as pointer to a GValue, which will remain valid for the life of p-spec and must not be modified.

- g-param-spec-get-name p-spec
- g-param-spec-get-nick p-spec
- g-param-spec-get-blurb p-spec
  Returns a string.
  Obtains and returns the name, nickname or short description for p-spec, respectively.

- g-param-spec-get-flags p-spec
  Returns a (possibly empty) list.
  Obtains and returns a list of the combination of [%g-param-flags], page 68, that applies to p-spec.

Types and Values

%g-param-flags
An instance of <gi-enum>, who’s members are the scheme representation of the GParamFlags:

type-name: GParamFlags
name: g-param-flags
enum-set:

readable  the parameter is readable
writable  the parameter is writable
readwrite alas for readable writable
construct  the parameter will be set upon object construction
construct-only the parameter can only be set upon object construction
lax-validation
upon parameter conversion, strict validation is not required

static-name
the string used as name when constructing the parameter is
guaranteed to remain valid and unmodified for the lifetime
of the parameter. Since 2.8

private internal

static-nick
the string used as nick when constructing the parameter is
guaranteed to remain valid and unmodified for the lifetime
of the parameter. Since 2.8

static-blurb
the string used as blurb when constructing the parameter is
guaranteed to remain valid and unmodified for the lifetime
of the parameter. Since 2.8

explicit-notify
calls to g_object_set_property for this property will not
automatically result in a ‘notify’ signal being emitted: the
implementation must call g_object_notify themselves in
case the property actually changes. Since: 2.42

deprecated
the parameter is deprecated and will be removed in a future
version. A warning will be generated if it is used while run-
ning with G_ENABLE_DIAGNOSTIC=1. Since 2.26

Closures
G-Golf GObject Closures low level API.

Closures - Functions as first-class objects

Procedures
[g-closure-size], page 70
[g-closure-ref-count], page 70
[g-closure-ref], page 70
[g-closure-sink], page 70
[g-closure-unref], page 70
[g-closure-free], page 71
[g-closure-invoke], page 71
[g-closure-add-invalidate-notifier], page 71
[g-closure-new-simple], page 71
[g-closure-set-marshall], page 71
[g-source-set-closure], page 71
Object Hierarchy

GBoxed
   +— GClosure

Description

A GClosure represents a callback supplied by the programmer. It will generally comprise a function of some kind and a marshaller used to call it. It is the responsibility of the marshaller to convert the arguments for the invocation from GValues into a suitable form, perform the callback on the converted arguments, and transform the return value back into a GValue.


Procedures

Note: in this section, the closure, marshal, source and function arguments are [must be] pointers to a GClosure, a GSource, a GClosureMarshal and a GClosureNotify respectively.

\texttt{g-closure-size} \hspace{1cm} \text{[Procedure]}

Returns an integer.

Obtains and returns the size (the number of bytes) that a GClosure occupies in memory.

\texttt{g-closure-ref-count \hspace{0.5cm} closure} \hspace{1cm} \text{[Procedure]}

Returns an integer.

Obtains and returns the reference count of closure.

\texttt{g-closure-ref \hspace{0.5cm} closure} \hspace{1cm} \text{[Procedure]}

Returns a pointer.

Increments the reference count of closure, to force it staying alive while the caller holds a pointer to it.

\texttt{g-closure-sink \hspace{0.5cm} closure} \hspace{1cm} \text{[Procedure]}

Returns nothing.

Takes over the initial ownership of closure. Each closure is initially created in a ‘floating’ state, which means that the initial reference count is not owned by any caller. [g-closure-sink], page 70, checks to see if the object is still floating, and if so, unsets the floating state and decreases the reference count. If the closure is not floating, [g-closure-sink], page 70, does nothing.

Because [g-closure-sink], page 70, may decrement the reference count of closure (if it hasn’t been called on closure yet) just like [g-closure-unref], page 70, [g-closure-ref], page 70, should be called prior to this function.

\texttt{g-closure-unref \hspace{0.5cm} closure} \hspace{1cm} \text{[Procedure]}

Returns nothing.
Decrement the reference count of closure after it was previously incremented by the same caller. If no other callers are using closure, then it will be destroyed and freed.

**g-closure-free closure**

Returns nothing.

Decrement the reference count of closure to 0 (so closure will be destroyed and freed).

**g-closure-invoke closure return-value n-param param-vals**

Returns nothing.

Invokes the closure, i.e. executes the callback represented by the closure.

The arguments are closure (a pointer to a GClosure), return-value (a pointer to a GValue), n-param (the length of the param-vals array), param-vals (a pointer to an array of GValue) and invocation-hint (a context dependent invocation hint).

**g-closure-add-invalidate-notifier closure data function**

Returns nothing.

Registers an invalidation notifier which will be called when the closure is invalidated with **g-closure-invalidate**. Invalidation notifiers are invoked before finalization notifiers, in an unspecified order.

The data argument is (must be) a pointer to the notifier data (or #f).

**g-closure-new-simple size data**

Returns a pointer.

Allocates a structure of the given size and initializes the initial part as a GClosure. The data (if any) are used to initialize the data fields of the newly allocated GClosure. The returned value is a floating reference (a pointer) to a new GClosure.

**g-closure-set-marshal closure marshal**

Returns nothing.

Sets the closure marshaller to marshal.

**g-source-set-closure source closure**

Returns nothing.

Set the source callback to closure.

If the source is not one of the standard GLib types, the closure_callback and closure_marshal fields of the GSourceFuncs structure must have been filled in with pointers to appropriate functions.

**Signals**

G-Golf GObject Signals low level API.

Signals — A means for customization of object behaviour and a general purpose notification mechanism.
Procedures

[g-signal-query], page 72
[g-signal-lookup], page 73
[g-signal-list-ids], page 73
[g-signal-parse-name], page 73

Types and Values

[%g-signal-flags], page 73

Description

The basic concept of the signal system is that of the emission of a signal. Signals are introduced per-type and are identified through strings. Signals introduced for a parent type are available in derived types as well, so basically they are a per-type facility that is inherited.


Procedures

g-signal-query id

Returns a list.

Obtains and returns a list composed of the signal id, name, interface-type, flags, return-type, number of arguments and their types. For example:

```
(use (g-golf))
(gi-import "Clutter")

(make <clutter-actor>)
⇒ $2 = #<clutter-actor> 565218c88a80>

(!g-type (class-of $2))
⇒ $3 = 94910597864000

(g-signal-list-ids $3)
⇒ $4 = (5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30)

(g-signal-query 20)
⇒ $5 = (20 "enter-event" 94910597864000 (run-last) boolean 1 (boxed))
```

As you may have noticed, the signal query argument(s) list does not include the instance (and its type) upon which the signal is called, but both at C level and within the context of GClosure, callbacks must assume that the instance upon which a signal is called is always the first argument of the callback.

---

26 Within this context, the interface-type is the GType of the GObject subclass the signal is attached to - knowing that signals are inherited.

27 At least one GObject subclass instance must have been created prior to attempt to query any of its class signal(s).
g-signal-lookup name g-type

Retrieves a signal’s identifying integer, given the name of the signal and the object g-type it connects to. If a signal identifier cannot be found for the given name and g-type, an exception is raised.

Procedure

g-signal-list-ids g-type

Returns a list of integers.

Obtains and returns the list of signal’s identifying integers for g-type. (Note that at least one g-type instance must have been created prior to attempt to list or query signal’s identifying integers for a given g-type).

Procedure

g-signal-parse-name detailed-signal g-type [force-detail-quark #t]

Retrieves two integer values.

Obtains and returns the signal-id and a detail corresponding to detailed-signal for g-type. The detailed-signal can be passed as a symbol or a string. When force-detail-quark is #t it forces the creation of a GQuark for the detail.

If the signal name could not successfully be parsed, it raises an exception.

Types and Values

%g-signal-flags

[Instance Variable of <gi-enum>]

The signal flags are used to specify a signal’s behaviour, the overall signal description outlines how especially the RUN flags control the stages of a signal emission.

An instance of <gi-enum>, who’s members are the scheme representation of the GSignalFlags:

- g-name: GSignalFlags
- name: g-signal-flags
- enum-set:

  run-first
  Invoke the object method handler in the first emission stage.

  run-last
  Invoke the object method handler in the third emission stage.

  run-cleanup
  Invoke the object method handler in the last emission stage.

  no-recurse
  Signals being emitted for an object while currently being in emission for this very object will not be emitted recursively, but instead cause the first emission to be restarted.

  detailed
  This signal supports "::detail" appendices to the signal name upon handler connections and emissions.

  action
  Action signals are signals that may freely be emitted on alive objects from user code via g-signal-emit and friends, without the need of being embedded into extra code that performs pre or post emission adjustments on the object. They
can also be thought of as object methods which can be called generically by third-party code.

**no-hooks**  
No emissions hooks are supported for this signal.

**must-collect**  
Varargs signal emission will always collect the arguments, even if there are no signal handlers connected. Since 2.30.

**deprecated**  
The signal is deprecated and will be removed in a future version. A warning will be generated if it is connected while running with `G_ENABLE_DIAGNOSTIC=1`. Since 2.32.

---

**GObject Introspection**

G-Golf GObject Introspection modules are defined in the `gi` subdirectory, such as `(g-golf gi repository)`.

Where you may load these modules individually, the easiest way to use G-Golf GObject Introspection is to import the `g-golf` module, which imports and re-exports the public interface of all modules used and defined by G-Golf (for a complete list, visit its source definition):

```
(use-modules (g-golf))
```

Most G-Golf GObject Introspection modules correspond to a GObject Introspection (manual) section, but there are some exceptions, such as `init` and `utils` ...

---

**Repository**

G-Golf Introspection Repository low level API.

GIRepository — GObject Introspection repository manager.

---

**Procedures**

- `[g-irepository-get-default], page 75`
- `[g-irepository-get-dependencies], page 75`
- `[g-irepository-get-loaded-namespaces], page 75`
- `[g-irepository-get-n-infos], page 75`
- `[g-irepository-get-info], page 75`
- `[g-irepository-enumerate-versions], page 75`
- `[g-irepository-get-typelib-path], page 75`
- `[g-irepository-require], page 76`
- `[g-irepository-get-c-prefix], page 76`
- `[g-irepository-get-shared-library], page 76`
- `[g-irepository-get-version], page 76`
- `[g-irepository-find-by-gtype], page 76`
- `[g-irepository-find-by-name], page 76`

**Description**

GIRepository is used to manage repositories of namespaces. Namespaces are represented on disk by type libraries (.typelib files).
Object Hierarchy

GOBJECT

Object Hierarchy

GOBJECT

Note: in this section, when the #:repository optional keyword argument is passed, it is [must be] a pointer to a GIRepository. Its default value is #f, the scheme representation for NULL, meaning the singleton process-global default GIRepository (see [g-irepository-get-default], page 75).

**g-irepository-get-default**

Returns a pointer to the singleton process-global default GIRepository.

GOBJECT Introspection does not currently support multiple repositories in a particular process, but this procedure is provided in the unlikely eventuality that it would become possible.

All G-Golf low level API procedures on GIRepository also accept an optional #:repository keyword argument which defaults to #f, meaning this singleton process-global default GIRepository.

**g-irepository-get-dependencies namespace [#:repository #f]**

Returns a list of all (transitive) versioned dependencies for namespace. Returned string are of the form namespace-version.

Note: The namespace must have already been loaded using a procedure such as g-irepository-require before calling this procedure.

**g-irepository-get-loaded-namespaces [#:repository #f]**

Return the list of currently loaded namespaces.

**g-irepository-get-n-infos namespace [#:repository #f]**

Returns the number of metadata entries in namespace. The namespace must have already been loaded before calling this procedure.

**g-irepository-get-info namespace index [#:repository #f]**

Returns a pointer to a particular metadata entry in the given namespace.

The namespace must have already been loaded before calling this procedure. See g-irepository-get-n-infos to find the maximum number of entries.

index is a 0-based offset into namespace for entry.

**g-irepository-enumerate-versions namespace [#:repository #f]**

Returns a (possibly empty) list.

Obtains and returns an unordered (possibly empty) list of versions (either currently loaded or available) for namespace in repository.

**g-irepository-get-typelib-path namespace [#:repository #f]**

Returns the full path to the typelib file namespace was loaded from, if loaded. If namespace is not loaded or does not exist, it will return #f. If the typelib for namespace was included in a shared library, it returns the special string "<builtin>".
g-irepository-require namespace [#:version #f] [#:repository #f]  [Procedure]

Returns a pointer a GITypelib structure, if the Typelib file for namespace exists. Otherwise, it raises an error.

Force the namespace to be loaded if it isn’t already. If namespace is not loaded, this procedure will search for a ".typelib" file using the repository search path. In addition, a version version of namespace may be specified. If version is not specified, the latest will be used.

g-irepository-get-c-prefix namespace [#:repository #f]  [Procedure]

Returns the "C prefix", or the C level namespace associated with the given introspection namespace. Each C symbol starts with this prefix, as well each GType in the library.

Note: The namespace must have already been loaded using a procedure such as g-irepository-require before calling this procedure.

g-irepository-get-shared-library namespace [#:repository #f]  [Procedure]

Returns a list of paths to the shared C libraries associated with the given namespace. There may be no shared library path associated, in which case this procedure will return an empty list.

g-irepository-get-version namespace [#:repository #f]  [Procedure]

Returns the loaded version associated with the given namespace.

Note: The namespace must have already been loaded using a procedure such as g-irepository-require before calling this procedure.

g-irepository-find-by-gtype gtype [#:repository #f]  [Procedure]

Returns a pointer to a GIBaseInfo representing metadata about gtype, or #f.

Searches all loaded namespaces for a particular GType. Note that in order to locate the metadata, the namespace corresponding to the type must first have been loaded. There is currently no mechanism for determining the namespace which corresponds to an arbitrary GType - thus, this procedure will operate most reliably when you know the GType to originate from be from a loaded namespace.

g-irepository-find-by-name namespace name [#:repository #f]  [Procedure]

Returns a pointer to a GIBaseInfo representing metadata about type, or #f.

Searches for a particular entry in namespace. Before calling this function for a particular namespace, you must call g-irepository-require once to load the namespace, or otherwise ensure the namespace has already been loaded.

Typelib
G-Golf Typelib low level API.
GITypelib — Layout and accessors for typelib.
Procedures

[g-golf-typelib-new], page 77
[call-with-input-typelib], page 77
[g-typelib-new-from-memory], page 77
[g-typelib-free], page 77
[g-typelib-get-namespace], page 77

Description

TODO.

Procedures

Note: in this section, the typelib argument is [must be] a pointer to a GITypelib.

[g-golf-typelib-new file] [Procedure]
Returns a pointer to a new GITypelib.
file must be a valid typelib filename.
This procedure actually sets things up and calls [g-typelib-new-from-memory], page 77.

call-with-input-typelib file proc [Procedure]
Returns the value(s) returned by proc.
file must be a valid typelib filename. Makes a new GITypelib by calling (g-golf-typelib-new file) and calls (proc typelib) with the resulting GITypelib.
When proc returns, the GITypelib is free’d by calling g-typelib-free. Otherwise the [Glib - C] memory chunk might not be free’d automatically, though the scheme pointer returned by g-golf-typelib-new will be garbage collected in the usual way if not otherwise referenced.

g-typelib-new-from-memory pointer size gerror [Procedure]
Returns a pointer to a new GITypelib.
pointer must be the address of a memory chunk containing the typelib, size is the number of bytes of the memory chunk containing the typelib, and gerror a pointer to a GError.
Creates a new GITypelib from a memory location. The memory block pointed to by typelib will be automatically g_free()d when the repository is destroyed.

g-typelib-free typelib [Procedure]
Returns nothing.
Free a GITypelib.

g-typelib-get-namespace typelib [Procedure]
Returns the namespace of typelib.

Common Types

G-Golf Common Types low level API.
common types - TODO
III. G-Golf Core Reference

Procedures

[g-type-tag-to-string], page 78

Types and Values

[%gi-type-tag], page 78
[%gi-array-type], page 79

Procedures

[g-type-tag-to-string] type-tag

[Procedure]

Returns a string or #f.

Obtains the string representation for type-tag or #f if it does not exist (note that in this case, the upstream function returns "unknown").

type-tag can either be an id or a symbol, a member of the enum-set of [%gi-type-tag], page 78.

Types and Values

[%gi-type-tag]

An instance of <gi-enum>, who’s members are the type tag of a GITypeInfo:

- g-name: GITag
- name: gi-type-tag
- enum-set:
  - void
  - boolean
  - int8
  - uint8
  - int16
  - uint16
  - int32
  - uint32
  - int64
  - uint64
  - float
  - double
  - gtype
  - utf8
  - filename
  - array
  - interface
  - glist
  - gslist
  - ghash
  - error
  - unichar
%\texttt{gi-array-type} \quad [\text{Instance Variable of} \ <\text{gi-enum}>]

An instance of \texttt{gi-enum}, who’s members are the type of array in a GITypeInfo:

\begin{verbatim}
g-name: GIArrayType
name: gi-array-type
enum-set:
c
array
ptr-array
byte-array
\end{verbatim}

Version Information

G-Golf GIRepository Version Information low level API.

Version Information - Procedures to check the GIRepository version.

Procedures

\begin{verbatim}
[gi-version], page 79
[gi-effective-version], page 79
[gi-major-version], page 79
[gi-minor-version], page 79
[gi-micro-version], page 79
[gi-check-version], page 79
\end{verbatim}

Description

Procedures to check the GIRepository version.

Procedures

\begin{verbatim}
\texttt{gi-version} \quad [\text{Procedure}]
\texttt{gi-effective-version} \quad [\text{Procedure}]
\texttt{gi-major-version} \quad [\text{as-integer? \ #f}] \quad [\text{Procedure}]
\texttt{gi-minor-version} \quad [\text{as-integer? \ #f}] \quad [\text{Procedure}]
\texttt{gi-micro-version} \quad [\text{as-integer? \ #f}] \quad [\text{Procedure}]
\end{verbatim}

Returns a string describing GIRepository full version number, effective version number, major, minor or micro version number, respectively.

The last three procedures will return the major, minor or micro version number as an integer if the optional \texttt{as-integer?} argument is \texttt{#t}.

\begin{verbatim}
\texttt{gi-check-version} \texttt{major minor micro} \quad [\text{Procedure}]
\end{verbatim}

Returns \texttt{#t} if the GIRepository version is the same as or newer than the \texttt{major minor micro} passed-in version.

Base Info

G-Golf Base Info low level API.

\texttt{GIBaseInfo} — Base struct for all GITypeLib structs.
Procedures

[g-base-info-ref], page 80
[g-base-info-unref], page 81
[g-base-info-equal], page 81
[g-base-info-get-type], page 81
[g-base-info-get-typelib], page 81
[g-base-info-get-namespace], page 81
[g-base-info-get-name], page 81
[g-base-info-get-attribute], page 81
[g-base-info-iterate-attributes], page 81
[g-base-info-get-container], page 81
[g-base-info-is-deprecated], page 81

Types and Values

[%gi-info-type], page 82

Struct Hierarchy

GIBaseInfo
  +— GIArgInfo
  +— GICallableInfo
  +— GIConstantInfo
  +— GIFieldInfo
  +— GIPROPERTY_INFO
  +— GIRegisteredTypeInfo
  +— GITypeInfo

Description

GIBaseInfo is the common base struct of all other *Info structs accessible through the GIRepository API.

Most GIRepository APIs returning a GIBaseInfo is actually creating a new struct, in other words, [g-base-info-unref], page 81, has to be called when done accessing the data. GIBaseInfos are normally accessed by calling either [g-irepository-find-by-name], page 76, [g-irepository-find-by-gtype], page 76, or [g-irepository-get-info], page 75.

Example: Getting the Button of the Gtk typelib

  ,use (g-golf gi)
  (g-irepository-require "Gtk")
  (g-irepository-find-by-name "Gtk" "Button")
  ⇒ $4 = #<pointer 0x20e0000>
  ... use button info ...
  (g-base-info-unref $4)

Procedures

Note: in this section, the info, info1 and info2 arguments are [must be] pointers to a GIBaseInfo.
III. G-Golf Core Reference

```
g-base-info-ref info
Returns the same info.
Increases the reference count of info.

```

```
g-base-info-unref info
Returns nothing.
Decreases the reference count of info. When its reference count drops to 0, the info
is freed.

```

```
g-base-info-equal info1 info2
Returns #t if and only if info1 equals info2.
Compares two GIBaseInfo.
Using pointer comparison is not practical since many functions return different in-
stances of GIBaseInfo that refers to the same part of the typelib: use this procedure
instead to do GIBaseInfo comparisons.

```

```
g-base-info-get-type info
Returns the info type of info.

```

```
g-base-info-get-typelib info
Returns a pointer to the GITypelib the info belongs to.

```

```
g-base-info-get-namespace info
Returns the namespace of info

```

```
g-base-info-get-name info
Returns the name of info or #f if it lacks a name.
What the name represents depends on the GIInfoType of the info. For instance for
GIFunctionInfo it is the name of the function.

```

```
g-base-info-get-attribute info name
Returns the value of the attribute or #f if not such attribute exists.

```

```
g-base-info-iterate-attributes info proc
Returns nothing.
Iterate and calls proc over all attributes associated with this node. proc must be a
procedure of two arguments, the name and the value of the attribute.

```

```
g-base-info-get-container info
Returns a pointer to a GIBaseInfo.
The container is the parent GIBaseInfo. For instance, the parent of a
GIFunctionInfo is an GIOBJECTINFO or GIINTERFACEINFO.

```

```
g-base-info-is-deprecated info
Returns #t if deprecated.
Obtain whether info represents a metadata which is deprecated or not.
```
Types and Values

\texttt{\%gi-info-type} [Instance Variable of \texttt{<gi-enum>}]

An instance of \texttt{<gi-enum>}, who’s members are the scheme representation of the type of a \texttt{GIBaseInfo} struct:

\begin{verbatim}
g-name: GIInfoType
name: gi-info-type
enum-set:
  invalid
  function
  callback
  struct
  boxed
  enum
  flags
  object
  interface
  constant
  error-domain
  union
  value
  signal
  vfunc
  property
  field
  arg
  type
  unresolved
\end{verbatim}

Callable Info

G-Golf Callable Info low level API.

\texttt{GICallableInfo} — Struct representing a callable.

Procedures

\begin{verbatim}
[g-callable-info-get-n-args], page 83
[g-callable-info-get-arg], page 83
[g-callable-info-get-caller-owns], page 83
[g-callable-info-get-instance-ownership-transfer], page 83
[g-callable-info-get-return-type], page 83
[g-callable-info-is-method], page 83
[g-callable-info-may-return-null], page 84
[g-callable-info-create-closure], page 84
\end{verbatim}

Struct Hierarchy

\begin{verbatim}
GIBaseInfoInfo
  +— GICallableInfo
\end{verbatim}
+++ GIFunctionInfo
+++ GICallbackInfo
+++ GISignalInfo
+++ GIVFuncInfo

Description

GICallableInfo represents an entity which is callable. Examples of callable are: functions (GIFunctionInfo), virtual functions, (GIVFuncInfo), callbacks (GICallbackInfo).

A callable has a list of arguments (GIArgInfo), a return type, direction and a flag which decides if it returns null.

Procedures

Note: in this section, the info argument is [must be] a pointer to a GICallableInfo.

**g-callable-info-get-n-args info**

Returns the number of arguments this info expects.

Obtain the number of arguments (both IN and OUT) for this info.

**g-callable-info-get-arg info n**

Returns a pointer to the nth GIArgInfo of info.

It must be freed by calling [g-base-info-unref], page 81, when done accessing the data.

**g-callable-info-get-caller-owns info**

Returns a GITransfer enumerated value.

See whether the caller owns the return value of this callable. See [%gi-transfer], page 104, for the list of possible values.

**g-callable-info-get-instance-ownership-transfer info**

Returns a GITransfer enumerated value.

Obtains the ownership transfer for the instance argument. See [%gi-transfer], page 104, for the list of possible values.

**g-callable-info-get-return-type info**

Returns a pointer to the GITypeInfo.

It must be freed by calling [g-base-info-unref], page 81, when done accessing the data.

**g-callable-info-is-method info**

Returns #t if the callable info is a method, otherwise it return #f.

Determines if the callable info is a method. For GIVFuncInfo and GISignalInfo, this is always true. Otherwise, this looks at the GI_FUNCTION_IS_METHOD flag on the GIFunctionInfo.

Concretely, this function returns whether [g-callable-info-get-n-args], page 83, matches the number of arguments in the raw C method. For methods, there is one more C argument than is exposed by introspection: the ‘self’ or ‘this’ object.
g-callable-info-may-return-null info [Procedure]

Returns #t if the callable info could return NULL.

See if a callable could return NULL.

g-callable-info-create-closure info ffi-cif ffi-closure-callback user-data [Procedure]

Returns the ffi-closure or #f on error.

The return value should be freed by calling g-callable-info-destroy-closure.

Signal Info
G-Golf Signal Info low level API.
GISignalInfo — Struct representing a signal.

Procedures

[g-signal-info-get-flags], page 84

Description
GISignalInfo represents a signal. It’s a sub-struct of GICallableInfo and contains a set of flags and a class closure.

See also [Callable Info], page 82, for information on how to retrieve arguments and other metadata from the signal.

Struct Hierarchy

GIBaseInfoInfo
  +— GICallableInfo
    +— GIFunctionInfo
    +— GISignalInfo
    +— GIVFuncInfo

Procedures
Note: in this section, the info argument is [must be] a pointer to a GISignalInfo.

g-signal-info-get-flags info [Procedure]

Returns a list of [%g-signal-flags], page 73.

Obtain the flags for this signal info. See [%g-signal-flags], page 73, for more information about possible flag values.

Function Info
G-Golf Function Info low level API.
GIFunctionInfo — Struct representing a function.
Procedures

[gi-function-info-is-method?], page 85
[g-function-info-get-flags], page 85
[g-function-info-get-property], page 85
[g-function-info-get-symbol], page 85
[g-function-info-get-vfunc], page 86
[g-function-info-invocation], page 86

Types and Values

[%g-function-info-flags], page 86

Struct Hierarchy

GIBaseInfoInfo
  +— GICallableInfo
    +— GIFunctionInfo
    +— GISignalInfo
    +— GIVFuncInfo

Description

GIFunctionInfo represents a function, method or constructor. To find out what kind of
entity a GIFunctionInfo represents, call [g-function-info-get-flags], page 85.

See also [Callable Info], page 82, for information on how to retrieve arguments and other
metadata.

Procedures

Note: in this section, the info argument is [must be] a pointer to a GIFunctionInfo.

gi-function-info-is-method? info [flags #f] [Procedure]
Returns #t if info is a method, that is if is-method is a member of the info flags.
Otherwise, it returns #f.

The optional flags argument, if passed, must be the list of the function info flags as
returned by [g-function-info-get-flags], page 85.

g-function-info-get-flags info [Procedure]
Returns a list of [%g-function-info-flags], page 86.
Obtain the GIFunctionInfoFlags for info.

g-function-info-get-property info [Procedure]
Returns a pointer or #f.

Obtains the GIPROPERTYINFO associated with info. Only GIFunctionInfo with the
flag is-getter or is-setter have a property set. For other cases, #f will be returned.
The GIPROPERTYINFO must be freed by calling [g-base-info-unref], page 81, when done.

g-function-info-get-symbol info [Procedure]
Returns a string.
Obtain the ‘symbol’ of the function\(^{28}\).

\textbf{g-function-info-get-vfunc info} \hspace{1cm} \textbf{[Procedure]}

Returns a pointer or \#f.

Obtains the GIVFuncInfo associated with \textit{info}. Only GIFunctionInfo with the flag \texttt{wraps-vfunc} has its virtual function set. For other cases, \#f will be returned.

The GIVFuncInfo must be freed by calling [g-base-info-unref], page 81, when done.

\textbf{g-function-info-invoke info in-args n-in out-args n-out r-val g-error} \hspace{1cm} \textbf{[Procedure]}

Returns \#t if the function has been invoked, \#f if an error occurred.

Invokes the function described in \textit{info} with the given arguments. Note that inout parameters must appear in both argument lists. The arguments are:

- \textit{info} \hspace{1cm} a pointer to a GIFunctionInfo describing the function to invoke.
- \textit{in-args} \hspace{1cm} a pointer to an array of GIArguments, one for each \texttt{in} and \texttt{inout} parameter of \textit{info}. If there are no \texttt{in} parameter, \textit{in-args} must be the \%null-pointer.
- \textit{n-in} \hspace{1cm} the length of the \textit{in-args} array.
- \textit{out-args} \hspace{1cm} a pointer to an array of GIArguments, one for each \texttt{out} and \texttt{inout} parameter of \textit{info}. If there are no \texttt{out} parameter, \textit{out-args} must be the \%null-pointer.
- \textit{n-out} \hspace{1cm} the length of the \textit{out-args} array.
- \textit{r-val} \hspace{1cm} a pointer to a GIArguments, the return location for the return value of the function. If the function returns \texttt{void}, \textit{r-val} must be the \%null-pointer.
- \textit{g-error} \hspace{1cm} a pointer to a newly allocated (and ‘empty’) GError (the recommended way for procedure calls that need such a pointer is to ‘surround’ the call using \texttt{[with-gerror]}, page 112).

\textbf{Types and Values}

\%g-function-info-flags \hspace{1cm} \textbf{[Instance Variable of <gi-flag>]}  

An instance of <\textit{gi-flag}>, who’s members are the scheme representation of the GIFunctionInfoFlags:

\begin{itemize}
  \item \texttt{g-name}: GIFunctionInfoFlags
  \item \texttt{name}: gi-function-info-flags
  \item \texttt{enum-set}:
    \begin{itemize}
      \item \texttt{is-method} \hspace{1cm} \texttt{Is a method.}
    \end{itemize}
\end{itemize}

\(^{28}\) As you have noticed already, since \texttt{g-function-info-get-symbol} returns a string, in the Glib, GObject and GObject Introspection worlds, symbol has a different meaning then in the Lisp/Scheme worlds. However, since the procedure is part of the G-Golf low-level API, we decided to keep its name as close as the original name as possible, which in Glib terms is the name of the exported function, ‘\textit{suitable to be used as an argument to g_module_symbol()}’
is-constructor
Is a constructor.

is-getter
Is a getter of a GIPropertyInfo.

is-setter
Is a setter of a GIPropertyInfo.

wraps-vfunc
Represent a virtual function.

throws
The function may throw an error.

Registered Type Info

G-Golf Registered Type Info low level API.
GIRegisteredTypeInfo — Struct representing a struct with a GType.

Procedures

[g-registered-type-info-name], page 87
[g-registered-type-info-get-type-name], page 88
[g-registered-type-info-get-type-init], page 88
[g-registered-type-info-get-g-type], page 88

Struct Hierarchy

GIBaseInfo
    +—-GIRegisteredTypeInfo
        +—-GIEnumInfo
        +—-GIInterfaceInfo
        +—-GIOBJECTINFO
        +—-GIStructInfo
        +—-GIUnionInfo

Description

GIRegisteredTypeInfo represents an entity with a GType associated. Could be either a GIEnumInfo, GIInterfaceInfo, GIOBJECTINFO, GIStructInfo or a GIUnionInfo.

A registered type info struct has a name and a type function.

Procedures

Note: in this section, the info argument is [must be] a pointer to a GIRegisteredTypeInfo.

gi-registered-type-info-name info
[Procedure]
Returns a type name.

Some registered type are not ‘registered’, and calling [g-registered-type-info-get-type-name], page 88, returns #f29.

29 Another symptom for those is that if you call ([g-type-name], page 55, g-type), it returns “void”.

Even though they are ‘unnamed’, some are present in their typelib, like "GLib" "SpawnFlags", or "GObject" "ParamFlags", and may be imported - sometimes manually, sometimes automatically.

In G-Golf, imported GIRegisteredTypeInfo must have a unique name, since it is used as the secondary key in its cache ‘mechanism’ (See [Cache Park], page 31).

Obtains and returns a unique name for info. If [g-registered-type-info-get-type-name], page 88, returns a name, that name is returned. Otherwise, it returns a name composed of the namespace and name for info.

Here is an example, to illustrate:

```lisp
(g-irepository-find-by-name "GObject" "ParamFlags")
⇒ $2 = #<pointer 0x5654c59ee4f0>

(g-registered-type-info-get-type-name $2)
⇒ $3 = #f

(gi-registered-type-info-name $2)
⇒ $4 = "GObjectParamFlags"

(g-name->name $4)
⇒ $5 = g-object-param-flags
```

**g-registered-type-info-get-type-name info**

[Procedure]

Returns the type name.

Obtain the type name of the struct within the GObject type system. This name can be passed to g_type_from_name to get a GType.

**g-registered-type-info-get-type-init info**

[Procedure]

Returns the name of the type init function.

Obtain the type init function for info. The type init function is the function which will register the GType within the GObject type system. Usually this is not called by language bindings or applications.

**g-registered-type-info-get-g-type info**

[Procedure]

Returns the GType for info.

Obtain the GType for this registered type or G_TYPE_NONE which has a special meaning. It means that either there is no type information associated with this info or that the shared library which provides the type_init function for this info cannot be called.

**Enum Info**

G-Golf Enum Info low level API.

GIEnumInfo — Structs representing an enumeration and its values.
Procedures

[gi-enum-import], page 89
[gi-enum-value-values], page 89
[g-enum-info-get-n-values], page 89
[g-enum-info-get-value], page 89
[g-enum-info-get-n-methods], page 90
[g-enum-info-get-method], page 90
[g-value-info-get-value], page 90

Struct Hierarchy

GIBaseInfo
  +— GIRegisteredTypeInfo
  +— GIEnumInfo

Description

GIEnumInfo represents an argument. An argument is always part of a GICallableInfo.

Procedures

Note: in this section, unless otherwise specified, the info argument is [must be] a pointer to a GIEnumInfo.

gi-enum-import info
  [Procedure]
  Returns a <gi-enum> instance.
  Obtains the values this enumeration contains, then makes and returns a <gi-enum> instance.

gi-enum-value-values info
  [Procedure]
  Returns an alist.
  Obtains and returns the list pairs (symbol . id) the enum GI definition pointed by info contains. If you think the name is strange, compare it with, for example [gi-struct-field-types], page 91: just like a GITStructInfo holds a list of pointers to GIFieldInfo from which we get the (field) type, aGIEnumInfo holds a list of pointers to GIValueInfo from which we get the (enum) value - which in the GI world is a name (a string) that we transform, in the scheme world, to a symbol.

g-enum-info-get-n-values info
  [Procedure]
  Returns the number of values.
  Obtains the number of values this enumeration contains.

g-enum-info-get-value info index
  [Procedure]
  Returns a pointer to a GIValueInfo or #f if type tag is wrong.
  Obtains a value for this enumeration. The GIValueInfo must be free’d using g-base-info-unref when done.
  index is a 0-based offset into info for a value.
III. G-Golf Core Reference

**Procedure**

**g-enum-info-get-n-methods info**  
[Procedure]  
Returns the number of methods.  
Obtains the number of methods this enumeration has.

**g-enum-info-get-method info index**  
[Procedure]  
Returns a pointer to a GIFunctionInfo or #f if type tag is wrong.  
Obtains a method for this enumeration. The GIFunctionInfo must be free’d using g-base-info-unref when done.  
*index* is a 0-based offset into info for a method.

**g-value-info-get-value info**  
[Procedure]  
Returns the enumeration value.  
Obtains a value of the GIValueInfo.  
info is [must be] a pointer to a GIValueInfo.

**Struct Info**

G-Golf Struct Info low level API.  
GIStructInfo — Structs representing a C structure.

**Procedures**

[gi-struct-import], page 90
[gi-struct-field-desc], page 91
[gi-struct-field-types], page 91
[g-struct-info-get-alignment], page 91
[g-struct-info-get-size], page 91
[g-struct-info-is-gtype-struct], page 91
[g-struct-info-is-foreign], page 91
[g-struct-info-get-n-fields], page 91
[g-struct-info-get-field], page 91
[g-struct-info-get-n-methods], page 91
[g-struct-info-get-method], page 92

**Struct Hierarchy**

GIStructInfo
to GRegisterTypeInfo
to GIBaseInfo

**Description**

GIStructInfo represents a generic C structure type.  
A structure has methods and fields.

**Procedures**

Note: in this section, unless otherwise specified, the *info* argument is [must be] a pointer to a GIStructInfo.
**gi-struct-import info** [Procedure]

Returns a `<gi-struct>` instance.

Obtains the list of (field) types the C struct GI definition pointed by `info` contains, then makes and returns a `<gi-struct>` instance.

**gi-struct-field-desc info** [Procedure]

Returns a list.

Obtains and returns the list of (field) descriptions for `info`. A field description is a list: `(name type-tag offset flags)`.

**gi-struct-field-types info** [Procedure]

Returns a list.

Obtains and returns the list of (field) types the C struct GI definition pointed by `info` contains.

**g-struct-info-get-alignment info** [Procedure]

Returns an integer.

Obtains and returns the required alignment for `info`.

**g-struct-info-get-size info** [Procedure]

Returns an integer.

Obtains and returns the total size of the structure specified `info`.

**g-struct-info-is-gtype-struct info** [Procedure]

Returns #t or #f.

Return true if the structure specified by `info` represents the "class structure" for some GObject or GInterface.

**g-struct-info-is-foreign info** [Procedure]

Returns #t or #f.

FIXME. No upstream documentation, though the procedure works.

**g-struct-info-get-n-fields info** [Procedure]

Returns an integer.

Obtains the number of fields for `info`.

**g-struct-info-get-field info n** [Procedure]

Returns a pointer.

Obtains and returns the `info` type information (a pointer to a GIFieldInfo) for the field at the specified `n` index.

The GIFieldInfo must be freed by calling `[g-base-info-unref]`, page 81, when done.

**g-struct-info-get-n-methods info** [Procedure]

Returns an integer.

Obtains the number of methods for `info`. 
**g-struct-info-get-method**  \textit{info} \textit{n}  

Procedure

Returns a pointer.

Obtains and returns the \textit{info} type information (a pointer to a \texttt{GIFunctionInfo}) for the method at the specified \textit{n} index.

The \texttt{GIFunctionInfo} must be freed by calling \[g-base-info-unref\], page 81, when done.

**Union Info**

G-Golf Union Info low level API.

\texttt{GIUnionInfo} — Struct representing a C union.

**Procedures**

- \[g-union-info-get-n-fields\], page 92
- \[g-union-info-get-field\], page 92
- \[g-union-info-get-n-methods\], page 93
- \[g-union-info-get-method\], page 93
- \[g-union-info-is-discriminated\?\], page 93
- \[g-union-info-get-discriminator-offset\], page 93
- \[g-union-info-get-discriminator-type\], page 93
- \[g-union-info-get-discriminator\], page 93
- \[g-union-info-get-size\], page 93
- \[g-union-info-get-alignment\], page 93

**Description**

\texttt{GIUnionInfo} represents a union type.

A union has methods and fields. Unions can optionally have a discriminator, which is a field deciding what type of real union fields is valid for specified instance.

**Struct Hierarchy**

\begin{verbatim}
GIBaseInfo
    +--- GIRegisteredTypeInfo
        +--- GIUnionInfo
\end{verbatim}

**Procedures**

Note: in this section, unless otherwise specified, the \textit{info} argument is [must be] a pointer to a \texttt{GIUnionInfo}.

\textbf{g-union-info-get-n-fields}  \textit{info}  

Procedure

Returns an integer.

Obtains and returns the number of fields the \textit{info} union has.

\textbf{g-union-info-get-field}  \textit{info} \textit{n}  

Procedure

Returns a pointer.

Obtains and returns a pointer to the \texttt{GIFieldInfo} for \textit{info}, given its \textit{n}. The \texttt{GIFieldInfo} must be free’d by calling \[g-base-info-unref\], page 81, when done.
g-union-info-get-n-methods info
Returns an integer.
Obtains and returns the number of methods the info union has.

g-union-info-get-method info n
Returns a pointer.
Obtains and returns a pointer to the GIFunctionInfo for info, given its n, which must be free’d by calling [g-base-info-unref], page 81, when done.

g-union-info-is-discriminated? info
Returns #t if info contains a discriminator field, otherwise it returns #f.

g-union-info-get-discriminator-offset info
Returns an integer.
Obtains and returns the offset of the discriminator field for info.

g-union-info-get-discriminator-type info
Returns a pointer.
Obtains and returns a pointer to the GITypeInfo for info, which must be free’d by calling [g-base-info-unref], page 81, when done.

g-union-info-get-discriminator info n
Returns a pointer.
Obtains and returns a pointer to the GIConstantInfo assigned for the info n-th union field - i.e. the n-th union field is the active one if discriminator contains this constant (value) - which must be free’d by calling [g-base-info-unref], page 81, when done.

g-union-info-get-size info
Returns an integer.
Obtains and returns the total size of the union specified by info.

g-union-info-get-alignment info
Returns an integer.
Obtains and returns the required alignment for info.

Object Info
G-Golf Object Info low level API.
GIOBJECTINFO — Structs representing a GObject.
Procedures

[gobject-show], page 94
[gobject-property-names], page 95
[gobject-method-names], page 95
[gobject-method-find-by-name], page 96
[gobject-info-get-abstract], page 96
[gobject-info-get-parent], page 96
[gobject-info-get-type-name], page 96
[gobject-info-get-type-init], page 96
[gobject-info-get-n-constants], page 96
[gobject-info-get-constant], page 96
[gobject-info-get-n-fields], page 96
[gobject-info-get-field], page 96
[gobject-info-get-n-interfaces], page 96
[gobject-info-get-interface], page 96
[gobject-info-get-n-methods], page 97
[gobject-info-get-method], page 97
[gobject-info-find-method], page 97
[gobject-info-get-n-properties], page 97
[gobject-info-get-property], page 97
[gobject-info-get-n-signals], page 97
[gobject-info-get-signal], page 97
[gobject-info-find-signal], page 97
[gobject-info-get-n-vfunc], page 97
[gobject-info-get-vfunc], page 97
[gobject-info-get-class-struct], page 97

Struct Hierarchy

GIBaseInfo
  +— GIRegisteredTypeInfo
    +— GIOBJECTINFO

Description

This doesn't represent a specific instance of a GObject, instead this represent the object type (eg class).

A GObject has methods, fields, properties, signals, interfaces, constants and virtual functions.

Procedures

Note: in this section, unless otherwise specified, the info argument is [must be] a pointer to a GIOBJECTINFO.

[gobject-show] info
  [Procedure]

  Returns nothing.
Obtains and displays the following informations about the object (and its parent) pointed to by info:

```lisp
,use (g-golf)
(g-irepository-require "Clutter")
⇒ $2 = #<pointer 0x56396a4f9f80>

(g-irepository-find-by-name "Clutter" "Actor")
⇒ $3 = #<pointer 0x56396a4fdc00>

(gi-object-show $3)
⊣
⊣
⊣
⊣
#<pointer 0x56396a4fdc00> is a (pointer to a) GIObjectInfo:
⊣
⊣
⊣
⊣
Parent:
⊣
⊣
⊣
⊣
namespace: "GObject"
⊣
⊣
⊣
⊣
name: "InitiallyUnowned"
⊣
⊣
⊣
⊣
g-type: 94804596757600
⊣
⊣
⊣
⊣
g-type-name: "GInitiallyUnowned"
⊣
⊣
⊣
⊣
Object:
⊣
⊣
⊣
⊣
namespace: "Clutter"
⊣
⊣
⊣
⊣
name: "Actor"
⊣
⊣
⊣
⊣
g-type: 94804596864480
⊣
⊣
⊣
⊣
g-type-name: "ClutterActor"
⊣
⊣
⊣
⊣
abstract: #f
⊣
⊣
⊣
⊣
n-constants: 0
⊣
⊣
⊣
⊣
n-fields: 4
⊣
⊣
⊣
⊣
n-interfaces: 4
⊣
⊣
⊣
⊣
n-methods: 238
⊣
⊣
⊣
⊣
n-properties: 82
⊣
⊣
⊣
⊣
n-signals: 26
⊣
⊣
⊣
⊣
n-vfuncts: 35
```

**gi-object-property-names info** [Procedure]

Obtains and returns the (possibly empty) list of the (untranslated) GI property names for info (see [g-name->name], page 123, to obtain their scheme representation).

**gi-object-method-names info** [Procedure]

Obtains and returns the (possibly empty) list of pairs of the (untranslated) GI method names for info (see [g-name->name], page 123, to obtain their scheme representation).

Each pair is composed of the info [g-function-info-get-symbol], page 85, and [g-base-info-get-name], page 81, names.
### Procedure

**gi-object-method-find-by-name**

info name

Returns a pointer or #f.

Obtains and returns a pointer to the method GIFunctionInfo contained in info, for which [g-function-info-get-symbol], page 85, is string=? to name. If there is such method, it returns #f.

**g-object-info-get-abstract**

info

Returns #t if the info object type is abstract.

Obtain if the object type is an abstract type, eg if it cannot be instantiated.

**g-object-info-get-parent**

info

Returns a pointer or #f.

Obtains and returns a pointer to the info’s parent GIObjectInfo, or #f if info has no parent.

**g-object-info-get-type-name**

info

Returns the name of the object type for info.

Obtain the name of the object class/type for info.

**g-object-info-get-type-init**

info

Returns a function name (a string).

Obtain the function name which when called will return the GType function for which this object type is registered.

**g-object-info-get-n-constants**

info

Returns the number of constants for info.

Obtain the number of constants that this object type has.

**g-object-info-get-constant**

info n

Returns a pointer to the n\textsuperscript{th} GIConstantInfo of info.

It must be freed by calling [g-base-info-unref], page 81, when done accessing the data.

**g-object-info-get-n-fields**

info

Returns the number of fields for info.

Obtain the number of fields that this object type has.

**g-object-info-get-field**

info n

Returns a pointer to the n\textsuperscript{th} GIFieldInfo of info.

It must be freed by calling [g-base-info-unref], page 81, when done accessing the data.

**g-object-info-get-n-interfaces**

info

Returns the number of interfaces for info.

Obtain the number of interfaces that this object type has.

**g-object-info-get-interface**

info n

Returns a pointer to the n\textsuperscript{th} GIInterfaceInfo of info.

It must be freed by calling [g-base-info-unref], page 81, when done accessing the data.
g-object-info-get-n-methods info
Returns the number of methods for info.
Obtain the number of methods that this object type has.

Procedure

GOLF Core Reference

g-object-info-get-method info n
Returns a pointer to the nth GIFunctionInfo of info.
It must be freed by calling [g-base-info-unref], page 81, when done accessing the data.

Procedure

g-object-info-find-method info name
Returns a pointer to a GIFunctionInfo or #f if there is no method available with that name.
It must be freed by calling [g-base-info-unref], page 81, when done accessing the data.

Procedure

g-object-info-get-n-properties info
Returns the number of properties for info.
Obtain the number of properties that this object type has.

Procedure

g-object-info-get-property info n
Returns a pointer to the nth GIPropertyInfo of info.
It must be freed by calling [g-base-info-unref], page 81, when done accessing the data.

Procedure

GObject has two structures: an instance structure and a class structure. This function returns a pointer to the info class structure.

g-object-info-get-n-signals info
Returns the number of signals for info.
Obtain the number of signals that this object type has.

Procedure

g-object-info-get-signal info n
Returns a pointer to the nth GISignalInfo of info.
It must be freed by calling [g-base-info-unref], page 81, when done accessing the data.

Procedure

g-object-info-find-signal info name
Returns a pointer to a GISignalInfo or #f if there is no signal available with that name.
It must be freed by calling [g-base-info-unref], page 81, when done accessing the data.

Procedure

g-object-info-get-n-vfuncs info
Returns the number of vfuncs for info.
Obtain the number of vfuncs that this object type has.

Procedure

g-object-info-get-vfunc info n
Returns a pointer to the nth GIVfuncInfo of info.
It must be freed by calling [g-base-info-unref], page 81, when done accessing the data.

Procedure

g-object-info-get-class-struct info
Returns a pointer to the n-th GIStructInfo of info, or #f.

Procedure

Every GObject has two structures: an instance structure and a class structure. This function returns a pointer to the info class structure.

It must be freed by calling [g-base-info-unref], page 81, when done accessing the data.
Interface Info

G-Golf Interface Info low level API.
GIInterfaceInfo — Structs representing a GInterface.

Procedures

[gi-interface-import], page 98
[gi-interface-show], page 98
[g-interface-info-get-n-prerequisites], page 100
[g-interface-info-get-prerequisite], page 100
[g-interface-info-get-n-properties], page 100
[g-interface-info-get-property], page 100
[g-interface-info-get-n-methods], page 100
[g-interface-info-get-method], page 100
[g-interface-info-find-method], page 100
[g-interface-info-get-n-signals], page 100
[g-interface-info-get-signal], page 100
[g-interface-info-find-signal], page 101
[g-interface-info-get-n-vfuncs], page 101
[g-interface-info-get-vfunc], page 101
[g-interface-info-find-vfunc], page 101
[g-interface-info-get-n-constants], page 101
[g-interface-info-get-constant], page 101
[g-interface-info-find-iface-struct], page 101

Description

A GInterface has methods, properties, signals, constants, virtual functions and prerequisites.

Struct Hierarchy

GIBaseInfo
  +— GIRegisteredTypeInfo
    +— GIInterfaceInfo

Procedures

Note: in this section, unless otherwise specified, the info argument is [must be] a pointer to a GIInterfaceInfo.

\textit{gi-interface-import} \textit{info} \hspace{1cm} \text{[Procedure]}

Returns a list.

In the current version of G-Golf, interfaces are `opques'. Returns a list composed of the 'interface (type-tag) symbol, the interface (scheme and symbol) name, g-name, g-type and \#t (a boolean that means the type is confirmed). Here is an example:

\begin{verbatim}
(interface gtk-orientable "GtkOrientable" 94578771473520 \#t)
\end{verbatim}

\textit{gi-interface-show} \textit{info} \hspace{1cm} \text{[Procedure]}

Returns nothing.
Obtains and displays the following informations about the interface pointed to by info:

```
,use (g-golf)
  (g-irepository-require "Gdk" #:version "4.0")
⇒ $2 = #<pointer 0x55649014c780>

(g-irepository-find-by-name "Gdk" "Paintable")
⇒ $3 = #<pointer 0x5564901531e0>

(gi-interface-show $3)
⊢ #<pointer 0x5564901531e0> is a (pointer to a) GIInterfaceInfo:
  namespace: "Gdk"
  name: "Paintable"
  g-type: 93947637686432
  g-type-name: "GdkPaintable"
  n-prerequisites: 0
  n-properties: 0
  n-methods: 10
  n-signals: 2
  n-vfuncs: 6
  n-constants: 0
  iface-struct: #<pointer 0x5571e38ec190>
  iface-struct-name: "PaintableInterface"

Methods:
  0. #f
     gdk-paintable-new-empty
  1. compute-concrete-size
     gdk-paintable-compute-concrete-size
  2. get-current-image
     gdk-paintable-get-current-image
  3. get-flags
     gdk-paintable-get-flags
     ...

VFuncs:
  0. get-current-image
  1. get-flags
```
III. G-Golf Core Reference

---

2. get-intrinsic-aspect-ratio
---

3. get-intrinsic-height
---

4. get-intrinsic-width
---

5. snapshot

**g-interface-info-get-n-prerequisites** info

[Procedure]

Returns the number of prerequisites for info.

Obtain the number of prerequisites for this interface type. A prerequisite is another interface that needs to be implemented for interface, similar to a base class for GObjects.

**g-interface-info-get-prerequisite** info n

[Procedure]

Returns a pointer to the n\textsuperscript{th} prerequisite for info.

The prerequisite as a GIBaseInfo. It must be freed by calling [g-base-info-unref], page 81, when done accessing the data.

**g-interface-info-get-n-properties** info

[Procedure]

Returns the number of properties for info.

Obtain the number of properties that this interface type has.

**g-interface-info-get-property** info n

[Procedure]

Returns a pointer to the n\textsuperscript{th} GIPropertyInfo of info.

It must be freed by calling [g-base-info-unref], page 81, when done accessing the data.

**g-interface-info-get-n-methods** info

[Procedure]

Returns the number of methods for info.

Obtain the number of methods that this interface type has.

**g-interface-info-get-method** info n

[Procedure]

Returns a pointer to the n\textsuperscript{th} GIFunctionInfo of info.

It must be freed by calling [g-base-info-unref], page 81, when done accessing the data.

**g-interface-info-find-method** info name

[Procedure]

Returns a pointer to a GIFunctionInfo or #f if there is no method available with that name.

It must be freed by calling [g-base-info-unref], page 81, when done accessing the data.

**g-interface-info-get-n-signals** info

[Procedure]

Returns the number of signals for info.

Obtain the number of signals that this interface type has.

**g-interface-info-get-signal** info n

[Procedure]

Returns a pointer to the n\textsuperscript{th} GISignalInfo of info.

It must be freed by calling [g-base-info-unref], page 81, when done accessing the data.
g-interface-info-find-signal info name

Returns a pointer to a GISignalInfo or #f if there is no signal available with that name.

It must be freed by calling [g-base-info-unref], page 81, when done accessing the data.

[g(interface-info-get-n-vfuncs) info]

Returns the number of vfuncs for info.

Obtain the number of vfuncs that this interface type has.

[g(interface-info-get-vfunc info n)]

Returns a pointer to the nth GIVfuncInfo of info.

It must be freed by calling [g-base-info-unref], page 81, when done accessing the data.

[g(interface-info-find-vfunc info name)]

Returns a pointer to a GIFunctionInfo or #f if there is no signal available with that name.

It must be freed by calling [g-base-info-unref], page 81, when done accessing the data.

[g(interface-info-get-n-constants info)]

Returns the number of constants for info.

Obtain the number of constants that this interface type has.

[g(interface-info-get-constant info n)]

Returns a pointer to the nth GIConstantInfo of info.

It must be freed by calling [g-base-info-unref], page 81, when done accessing the data.

[g(interface-info-get-iface-struct info)]

Returns a pointer to a GIStructInfo for info, or #f.

Obtains and returns the layout C structure associated with info. It must be freed by calling [g-base-info-unref], page 81, when done accessing the data.

Arg Info

G-Golf Arg Info low level API.

GIArgInfo — Struct representing an argument.

Procedures

[g(arg-info-get-closure)], page 102
[g(arg-info-get-destroy)], page 102
[g(arg-info-get-direction)], page 102
[g(arg-info-get-ownership-transfer)], page 102
[g(arg-info-get-scope)], page 102
[g(arg-info-get-type)], page 102
[g(arg-info-may-be-null)], page 103
[g(arg-info-is-caller-allocates)], page 103
[g(arg-info-is-optional)], page 103
[g(arg-info-is-return-value)], page 103
[g(arg-info-is-skip)], page 103
Types and Values

- [%gi-direction], page 103
- [%gi-scope-type], page 103
- [%gi-transfer], page 104

Struct Hierarchy

GIBaseInfo
   +— GIArgInfo

Description

GIArgInfo represents an argument. An argument is always part of a GICallableInfo.

Procedures

Note: in this section, the info argument is [must be] a pointer to a GIArgInfo.

**g-arg-info-get-closure info** [Procedure]

Returns the index of the user data argument or -1 if there is none.

Obtains the index of the user data argument. This is only valid for arguments which are callbacks.

**g-arg-info-get-destroy info** [Procedure]

Returns the index of the GDestroyNotify argument or -1 if there is none.

Obtains the index of the GDestroyNotify argument. This is only valid for arguments which are callbacks.

**g-arg-info-get-direction info** [Procedure]

Returns a symbol.

Obtains and returns the [%gi-direction], page 103, of the argument.

**g-arg-info-get-ownership-transfer info** [Procedure]

Returns a symbol.

Obtains and returns the [%gi-transfer], page 104, for this argument.

**g-arg-info-get-scope info** [Procedure]

Returns a symbol.

Obtains and returns the [%gi-scope-type], page 103, for this argument. The scope type explains how a callback is going to be invoked, most importantly when the resources required to invoke it can be freed.

**g-arg-info-get-type info** [Procedure]

Returns a pointer.

Obtains the GITypeInfo holding the type information for info. Free it using [g-base-info-unref], page 81, when done.


### Procedure

**g-arg-info-may-be-null info**

[Procedure]

Returns #t or #f.

Obtains if the type of the argument includes the possibility of NULL. For 'in' values this means that NULL is a valid value. For 'out' values, this means that NULL may be returned.

**g-arg-info-is-caller-allocates info**

[Procedure]

Returns #t or #f.

Obtain if the argument is a pointer to a struct or object that will receive an output of a function. The default assumption for out arguments which have allocation is that the callee allocates; if this is TRUE, then the caller must allocate.

**g-arg-info-is-optional info**

[Procedure]

Returns #t or #f.

Obtains if the argument is optional. For 'out' arguments this means that you can pass NULL in order to ignore the result.

**g-arg-info-is-return-value info**

[Procedure]

Returns #t or #f.

Obtains if the argument is a return value. It can either be a parameter or a return value.

**g-arg-info-is-skip info**

[Procedure]

Returns #t or #f.

Obtains if an argument is only useful in C.

### Types and Values

**%gi-direction**

[Instance Variable of <gi-enum>]

An instance of <gi-enum>, who's members are the scheme representation of the direction of a GIArgInfo:

- **g-name**: GIDirection
- **name**: gi-direction
- **enum-set**:
  - in in argument.
  - out out argument.
  - inout in and out argument.

**%gi-scope-type**

[Instance Variable of <gi-enum>]

An instance of <gi-enum>, who’s members are the scheme representation of the scope of a GIArgInfo. Scope type of a GIArgInfo representing callback, determines how the callback is invoked and is used to decide when the invoke structs can be freed.

- **g-name**: GIScopeType
- **name**: gi-scope-type
- **enum-set**:
  - invalid The argument is not of callback type.
call

The callback and associated user_data is only used during the call to this function.

async

The callback and associated user_data is only used until the callback is invoked, and the callback is invoked always exactly once.

notified

The callback and associated user_data is used until the caller is notified via the destroy notify.

%gi-transfer

[Instance Variable of <gi-enum>]

The transfer is the exchange of data between two parts, from the callee to the caller. The callee is either a function/method/signal or an object/interface where a property is defined. The caller is the side accessing a property or calling a function. GITransfer specifies who’s responsible for freeing the resources after the ownership transfer is complete. In case of a containing type such as a list, an array or a hash table the container itself is specified differently from the items within the container itself. Each container is freed differently, check the documentation for the types themselves for information on how to free them.

An instance of <gi-enum>, who’s members are the scheme representation of the GITransfer:

  g-name: GITransfer
  name: gi-transfer
  enum-set:

  nothing

transfer nothing from the callee (function or the type instance the property belongs to) to the caller. The callee retains the ownership of the transfer and the caller doesn’t need to do anything to free up the resources of this transfer

container

transfer the container (list, array, hash table) from the callee to the caller. The callee retains the ownership of the individual items in the container and the caller has to free up the container resources g_list_free, g_hash_table_destroy, ... of this transfer

everything

transfer everything, eg the container and its contents from the callee to the caller. This is the case when the callee creates a copy of all the data it returns. The caller is responsible for cleaning up the container and item resources of this transfer

Constant Info

G-Golf Constant Info low level API.

GIConstantInfo — Struct representing a constant.
Procedures

- \[g\text{-constant\text{-info\text{-free\text{-value}}}],\text{ page 105}\]
- \[g\text{-constant\text{-info\text{-get\text{-type}}}],\text{ page 105}\]
- \[g\text{-constant\text{-info\text{-get\text{-value}}}],\text{ page 105}\]

Struct Hierarchy

GIBaseInfo
  +— GIConstantInfo

Description

GIConstantInfo represents a constant. A constant has a type associated which can be obtained by calling [g-constant-info-get-type], page 105, and a value, which can be obtained by calling [g-constant-info-get-value], page 105.

Procedures

Note: in this section, the info and value arguments are [must be] pointers to a GIConstantInfo and a GIArgument, respectively.

\[g\text{-constant\text{-info\text{-free\text{-value}}}]\text{ info value}\]
  [Procedure]

  Returns nothing.
  Frees the value returned from [g-constant-info-get-value], page 105.

\[g\text{-constant\text{-info\text{-get\text{-type}}}]\text{ info}\]
  [Procedure]

  Returns a pointer.
  Obtains and returns a pointer to the GITypeInfo for info. Free it using [g-base-info-unref], page 81, when done.

\[g\text{-constant\text{-info\text{-get\text{-value}}}]\text{ info value}\]
  [Procedure]

  Returns an integer (the size of a constant).
  Obtains the value associated with info and store it in the value parameter, which must be allocated before passing it.
  The size of the constant value stored in argument will be returned. Free the value argument with [g-constant-info-free-value], page 105.

Field Info

G-Golf Field Info low level API.
GIFieldInfo — Struct representing a struct or union field.

Procedures

- \[g\text{-field\text{-info\text{-get\text{-flags}}}],\text{ page 106}\]
- \[g\text{-field\text{-info\text{-get\text{-offset}}}],\text{ page 106}\]
- \[g\text{-field\text{-info\text{-get\text{-type}}}],\text{ page 106}\]
Struct Hierarchy

GIBaseInfo
  +— GIBaseInfo

Description

A GIFieldInfo struct represents a field of a struct (see [Struct Info], page 90), union (see GIUnionInfo) or an object (see [Object Info], page 93). The GIFieldInfo is fetched by calling [g-struct-info-get-field], page 91, [g-union-info-get-field] or [g-object-info-get-field], page 96. A field has a size, type and a struct offset associated and a set of flags, which are currently readable or writable.

Procedures

Note: in this section, unless otherwise specified, the info argument is [must be] a pointer to a GIFieldInfo.

**g-field-info-get-flags info**

[Procedure]

Returns a (possibly empty) list.

Obtains and returns the flags for info, which currently are readable or writable.

**g-field-info-get-offset info**

[Procedure]

Returns an unsigned integer.

Obtains and returns the offset in bytes for info, the field member, this is relative to the beginning of the struct or union.

**g-field-info-get-type info**

[Procedure]

Returns a pointer.

Obtains and returns the GITypeInfo for info.

The GITypeInfo must be freed by calling [g-base-info-unref], page 81, when done.

Property Info

G-Golf Property Info low level API.
GIBaseInfoInfo — Struct representing a property.

Procedures

[gi-property-g-type], page 107

[g-property-info-get-flags], page 107

[g-property-info-get-ownership-transfer], page 107

[g-property-info-get-type], page 107

Struct Hierarchy

GIBaseInfoInfo
  +— GIBaseInfoInfo

Description

GIBaseInfoInfo represents a property. A property belongs to either a GIOBJECTINFO or a GIINTERFACEINFO.
Procedures
Note: in this section, the info argument is [must be] a pointer to a GIPropertyInfo.

**gi-property-g-type info**  
Returns an integer.  
Obtains and returns the GType value of the property.

**g-property-info-get-flags info**  
Returns a list of [%g-param-flags], page 68.  
Obtain the flags for this property info. See [GParamSpec], page 66, for the list of possible flag values.

**g-property-info-get-ownership-transfer info**  
Returns the ownership transfer for this property.  
Obtain the ownership transfer for this property. See [%gi-transfer], page 104, for more information about transfer values.

**g-property-infoxs-get-type info**  
Returns a pointer to a GITypeInfo.  
Obtain the type information for this property. The GITypeInfo must be free’d using g-base-info-unref when done.

Type Info
G-Golf Type Info low level API.
GITypeInfo — Struct representing a type.

Procedures

[g-info-type-to-string], page 108  
[g-type-info-is-pointer], page 108  
[g-type-info-get-tag], page 108  
[g-type-info-get-param-type], page 108  
[g-type-info-get-interface], page 108  
[g-type-info-get-array-length], page 108  
[g-type-info-get-array-fixed-size], page 108  
[g-type-info-is-zero-terminated], page 109  
[g-type-info-get-array-type], page 109

Struct Hierarchy

GITypeInfo  
+— GITypeInfo

Description
GITypeInfo represents a type. You can retrieve a type info from an argument (see [Arg Info], page 101), a functions return value (see [Function Info], page 84), a field (see GIBaseInfo), a property (see [Property Info], page 106), a constant (see GIClassInfo) or for a union discriminator (see GIBaseInfo).
A type can either be a of a basic type which is a standard C primitive type or an interface type. For interface types you need to call `g-type-info-get-interface` to get a reference to the base info for that interface.

## Procedures

Note: in this section, the `info` argument is [must be] a pointer to a `GTypeInfo`.

**g-info-type-to-string** `info-type`  
Returns a string or `#f`.  
Obtains the string representation for `info-type` or `#f` if it does not exists.  
`info-type` can either be a `symbol` or an `id`, a member of the `enum-set` of `[%gi-info-type]`, page 82, (otherwise, `#f` is returned).

**g-type-info-is-pointer** `info`  
Returns `#t` or `#f`.  
Obtains if the `info` type is passed as a reference.  
Note that the types of `out` and `inout` parameters (see `[%gi-direction]`, page 103) will only be pointers if the underlying type being transferred is a pointer (i.e. only if the type of the C function’s formal parameter is a pointer to a pointer).

**g-type-info-get-tag** `info`  
Returns a symbol.  
Obtains the type tag for `info` (see `[%gi-type-tag]`, page 78, for the list of type tags).

**g-type-info-get-param-type** `info` `n`  
Returns a pointer or `#f`.  
Obtains the parameter type `n` (the index of the parameter). When there is no such `n` parameter, the procedure returns `#f`.

**g-type-info-get-interface** `info`  
Returns a pointer or `#f`.  
For `interface` types (see `[%gi-type-tag]`, page 78) such as GObjects and boxed values, this procedure returns a (pointer to a) `GIBaseInfo`, holding full information about the referenced type. You can then inspect the type of the returned `GIBaseInfo` to further query whether it is a concrete GObject, a GInterface, a structure, etc. using `[g-base-info-get-type]`, page 81.

**g-type-info-get-array-length** `info`  
Returns an integer.  
Obtain the array length of the type. The type tag must be a `array` (see `[%gi-type-tag]`, page 78), or `1` will returned.

**g-type-info-get-array-fixed-size** `info`  
Returns an integer.  
Obtain the fixed array size of the type. The type tag must be a `array` (see `[%gi-type-tag]`, page 78), or `-1` will returned.
**g-type-info-is-zero-terminated info**  
Returns #t or #f.  
Obtains if the last element of the array is NULL. The type tag must be a array (see [%gi-type-tag], page 78), or #f will be returned.

**g-type-info-get-array-type info**  
Returns a symbol or #f.  
Obtain the array type for this type (see [%gi-array-type], page 79). If the type tag of this type is not array, #f will be returned.

### FFI Interface

G-Golf FFI Interface low level API.  
girffi — TODO.

#### Procedures

- [gi-type-tag-get-ffi-type], page 109  
- [g-type-info-get-ffi-type], page 109  
- [gi-type-info-extract-ffi-return-value], page 109  
- [g-type-tag-extract-ffi-return-value], page 110  
- [g-callable-info-prepare-closure], page 110

#### Description

TODO.

#### Procedures

**gi-type-tag-get-ffi-type type-tag is-pointer?**  
Returns a (pointer to) ffi-type corresponding to the platform default C ABI for type-tag and is-pointer?.  
The info argument is (must be) a valid [%gi-type-tag], page 78, otherwise an exception is raised.  
The is-pointer? argument, #t or #f, to indicate whether or not this is a pointer type.

**g-type-info-get-ffi-type info**  
Returns an (pointer to) ffi-type corresponding to the platform default C ABI for info.  
The info argument is [must be] a pointer to a GITypeInfo.

**gi-type-info-extract-ffi-return-value type-info ffi-value**  
Returns nothing.  
Extract the correct bits from ffi-value into gi-argument.  
The type-info is the GITypeInfo of ffi-value. The ffi-value is a pointer to a GIFFIFIReturnValue union containing the value from the ffi_call(). The gi-argument is a pointer to an allocated GIArgument.
**gi-type-tag-extract-ffi-return-value**  
`return-tag`  
`interface-type`  
`ffi-value`  
`gi-argument`  

Returns nothing.

Extract the correct bits from `ffi-value` into `gi-argument`.

The `return-tag` is the [%gi-type-tag], page 78, of `ffi-value`. The `interface-type` is the [%gi-info-type], page 82, of the underlying interface. The `ffi-value` is a pointer to a `GIFFIReturnValue` union containing the value from the `ffi_call()`. The `gi-argument` is a pointer to an allocated `GIArgument`.

The `interface-type` argument only applies if `return-tag` is `interface`, otherwise it is ignored.

**g-callable-info-prepare-closure**  
`info`  
`ffi-cif`  
`ffi-closure-callback`  
`user-data`  

Returns the native address of the closure or `#f` on error.

The procedure has been deprecated since version 1.72 and should not be used in newly-written code. Use
[g-callable-info-create-closure], page 84, instead.

The return value should be freed by calling `g-callable-info-free-closure`.

**Utilities**

G-Golf GObject Introspection Utilities low level API.
Procedures and Syntax

- [gi-pointer-new], page 111
- [gi-pointer-inc], page 111
- [gi-attribute-iter-new], page 111
- [with-gerror], page 112
- [gi->scm], page 112
- [gi-boolean->scm], page 112
- [gi-string->scm], page 112
- [gi-n-string->scm], page 112
- [gi-strings->scm], page 113
- [gi-csv-string->scm], page 113
- [gi-pointer->scm], page 112
- [gi-n-pointer->scm], page 112
- [gi-pointers->scm], page 113
- [gi-n-gtype->scm], page 112
- [gi-glist->scm], page 113
- [gi-gslist->scm], page 113
- [scm->gi], page 113
- [scm->gi-boolean], page 113
- [scm->gi-string], page 113
- [scm->gi-n-string], page 114
- [scm->gi-strings], page 114
- [scm->gi-pointer], page 113
- [scm->gi-n-pointer], page 114
- [scm->gi-pointers], page 114
- [scm->gi-n-gtype], page 114
- [scm->gi-gslist], page 114

Types and Values

- [%gi-pointer-size], page 114

Description

G-Golf GObject Introspection utilities low level API.

Procedures and Syntax

**gi-pointer-new**

Returns a newly allocated (Glib) pointer.

**gi-pointer-inc** *pointer* [*#:offset %gi-pointer-size*]

Returns a foreign pointer object pointing to the address of *pointer* increased by *offset*.

**gi-attribute-iter-new**

Returns a pointer.

Creates and returns a foreign pointer to a C struct for a GIAtrributeIter (a C struct containing four pointers, initialized to %null-pointer).
with-gerror var body

Returns the result of the execution of body, or raises an exception.

var must be an identifier. Evaluate body in a lexical environment where var is bound to a pointer to a newly allocated (and 'empty') GError. var will always be freed. If no exception is raised, the result of the execution of body is returned.

**gi->scm** value type [cmpl #f]

Returns the scheme representation of value.

The type, a symbol name (also called a type tag or just a tag in the GI terminology) supported values are:

- **'boolean** Calls [gi-boolean->scm], page 112.
- **'string**
- **'pointer** Calls [gi-string->scm], page 112, or [gi-pointer->scm], page 112.
- **'n-string**
- **'n-pointer**
- **'n-gtype** Calls [gi-n-string->scm], page 112, [gi-n-pointer->scm], page 112, or [gi-n-gtype->scm], page 112.

The optional cmpl (complement) argument must be passed and set to the number of string(s), pointer(s) or gtype(s) contained in value, .

- **'strings**
- **'pointers**

Calls [gi-strings->scm], page 113, or [gi-pointers->scm], page 113.

- **'csv-string**

Calls [gi-csv-string->scm], page 113.

- **'glist**
- **'gslist**

Calls [gi-glist->scm], page 113, or [gi-gslist->scm], page 113, repsec-
tively.

**gi-boolean->scm** value

Returns #t or #f.

The GType of value must be a gboolean.

**gi-string->scm** value

**gi-pointer->scm** value

Returns a string, a pointer or #f if value is the %null-pointer.

The GType of value must be a gchar* or a gpointer.

**gi-n-string->scm** value n-string

**gi-n-pointer->scm** value n-pointer

**gi-n-gtype->scm** value n-gtype

Returns a (possibly empty list) of string(s), pointer(s) or GType(s).

The GType of value must be a gchar**, a gpointer[] or a GType[]. The n-string, n-pointer and n-gtype argument must be the length of the value array.
\[
\text{gi-strings->scm value} \quad \text{[Procedure]}
\]
\[
\text{gi-pointers->scm value} \quad \text{[Procedure]}
\]

Returns a (possibly empty) list of strings or pointer.

The GType of value must be a \texttt{gchar**} or \texttt{gpointer[]} . The array must be \texttt{NULL} terminated.

\[
\text{gi-csv-string->scm value} \quad \text{[Procedure]}
\]

Returns a list of string(s) or \texttt{#f} if value is the \%null-pointer .

The GType of value is \texttt{gchar*} . Unless \texttt{#f} , the list of string(s) is obtained by splitting the (comma separated value) string pointed to by value using using \texttt{\#, as the char-pred.}

\[
\text{gi-glist->scm g-list} \quad \text{[Procedure]}
\]
\[
\text{gi-gslist->scm g-slist} \quad \text{[Procedure]}
\]

Returns a (possibly empty) list.

Obtains and returns a (possibly empty) list of the pointers stored in the \texttt{data} field of each element of \texttt{g-list} or \texttt{g-slist}.

\[
\text{scm->gi value type [cmpl #f]} \quad \text{[Procedure]}
\]

Returns the GI representation of value.

The \texttt{type} , a symbol name (also called a \texttt{type tag} or just a \texttt{tag} in the GI terminology) supported values are:

- \texttt{'boolean} Calls \texttt{scm->gi-boolean} , page 113.
- \texttt{'string} Calls \texttt{scm->gi-string} , page 113, or \texttt{scm->gi-pointer} , page 113.
- \texttt{'n-string} Calls \texttt{scm->gi-n-string} , page 114, \texttt{scm->gi-n-pointer} , page 114, or \texttt{scm->gi-n-gtype} , page 114.
- \texttt{'n-gtype} Calls \texttt{scm->gi-n-string} , page 114, \texttt{scm->gi-n-pointer} , page 114, or \texttt{scm->gi-n-gtype} , page 114.

The optional \texttt{cmpl} (complement) argument may be passed and set to the number of string(s), pointer(s) or gtype(s) contained in value.

- \texttt{'strings} Calls \texttt{scm->gi-strings} , page 114, or \texttt{scm->gi-pointers} , page 114.
- \texttt{'pointers} Calls \texttt{scm->gi-strings} , page 114, or \texttt{scm->gi-pointers} , page 114.
- \texttt{'gslist} Calls \texttt{scm->gi-gslist} , page 114.

\[
\text{scm->gi-boolean value} \quad \text{[Procedure]}
\]

Returns 0 if value is \texttt{#f} , otherwise, it returns 1.

\[
\text{scm->gi-string value} \quad \text{[Procedure]}
\]
\[
\text{scm->gi-pointer value} \quad \text{[Procedure]}
\]

Returns a pointer.

If value is \texttt{#f} , it returns \texttt{%null-pointer} . Otherwise, it returns a pointer to the string in value or value.
### scm->gi-n-string value [n-string #f]

[Procedure]

Returns two values.

- If `value` is the empty list, it returns `%null-pointer` and an empty list. Otherwise, it returns a pointer to an array of pointer(s) to the string(s) in `value` and a list of the 'inner' string pointer(s).

- It is the caller’s responsibility to maintain a reference to those inner pointer(s), until the array 'itself' (the first returned value) is no longer needed/used.

- The array returned by `[scm->gi-strings]`, page 114, is NULL terminated, whereas the array returned by `[scm->gi-n-string]`, page 114, is not.

### scm->gi-n-pointer value [n-pointer #f]

[Procedure]

Returns a pointer.

- If `value` is an empty list, it returns `%null-pointer`. Otherwise, it returns a pointer to an array the pointer(s) or GType(s) in `value`.

- The returned array is not NULL nor 0- terminated.

### scm->gi-n-gtype value [n-gtype #f]

[Procedure]

Returns a pointer.

- If `value` is an empty list, it returns `%null-pointer`. Otherwise, it returns a pointer to an array the pointer(s) or GType(s) in `value`.

### Types and Values

#### `%gi-pointer-size`

[Variable]

The size (the number of bytes) that a (Glib) pointer occupies in memory (which is architecture dependent).

### Support

G-Golf uses a series of support modules, each documented in the following subsections. You may either import them all, like this `(use-modules (g-golf support))`, or individually, such as `(use-modules (g-golf support modules))`, `(use-modules (g-golf support goops))`, ...
Syntax

\texttt{re-export-public-interface\ mod1\ mod2\ .}\ldots[\text{Syntax}]

Re-export the public interface of a \texttt{mod1\ mod2\ .}\ldots

Invoked like \texttt{use-modules}, where each \texttt{mod1\ mod2\ .}\ldots is a module name (a list of symbol(s)).

Goops

Syntax, Procedures and Methods

\texttt{class-direct-virtual-slots\ (self\ <class>)}\ [Method]

Returns a list.

Obtains and returns the list of the class direct slots for \texttt{self} that satisfy the \texttt{(eq? (slot-definition-allocation slot) #:virtual)} predicate.

\texttt{class-virtual-slots\ (self\ <class>)}\ [Method]

Returns a list.

Obtains and returns the list of the class slots for \texttt{self} that satisfy the \texttt{(eq? (slot-definition-allocation slot) #:virtual)} predicate.

\texttt{class-direct-g-property-slots\ (self\ <class>)}\ [Method]

Returns a list.

Obtains and returns the list of the class direct slots for \texttt{self} that satisfy the \texttt{(eq? (slot-definition-allocation slot) #:g-property)} predicate.

\texttt{class-g-property-slots\ (self\ <class>)}\ [Method]

Returns a list.

Obtains and returns the list of the class slots for \texttt{self} that satisfy the \texttt{(eq? (slot-definition-allocation slot) #:g-property)} predicate.

\texttt{mslot-set!\ inst\ s1\ v1\ s2\ v2\ s3\ v3\ .}\ldots [Procedure]

Returns nothing.

Performs a multiple \texttt{slot-set!} for \texttt{inst}, setting its slot named \texttt{s1} to the value \texttt{v1}, \texttt{s2} to \texttt{v2}, \texttt{s3} to \texttt{v3}\ldots

\texttt{generic?\ value}\ [Procedure]

Returns \texttt{#t} if \texttt{value} is a \texttt{<generic>} instance. Otherwise, it returns \texttt{#f}. 
Enum
G-Golf class, accessors, methods and procedures to deal with C enum types.

Classes

[<enum>], page 116
[<gi-enum>], page 116

Procedures, Accessors and Methods

[!enum-set], page 117
[enum->value], page 117
[enum->values], page 117
[enum->symbol], page 117
[enum->symbols], page 117
[enum->name], page 117
[enum->names], page 117
[!g-type_], page 117
[!g-name], page 117
[!name__], page 117

Description
G-Golf class, accessors, methods and procedures to deal with C enum types.

Classes

<enum> [Class]

The <enum> class is for enumerated values. Its (unique) slot is:

    enum-set #:accessor !enum-set
               #:init-keyword #:enum-set

Notes:

• the enum-set can’t be empty and so you must use the #:enum-set (#:init-keyword) when creating new <enum> instances;
• the #:enum-set (#:init-keyword) accepts either a list of symbols or a well-formed enum-set;
• a well-formed enum-set is a list of (symbol . id) pairs, where id is a positive integer.
• each symbol and each id of an enum-set must be unique.

Instances of the <enum> class are immutable (to be precise, there are not meant to be mutated, see [GOOPS Notes and Conventions], page 7, ’Slots are not Immutable’).

<gi-enum> [Class]

The <gi-enum> class is a subclass of <enum>. Its class-direct-slots are:

    g-type #:accessor !g-type
           #:init-keyword #:g-type
           #:init-value #f
The name slot is automatically initialized.

Instances of the `<gi-enum>` class are immutable (to be precise, there are not meant to be mutated, see [GOOPS Notes and Conventions], page 7, 'Slots are not Immutable').

Procedures, Accessors and Methods

`!enum-set (inst <enum>)`  
Returns the content of the enum-set slot for `inst`.

`enum->value (inst <enum>) symbol`  
Method

`enum->values (inst <enum>)`  
Method

Returns the `inst` value for `symbol` (or `#f` if it does not exists), or the list of all values for `inst`, respectively.

`enum->symbol (inst <enum>) value`  
Method

`enum->symbols (inst <enum>)`  
Method

Returns the `inst` symbol for `value` (or `#f` if it does not exists), or the list of all symbols for `inst`, respectively.

`enum->name (inst <enum>) value`  
Method

`enum->names (inst <enum>)`  
Method

Returns the `inst` name (the string representation of the symbol) for `value` (or `#f` if it does not exists), or the list of all names for `inst`, respectively.

value can either be a symbol or an id.

`!g-type (inst <gi-enum>)`  
Accessor

`!g-name (inst <gi-enum>)`  
Accessor

`!name (inst <gi-enum>)`  
Accessor

Returns the content of the g-type, g-name or name slot for `inst`, respectively.

Flags

G-Golf class, accessors, methods and procedures to deal with C flags types.

Classes

`[<flags>], page 118`
`[<gi-flags>], page 118`

Procedures, Accessors and Methods

`[integer->flags], page 118`
`[flags->integer], page 118`
`[!g-type___], page 118`
`[!g-name______], page 118`
`[!name____], page 118`
III. G-Golf Core Reference

Description
G-Golf class, accessors, methods and procedures to deal with C flags types.

Classes

<flags> [Class]
The <flags> class is a subclass of [enum], page 116. It has no direct slots.

<gi-flags> [Class]
The <gi-flags> class is a subclass of <flags>. Its class-direct-slots are:

- g-type #:accessor !g-type
  #:init-keyword #:g-type
  #:init-value #f

- g-name #:accessor !g-name
  #:init-keyword #:g-name

- name #:accessor !name

The name slot is automatically initialized.

Instances of the <gi-flags> class are immutable (to be precise, there are not meant to be mutated, see [GOOPS Notes and Conventions], page 7, 'Slots are not Immutable').

Procedures, Accessors and Methods

integer->flags (inst <flags>) n
[Method]
Returns a possibly empty) list of symbol(s).
Obtains and returns the list of (symbol) flags for the given <flags> instance and its integer representation n.

flags->integer (inst <flags>) flags
[Method]
Returns an integer.
Compute and returns the integer representation for the list of (symbol(s)) given by flags and the given <flag> instance.

!g-type (inst <gi-flags>)
[Accessor]
!g-name (inst <gi-flags>)
[Accessor]
!name (inst <gi-flags>)
[Accessor]
Returns the content of the g-type, g-name or name slot for inst, respectively.

Struct
G-Golf class, accessors, methods and procedures to deal with C struct types.

Classes

[<gi-struct>], page 119
Procedures and Accessors

[!g-name_], page 120
[!name____], page 120
[!alignment], page 120
[!size], page 120
[!is-gtype-struct?], page 120
[!is-foreign?], page 120
[!field-types], page 120
[!scm-types], page 120
[!init-vals], page 120
[!is-opaque?], page 120
[!is-semi-opaque?], page 120

Description
G-Golf class, accessors, methods and procedures to deal with C struct types.

Classes

<gi-struct>  
The <gi-struct> class is a subclass of <struct>. Its class-direct-slots are:

g-name  #:accessor !g-name
        #:init-keyword #:g-name
name    #:accessor !name
alignment #:accessor !alignment
          #:init-keyword #:alignment
size     #:accessor !size
          #:init-keyword #:size
is-gtype-struct?
          #:accessor !is-gtype-struct?
          #:init-keyword #:is-gtype-struct?
field-types
          #:accessor !field-types
          #:init-keyword #:field-types
scm-types
          #:accessor !scm-types
init-vals
          #:accessor !init-vals
is-opaque?
          #:accessor !is-opaque?
is-semi-opaque
          #:accessor !is-semi-opaque?
The `name` and `scm-types` slots are automatically initialized.

Instances of the `<gi-struct>` are immutable (to be precise, there are not meant to be mutated, see [GOOPS Notes and Conventions], page 7, 'Slots are not Immutable').

### Procedures and Accessors

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>!g-name (inst &lt;gi-struct&gt;)</code></td>
<td>[Accessor]</td>
</tr>
<tr>
<td><code>!name (inst &lt;gi-struct&gt;)</code></td>
<td>[Accessor]</td>
</tr>
<tr>
<td><code>!alignment (inst &lt;gi-struct&gt;)</code></td>
<td>[Accessor]</td>
</tr>
<tr>
<td><code>!size (inst &lt;gi-struct&gt;)</code></td>
<td>[Accessor]</td>
</tr>
<tr>
<td><code>!is-gtype-struct? (inst &lt;gi-struct&gt;)</code></td>
<td>[Accessor]</td>
</tr>
<tr>
<td><code>!field-types (inst &lt;gi-struct&gt;)</code></td>
<td>[Accessor]</td>
</tr>
<tr>
<td><code>!scm-types (inst &lt;gi-struct&gt;)</code></td>
<td>[Accessor]</td>
</tr>
<tr>
<td><code>!init-vals (inst &lt;gi-struct&gt;)</code></td>
<td>[Accessor]</td>
</tr>
</tbody>
</table>

- Returns the content of their respective slot for `inst`.

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>!is-opaque? (inst &lt;gi-struct&gt;)</code></td>
<td>[Accessor]</td>
</tr>
</tbody>
</table>

- Returns `#t` if `inst` is 'opaque', otherwise, it returns `#f`.

A `<gi-struct>` instance is said to be 'opaque' when the call to `g-struct-info-get-size` upon its GIStructInfo pointer returns zero. In scheme, these `<gi-struct>` instances have no fields.

- 'Opaque' boxed types should never be 'decoded', nor 'encoded'. Instead, procedures, accessors and methods should 'blindly' receive, pass and/or return their pointer(s).

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>!is-semi-opaque? (inst &lt;gi-struct&gt;)</code></td>
<td>[Accessor]</td>
</tr>
</tbody>
</table>

- Returns `#t` if `inst` is 'semi-opaque', otherwise, it returns `#f`.

A `<gi-struct>` instance is said to be 'semi-opaque' when one of its field types is `void`.

- 'Semi-opaque' boxed types should never be 'decoded', nor 'encoded'. Instead, procedures, accessors and methods should 'blindly' receive, pass and/or return their pointer(s).

### Union

G-Golf class, accessors, methods and procedures to deal with C union types.

### Classes

[<gi-union>], page 121
Procedures, Accessors and Methods

[make-c-union], page 122
[c-union-ref], page 122
[c-union-set!], page 122
[!g-type__], page 122
[!g-name__], page 122
[!name___], page 122
[!size_], page 122
[!alignment_], page 122
[!fields], page 122
[!is-discriminated?], page 122
[!discriminator-offset], page 122
[!discriminator], page 122

Description

G-Golf class, accessors, methods and procedures to deal with C union types.

Classes

<gi-union>  
The <gi-union> class. Its class-direct-slots are:

  g-type #:accessor !g-type
  #:init-keyword #:g-type

  g-name #:accessor !g-name
  #:init-keyword #:g-name

  name #:accessor !name

  size #:accessor !size
  #:init-keyword #:size

  alignment #:accessor !alignment
  #:init-keyword #:alignment

  fields #:accessor !fields
  #:init-keyword #:fields

  is-discriminated? #:accessor !is-discriminated?
  #:init-keyword #:is-discriminated?

  discriminator-offset #:accessor !discriminator-offset
  #:init-keyword #:discriminator-offset

  discriminator #:accessor !discriminator #:init-keyword #:discriminator #:init-value #f
The name slot is automatically initialized.

Instances of the `<gi-union>` are immutable (to be precise, there are not meant to be mutated, see [GOOPS Notes and Conventions], page 7, 'Slots are not Immutable').

### Procedures, Accessors and Methods

**make-c-union**

```scheme
make-c-union types [type #f] [val #f]
```

*Procedure*

Create a foreign pointer to a C union for the list of types (see Foreign Types (https://www.gnu.org/software/guile/manual/guile.html#Foreign-Types) in the Guile Reference Manual for a list of supported types).

**c-union-ref**

```scheme
c-union-ref foreign size type
```

*Procedure*

Returns the content of the C union pointed by `foreign`, for the given `size` and `type`.

**c-union-set!**

```scheme
c-union-set! foreign size type val
```

*Procedure*

Sets the content of the C union pointed by `foreign` to `val`, given its `size` and `type`.

**!g-type**

```scheme
!g-type (inst <gi-union>)
```

*Accessor*

Returns the content of their respective slot for `inst`.

**Utilities**

### Procedures

- [g-studly-caps-expand], page 123
- [g-name->name], page 123
- [g-name->class-name], page 123
- [g-name->short-name], page 123
- [class-name->name], page 124
- [class-name->g-name], page 124
- [name->g-name], page 124
- [syntax-name->method-name], page 124
- [gi-type-tag->ffi], page 125
- [gi-type-tag->init-val], page 125

### Description

G-Golf utilities low level API.
Procedures

g-studly-caps-expand str
Returns a string\(^\text{30}\). Expand the StudlyCaps str to a more schemey-form, according to the conventions of GLib libraries. For example:

\[
\begin{align*}
\text{(g-studly-caps-expand "GStudlyCapsExpand")} & \Rightarrow \text{"g-studly-caps-expand"} \\
\text{(g-studly-caps-expand "GSource")} & \Rightarrow \text{"g-source"} \\
\text{(g-studly-caps-expand "GtkIMContext")} & \Rightarrow \text{"im-context"}
\end{align*}
\]

G-Golf slightly modified the original code to also allow the possibility to specially treat the str (expanded) tokens, such as:

\[
\begin{align*}
\text{(g-studly-caps-expand "WebKitWebContext")} & \Rightarrow \text{"webkit-web-context" ;; not "web-kit-web-context"}
\end{align*}
\]

The list of StudlyCaps token exception pairs are maintained in the [g-studly-caps-expand-token-exception], page 34, alist.

g-name->name g-name [as-string? #f] [Procedure]
g-name->class-name g-name [as-string? #f] [Procedure]

Return a symbol name, or a string name if as-string is #t.

\[\text{[g-name->name], page 123, first obtains, the scheme representation of } g-name, \text{as a string, by looking for a possible entry in [g-name-transform-exception], page 34, or if it failed, by calling [g-studly-caps-expand], page 123.}\]

If the optional as-string argument is #t, it returns that string, otherwise, it calls and returns the result of string->symbol.

\[\text{[g-name->class-name], page 123, calls [g-name->name], page 123, surrounds the result using #\langle and #\rangle characters then either return that string, if as-string? is #t, otherwise it calls and returns the result of string->symbol:}\]

\[
\begin{align*}
\text{(g-name->class-name "GtkWindow")} & \Rightarrow \text{<gtk-window>}
\end{align*}
\]

\text{\[g-name->short-name g-name g-class-name [as-string? #f] [Procedure]\]

Return a symbol name, or a string name if as-string is #t.\]

\(^{30}\)This procedure, as well as [g-name->name], page 123, and [g-name->class-name], page 123, come from Guile-GNOME (https://www.gnu.org/software/guile-gnome), where there are named GStudlyCapsExpand, gtype-name->scm-name and gtype-name->class-name.

In G-Golf, these procedures are also be used to transform other (GObject Introspection) names, such as function names, hence they use the g-name-> prefix instead.
Obtains and returns a (method) short name for g-name. It first obtains the sro (scheme representation of) both g-name and g-class-name (which is expected to be the upstream method container (class) name), as a string, then:

- if the sro g-class-name is (fully) contained in the sro g-name, it drops the sro g-class-name prefix - or its plural form - and its trailing #\- (hiphen) delimiter from the sro g-name;
- otherwise, it drops the longest common sro string prefix it finds.

If the optional as-string argument is #t, it returns that string, otherwise, it calls and returns the result of string->symbol.

To illustrate, here is an example for each of the three above exposed cases:

- \[(g-name->shortname "gdk_event_get_event_type" "GdkEvent")\]
  \[⇒ get-event-type\]

- \[(g-name->shortname "gdk_events_get_angle" "GdkEvent")\]
  \[⇒ get-angle\]

- \[(g-name->short-name "gtk_drag_begin" "GtkWidget")\]
  \[⇒ drag-begin\]

\[class-name->name class-name\]  [Procedure]
Returns a (symbol) name.
Obtains and returns the (symbol) name for class-name, by dropping the surrounding '<' and '>' characters. For example:

- \[(class-name->name '<foo-bar>')\]
  \[⇒ 'foo-bar\]

\[class-name->g-name class-name\]  [Procedure]
Returns a string.
Obtains and returns the StudlyCaps string representation for class-name. For example:

- \[(class-name->g-name '<foo-bar>')\]
  \[⇒ "FooBar"\]

\[name->g-name name [as-string? #f]\]  [Procedure]
Return a symbol, or a string if as-string is #t.
Unless name is a string, it first calls (symbol->string name), then changes all occurrences of - (hyphen) to _ (underscore) (other characters are not valid in a g-name).
If the optional as-string argument is #t, it returns that string, otherwise, it calls and returns the result of string->symbol.

\[syntax-name->method-name name\]  [Procedure]
Returns a (symbol) name.
This procedure is used to ‘protect’ syntax names, from being redefined as generic functions and methods.
Users should normally not call this procedure - except for testing purposes, if/when they customize its default settings - it is appropriately and automatically called by G-Golf when importing a GI typelib.

Here is what it does:

- it first checks if a renamer is available, by calling [syntax-name-protect-renamer], page 38, and if so, calls it passing name and returns the result;

- if no renamer is available, it checks if either or both [syntax-name-protect-prefix], page 38, and [syntax-name-protect-postfix], page 38, is(are) available, calls symbol-append adequately passing either or both and name and returns the result.

- It will raise an exception if none of the syntax name protect prefix, postfix and renamer is available.

See [Customization Square], page 32, - GI Syntax Name Protect. G-Golf GI Syntax Name Protect default values are:

[syntax-name-protect-prefix], #f
page 38,
[syntax-name-protect-postfix], ' _ (the symbol _)
page 38,
[syntax-name-protect-renamer], #f
page 38,

As an example, using these default settings, the method short name for gcr-secret-exchange-begin would be begin_.

**gi-type-tag->ffi type-tag**

[Procedure]

Returns an integer or '*' (the symbol *).

Obtains the corresponding Guile's ffi tag value for type-tag, which must be a member of [%gi-type-tag], page 78. If type-tag is unknown, an exception is raised. Note that Guile's ffi tag values are integers or '*' (the symbol *, used by convention to denote pointer types.

**gi-type-tag->init-val type-tag**

[Procedure]

Returns the default init value for type-tag.

Obtains and returns the default init value for type-tag, which will either be 0 (zero), or %null-pointer.
G-Golf High Level API

G-Golf High Level API modules are defined in the hl-api subdirectory, such as (g-golf hl-api gobject).

Where you may load these modules individually, the easiest way to use the G-Golf High Level API is to import the hl-api module: it imports and re-exports the public interface of (oop goops), some G-Golf support modules and all G-Golf High Level API modules:

(use-modules (g-golf hl-api))

As stated in the introduction, G-Golf high level API (main) objective is to make (imported) GOBject classes and methods available using GOOPS, the Guile Object Oriented System (see Section “GOOPS” in The GNU Guile Reference Manual), in a way that is largely inspired by Guile-Gnome (https://www.gnu.org/software/guile-gnome).

Closure

G-Golf closure high level API.
The G-Golf integration with GObject Closures.

Classes

[closure], page 127

Accessors and Methods

[closure], page 127

[function], page 127

[return-type], page 127

[param-types], page 127

[invoke], page 127

Description

The GLib/GObject type system supports the creation and invocation of ‘Closures’, which represents a callback supplied by the programmer (see [Closures], page 69, if you are curious about the low-level description and API, though you don’t need to to understand and use the high level API described here).

Its infrastructure allows one to pass a Scheme function to C, and have C call into Scheme, and vice versa. In Scheme, a <closure> instance holds a pointer to a GClosure instance, a Scheme procedure, the type of its return value, and a list of the type of its arguments.

Closures can be invoked with [invoke], page 127, for example:

(use (g-golf))

(make-closure
  #:function (lambda (a b) (+ a b))
  #:return-type 'int
  #:param-types '(int int))
⇒ $2 = #<closure> 55f24a0228d0>

(invoke $2 3 2)
⇒ $3 = 5

Classes

$<\text{closure}>$

Its slots are:

$g$-closure 
#:accessor !g-closure

function 
#:accessor !function
#:init-keyword #:function

return-type 
#:accessor !return-type
#:init-keyword #:return-type

param-types 
#:accessor !param-types
#:init-keyword #:param-types

The #:return-type and #:param-types accept respectively one symbol and a list of symbols that are members of the [%g-type-fundamental-types], page 57.

Instances of the $<\text{closure}>$ class are immutable (to be precise, there are not meant to be mutated, see [GOOPS Notes and Conventions], page 7, 'Slots are not Immutable').

Accessors and Methods

Note: in this section, the closure argument is [must be] a $<\text{closure}>$ instance.

!g-closure closure [Accessor]
!function closure [Accessor]
!return-type closure [Accessor]
!param-types closure [Accessor]

Returns the content of their respective slot for closure.

invoke closure . args [Method]

Returns the result of the invocation of closure, using (the possibly empty list of) args.

This is a ‘low level’ method, not used internally, provided mainly for debugging (or demonstration) purposes, so you may test and verify your callbacks and signals procedures$^{31}$.

Function

G-Golf GI function and argument high level API.
The G-Golf GI function and argument high level API.

Classes

[<function>], page 130
[<argument>], page 131

$^{31}$ From scheme, you would ‘immediately’ call the procedure instead of course.
Accessors and Methods

[!info_], page 132
[!namespace_], page 132
[!g-name____], page 132
[!name], page 132
[!override?], page 132
[!i-func], page 132
[!o-func], page 132
[!o-spec-pos], page 132
[!flags], page 132
[!is-method?], page 132
[!n-arg], page 132
[!caller-owns], page 132
[!return-type_], page 132
[!type-desc], page 132
[!may-return-null], page 132
[!arguments], page 132
[!n-gi-arg-in], page 132
[!args-in], page 132
[!gi-args-in], page 132
[!gi-args-in-bv], page 132
[!n-gi-arg-out], page 132
[!args-out], page 132
[!gi-args-out], page 132
[!gi-args-out-bv], page 132
[!gi-arg-result], page 132
[!g-name____], page 133
[!name_], page 133
[!closure], page 133
[!destroy], page 133
[!direction], page 133
[!transfert], page 133
[!scope], page 133
[!type-tag], page 133
[!type-desc_], page 133
[!forced-type], page 133
[!string-pointer], page 133
[!is-pointer?], page 133
[!may-be-null?], page 133
[!is-caller-allocate?], page 133
[!is-optional?], page 133
[!is-return-value?], page 133
[!is-skip?], page 133
[!arg-pos], page 133
[!gi-argument-in], page 133
[!gi-argument-in-bv-pos], page 133
[!gi-argument-out], page 133
[!gi-argument-out-bv-pos], page 133
[!gi-argument-field], page 133
Classes

<function>
Its slots are:

- info #:accessor !info
- namespace #:accessor !namespace
- g-name #:accessor !g-name
- name #:accessor !name
- override? #:accessor !override?
- i-func #:accessor !i-func
- o-func #:accessor !o-func
- o-spec-pos #:accessor !o-spec-pos
- flags #:accessor !flags
- is-method? #:accessor !is-method
- n-arg #:accessor !n-arg
- caller-owns #:accessor !caller-owns
- return-type #:accessor !return-type
- type-desc #:accessor !type-desc
- may-return-null? #:accessor !may-return-null?
- arguments #:accessor !arguments
- n-gi-arg-in #:accessor !n-gi-arg-in
- args-in #:accessor !args-in
- gi-args-in #:accessor !gi-args-in
- gi-args-in-bv #:accessor !gi-args-in-bv
- n-gi-arg-out #:accessor !n-gi-arg-out
Instances of the `<function>` class are immutable (to be precise, there are not meant to be mutated, see [GOOPS Notes and Conventions], page 7, 'Slots are not Immutable').

<argument>

Its slots are:

g-name  #:accessor !g-name
    #:init-keyword #:g-name
name    #:accessor !name
    #:init-keyword #:name
closure #:accessor !closure
destroy #:accessor !destroy
direction #:accessor !direction
    #:init-keyword #:direction
transfert #:accessor !transfert
scope    #:accessor !scope
type-tag #:accessor !type-tag
    #:init-keyword #:type-tag
type-desc #:accessor !type-desc
    #:init-keyword #:type-desc
forced-type #:accessor !forced-type
    #:init-keyword #:forced-type
string-pointer #:accessor !string-pointer
is-pointer? #:accessor !is-pointer?
    #:init-keyword #:is-pointer?
may-be-null? #:accessor !may-be-nul?
    #:init-keyword #:may-be-null?
is-caller-allocate?
   #:accessor !is-caller-allocate?
is-optional?
   #:accessor !is-optional?
is-return-value?
   #:accessor !is-return-value?
is-skip?  #:accessor !is-skip?
arg-pos   #:accessor !arg-pos
   #:init-keyword #:arg-pos
gi-argument-in
   #:accessor !gi-argument-in
   #:init-value #f
gi-argument-in-bv-pos
   #:accessor !gi-argument-in-bv-pos
   #:init-value #f
gi-argument-out
   #:accessor !gi-argument-out
   #:init-value #f
gi-argument-out-bv-pos
   #:accessor !gi-argument-out-bv-pos
   #:init-value #f
name      #:accessor !gi-argument-field
   #:init-keyword #:gi-argument-field

Instances of the <argument> class are immutable (to be precise, there are not meant to be mutated, see [GOOPS Notes and Conventions], page 7, 'Slots are not Immutable').

Accessors and Methods
Note: in this section, the function and argument arguments are [must be] a <function> and an <argument> instance, respectively.

!info function [Accessor]
!namespace function [Accessor]
!g-name function [Accessor]
!name function [Accessor]
!override? function [Accessor]
!i-func function [Accessor]
!o-func function [Accessor]
!o-spec-pos function [Accessor]
!flags function [Accessor]
!is-method? function [Accessor]
!n-arg function [Accessor]
!caller-owns function [Accessor]
!return-type function [Accessor]
Returns the content of their respective slot for function.

Returns the content of their respective slot for argument.

Import

G-Golf GI import interfaces.
The G-Golf GI namespace (Typelib) import interfaces.
Procedures

[gi-import-info], page 134
[gi-import-enum], page 134
[gi-import-flags], page 134
[gi-import-struct], page 134
[gi-import-function], page 135
[gi-import-constant], page 136

Variables

[%gi-base-info-types], page 136
[%gi-imported-base-info-types], page 136

Procedures

[Procedure]

[gi-import-info] info

Returns the object or constant returned by the one of the gi-import-enum, gi-import-flags, ..., called upon info.

Obtains the GIBaseInfo type for info and uses it to dispatch a call to gi-import-enum, gi-import-enum, ..., and returns the object or constant returned by the procedure that has been called.

You probably will prefer to call [gi-import-by-name], page 19, most of the time, but here is an example:

,use (g-golf)
(g-irepository-require "Clutter")
⇒ $2 = #<pointer 0x5642cb065e30>

(g-irepository-find-by-name "Clutter" "ActorFlags")
⇒ $3 = #<pointer 0x5642cb067de0>

(gi-import-info $3)
⇒ $4 = #<gi-flags> 5642cb13c5d0

(describe $4)
⇒ #<gi-flags> 5642cb13c5d0> is an instance of class <gi-flags>
⇒ Slots are:
⇒   enum-set = ((mapped . 2) (realized . 4) (reactive . 8) (visible . 16) (no-layout . 32))
⇒   g-type = 94844874149456
⇒   g-name = "ClutterActorFlags"
⇒   name = clutter-actor-flags

[Procedure]

[gi-import-enum] info [#:with-method ?t]

[Procedure]

[gi-import-flags] info [#:with-method ?t]

[Procedure]

[gi-import-struct] info [#:with-method ?t]

Returns a [gi-enum], page 116, a [gi-flags], page 118, or a [gi-struct], page 119, instance, respectively.
The \textit{info} argument is (must be) a pointer to \texttt{GEnumInfo}, a \texttt{GEnumInfo} for which ([\texttt{g-base-info-get-type}], page 81, \textit{info}) returned 'flags and a \texttt{GIEnumInfo} respectively. It is an error to call any of these procedures upon an invalid \textit{info} argument.

The optional keyword 
\texttt{#:with-method} argument - which is \#t by default - is passed using \#f, then \textit{info} will be imported without its respective methods. A description and an example were also given here above, as part of the [\texttt{gi-import-by-name}], page 19, documentation entry.

Every imported [\texttt{<gi-enum>}, page 116, \texttt{<gi-flags>}, page 118, and \texttt{<gi-struct>}, page 119, instance is cached under the 'enum, 'flags and 'boxed main key (respectively), using the content of their (symbol) name slot as the secondary key.

For example, reusing the "Clutter" "ActorFlags" namespace/name introduced above, you would retrieve its [\texttt{<gi-flags>}, page 118, instance as is:

\begin{verbatim}
  (gi-cache-ref 'flags 'clutter-actor-flags)
⇒ $6 = #<<gi-flags> 5642cb13c5d0>
\end{verbatim}

\texttt{gi-import-function \textit{info}}

\texttt{gi-import-function \textit{info}} Returns a [\texttt{<function>}, page 130, instance.

Imports \textit{info} - a pointer to a \texttt{GIFunctionInfo} (see [Function Info], page 84), which represents a function, a method or a constructor - in Guile and exports its interface. This procedure also imports, recursively (and exports the interface of) its argument's type(s) and method(s).

Every imported function, method and constructor is cached under 'function main key, and using the value of their [\texttt{<function>}, page 130, instance name slot as the secondary key. Here is an example:

\begin{verbatim}
use (g-golf)
(g-irepository-require "Clutter")
⇒ $2 = #<pointer 0x55c191f3fe30>

(g-irepository-find-by-name "Clutter" "init")
⇒ $3 = #<pointer 0x55c191f41de0>

(gi-import-function $3)
⇒ $4 = #<<function> 55c191e81510>

(describe $4)
⇒ #<<function> 55c191e81510> is an instance of class <function>
⇒ Slots are:
⇒   info = #<pointer 0x55c191f41de0>
⇒   name = clutter-init
⇒   flags = ()
⇒   n-arg = 2
⇒   caller-owns = nothing
⇒   return-type = interface
\end{verbatim}
In most situations - unless the return-type is 'void (in which case nothing is returned) - the function or method returned value comes first, then in order, if any, the value(s) of its 'inout and 'out argument(s).

However, some function and method, that have at least one 'inout or 'out argument(s), do return #t or #f solely to indicate that the function or method call was successful or not. It is only if the call is successful that the 'inout and 'out argument(s) have been 'correctly' set and may safely be used.

In scheme, when binding such a function or method, we would rather (a) when the call is successful, elude the boolean and return, in order, the 'inout and/or 'out argument(s) value(s); and (b), when the call is unsuccessful, raise an exception.

Since it is not possible to automatically detect these functions and methods, G-Golf provides a series of interfaces to maintain, at user discretion and responsibility, a list of GI typelib functions and methods for which G-Golf is expected to elude their result value from the returned value(s). G-Golf interfaces to maintain this list are documented in the [Customization Square], page 32, section.

\textbf{gi-import-constant} \textit{info} \hfill [\textbf{Procedure}]

Returns two values, the constant value and its name.

Obtains and returns the \textit{info} constant value and its name. For example:

\begin{verbatim}
use (g-golf)
(g-irepository-require "GLib")
⇒ #<pointer 0x55ad58e6ae00>

(g-irepository-find-by-name "GLib" "PRIORITY_DEFAULT_IDLE")
⇒ $3 = #<pointer 0x55ad58e6cde0>

(gi-import-constant $3)
⇒ $4 = 200
⇒ $5 = "PRIORITY_DEFAULT_IDLE"
\end{verbatim}

Constants are currently not being automatically imported, though this will probably change in the near future, stay tuned.

\textbf{Variables}

\texttt{\%gi-base-info-types} \hfill [\textbf{Variable}]
\texttt{\%gi-imported-base-info-types} \hfill [\textbf{Variable}]

A (cumulative) list of the distinct (top level) base info types contained in the imported namespace(s).

These two variables have no other purpose then offering a feedback about: (a) the (top level) base info types contained in the namespace(s) passed to [gi-import], page 19;
(b) the (top level) base info types that have effectively been imported - when G-Golf is complete, both lists should be identical.

Initially, these variables are empty. As [gi-import], page 19, [gi-import-info], page 134, and/or [gi-import-by-name], page 19, are being called, they are filled with new types, which are added to both lists.

Note that the order in which base info types appear in these two lists is irrelevant, and may slightly vary, depending on the order of the namespace used for the successive [gi-import], page 19, calls and how complete is G-Golf.

Utilities
G-Golf additional utilities.

Procedures

[gi-find-by-property-name], page 137

Description
G-Golf additional utilities.

Procedures

[gi-find-by-property-name] namespace name

Returns a (possibly empty) list.

Obtains and returns a (possibly empty) list of (pointers to) GIObjectInfo in namespace that have a property named name. Property names are obtained calling g-base-info-get-name, with no translation/transformation - underscore, if any, are kept 'as is', and the comparison with name is case sensitive.

Appendix A GNU Free Documentation License

Version 1.3, 3 November 2008

http://fsf.org/

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

0. PREAMBLE
The purpose of this License is to make a manual, textbook, or other functional and useful document free in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or non-commercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.
This License is a kind of “copyleft”, which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The “Document”, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as “you”. You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A “Modified Version” of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A “Secondary Section” is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document’s overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The “Invariant Sections” are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The “Cover Texts” are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A “Transparent” copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image
format is not Transparent if used for any substantial amount of text. A copy that is not “Transparent” is called “Opaque”.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The “Title Page” means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, “Title Page” means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

The “publisher” means any person or entity that distributes copies of the Document to the public.

A section “Entitled XYZ” means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as “Acknowledgements”, “Dedications”, “Endorsements”, or “History”.) To “Preserve the Title” of such a section when you modify the Document means that it remains a section “Entitled XYZ” according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document’s license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher.
of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document’s license notice.
H. Include an unaltered copy of this License.

I. Preserve the section Entitled “History”, Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled “History” in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the “History” section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled “Acknowledgements” or “Dedications”, Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled “Endorsements”. Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled “Endorsements” or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version’s license notice. These titles must be distinct from any other section titles.

You may add a section Entitled “Endorsements”, provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.
5. COMBINING DOCUMENTS
You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled “History” in the various original documents, forming one section Entitled “History”; likewise combine any sections Entitled “Acknowledgements”, and any sections Entitled “Dedications”. You must delete all sections Entitled “Endorsements.”

6. COLLECTIONS OF DOCUMENTS
You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS
A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an “aggregate” if the copyright resulting from the compilation is not used to limit the legal rights of the compilation’s users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document’s Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION
Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the
license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled “Acknowledgements”, “Dedications”, or “History”, the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License “or any later version” applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy’s public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

“Massive Multiauthor Collaboration Site” (or “MMC Site”) means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of
such a server. A “Massive Multiauthor Collaboration” (or “MMC”) contained in the
site means any set of copyrightable works thus published on the MMC site.
“CC-BY-SA” means the Creative Commons Attribution-Share Alike 3.0 license pub-
lished by Creative Commons Corporation, a not-for-profit corporation with a principal
place of business in San Francisco, California, as well as future copyleft versions of that
license published by that same organization.
“Incorporate” means to publish or republish a Document, in whole or in part, as part
of another Document.
An MMC is “eligible for relicensing” if it is licensed under this License, and if all works
that were first published under this License somewhere other than this MMC, and
subsequently incorporated in whole or in part into the MMC, (1) had no cover texts
or invariant sections, and (2) were thus incorporated prior to November 1, 2008.
The operator of an MMC Site may republish an MMC contained in the site under
CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is
eligible for relicensing.
ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

Copyright (C) year your name.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.3
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover
Texts. A copy of the license is included in the section entitled "GNU
Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

with the Invariant Sections being list their titles, with
the Front-Cover Texts being list, and with the Back-Cover Texts
being list.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

\input texinfo
## Concept Index

This index contains concepts, keywords and non-Schemey names for several features, to make it easier to locate the desired sections.

<table>
<thead>
<tr>
<th>C</th>
<th>L</th>
</tr>
</thead>
<tbody>
<tr>
<td>copying</td>
<td>license</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>G</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPL</td>
<td>the GNU Project</td>
</tr>
</tbody>
</table>
This is an alphabetical list of all the procedures, methods and macros in G-Golf.
<table>
<thead>
<tr>
<th>Procedure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>g-irepository-find-by-gtype</td>
<td>76</td>
</tr>
<tr>
<td>g-irepository-find-by-name</td>
<td>76</td>
</tr>
<tr>
<td>g-irepository-get-c-prefix</td>
<td>76</td>
</tr>
<tr>
<td>g-irepository-get-default</td>
<td>75</td>
</tr>
<tr>
<td>g-irepository-get-dependencies</td>
<td>75</td>
</tr>
<tr>
<td>g-irepository-get-info</td>
<td>75</td>
</tr>
<tr>
<td>g-irepository-get-loaded-namespaces</td>
<td>75</td>
</tr>
<tr>
<td>g-irepository-get-n-infos</td>
<td>75</td>
</tr>
<tr>
<td>g-irepository-get-shared-library</td>
<td>76</td>
</tr>
<tr>
<td>g-irepository-get-typelib-path</td>
<td>75</td>
</tr>
<tr>
<td>g-irepository-get-version</td>
<td>76</td>
</tr>
<tr>
<td>g-irepository-require</td>
<td>76</td>
</tr>
<tr>
<td>g-list-data</td>
<td>51</td>
</tr>
<tr>
<td>g-list-free</td>
<td>51</td>
</tr>
<tr>
<td>g-list-length</td>
<td>51</td>
</tr>
<tr>
<td>g-list-next</td>
<td>51</td>
</tr>
<tr>
<td>g-list-nth-data</td>
<td>51</td>
</tr>
<tr>
<td>g-list-prev</td>
<td>51</td>
</tr>
<tr>
<td>g-main-context-default</td>
<td>45</td>
</tr>
<tr>
<td>g-main-context-new</td>
<td>45</td>
</tr>
<tr>
<td>g-main-loop-new</td>
<td>44</td>
</tr>
<tr>
<td>g-main-loop-quit</td>
<td>45</td>
</tr>
<tr>
<td>g-main-loop-ref</td>
<td>44</td>
</tr>
<tr>
<td>g-main-loop-run</td>
<td>45</td>
</tr>
<tr>
<td>g-main-loop-unref</td>
<td>45</td>
</tr>
<tr>
<td>g-malloc</td>
<td>43</td>
</tr>
<tr>
<td>g-malloc0</td>
<td>43</td>
</tr>
<tr>
<td>g-memdup</td>
<td>43</td>
</tr>
<tr>
<td>g-name-&gt;class-name</td>
<td>123</td>
</tr>
<tr>
<td>g-name-&gt;name</td>
<td>123</td>
</tr>
<tr>
<td>g-name-&gt;short-name</td>
<td>123</td>
</tr>
<tr>
<td>g-name-&gt;transform-exception</td>
<td>34</td>
</tr>
<tr>
<td>g-name-&gt;transform-exception-add</td>
<td>34</td>
</tr>
<tr>
<td>g-name-&gt;transform-exception-remove</td>
<td>34</td>
</tr>
<tr>
<td>g-name-&gt;transform-exception-reset</td>
<td>34</td>
</tr>
<tr>
<td>g-object-class-find-property</td>
<td>59</td>
</tr>
<tr>
<td>g-object-get-property</td>
<td>60</td>
</tr>
<tr>
<td>g-object-info-find-method</td>
<td>97</td>
</tr>
<tr>
<td>g-object-info-find-signal</td>
<td>97</td>
</tr>
<tr>
<td>g-object-info-get-abstract</td>
<td>96</td>
</tr>
<tr>
<td>g-object-info-get-class-struct</td>
<td>97</td>
</tr>
<tr>
<td>g-object-info-get-constant</td>
<td>96</td>
</tr>
<tr>
<td>g-object-info-get-field</td>
<td>96</td>
</tr>
<tr>
<td>g-object-info-get-interface</td>
<td>96</td>
</tr>
<tr>
<td>g-object-info-get-method</td>
<td>97</td>
</tr>
<tr>
<td>g-object-info-get-n-constants</td>
<td>96</td>
</tr>
<tr>
<td>g-object-info-get-n-fields</td>
<td>96</td>
</tr>
<tr>
<td>g-object-info-get-n-properties</td>
<td>97</td>
</tr>
<tr>
<td>g-object-info-get-n-signals</td>
<td>97</td>
</tr>
<tr>
<td>g-object-info-get-n-vfuncs</td>
<td>97</td>
</tr>
<tr>
<td>g-object-info-get-parent</td>
<td>96</td>
</tr>
<tr>
<td>g-object-info-get-property</td>
<td>97</td>
</tr>
<tr>
<td>g-object-info-get-property</td>
<td>97</td>
</tr>
<tr>
<td>g-object-info-get-type-init</td>
<td>96</td>
</tr>
<tr>
<td>g-object-info-get-type-name</td>
<td>96</td>
</tr>
<tr>
<td>g-object-info-get-type-init</td>
<td>96</td>
</tr>
<tr>
<td>g-object-info-get-type-name</td>
<td>96</td>
</tr>
<tr>
<td>g-object-info-get-vfunc</td>
<td>97</td>
</tr>
<tr>
<td>g-object-is-floating</td>
<td>60</td>
</tr>
<tr>
<td>g-object-new</td>
<td>59</td>
</tr>
<tr>
<td>g-object-new-with-properties</td>
<td>59</td>
</tr>
<tr>
<td>g-object-ref</td>
<td>59</td>
</tr>
<tr>
<td>g-object-ref-count</td>
<td>60</td>
</tr>
<tr>
<td>g-object-ref-sink</td>
<td>59</td>
</tr>
<tr>
<td>g-object-set-property</td>
<td>60</td>
</tr>
<tr>
<td>g-object-type</td>
<td>60</td>
</tr>
<tr>
<td>g-object-type-name</td>
<td>60</td>
</tr>
<tr>
<td>g-object-unref</td>
<td>59</td>
</tr>
<tr>
<td>g-param-spec-get-blurb</td>
<td>68</td>
</tr>
<tr>
<td>g-param-spec-get-default-value</td>
<td>68</td>
</tr>
<tr>
<td>g-param-spec-get-flags</td>
<td>68</td>
</tr>
<tr>
<td>g-param-spec-get-name</td>
<td>68</td>
</tr>
<tr>
<td>g-param-spec-get-nick</td>
<td>68</td>
</tr>
<tr>
<td>g-param-spec-get-type</td>
<td>68</td>
</tr>
<tr>
<td>g-quark-from-string</td>
<td>53</td>
</tr>
<tr>
<td>g-quark-to-string</td>
<td>53</td>
</tr>
<tr>
<td>g-registered-type-info-get-g-type</td>
<td>88</td>
</tr>
<tr>
<td>g-registered-type-info-get-init</td>
<td>88</td>
</tr>
<tr>
<td>g-registered-type-info-get-type-name</td>
<td>88</td>
</tr>
<tr>
<td>g-set-prgname</td>
<td>49</td>
</tr>
<tr>
<td>g-signal-info-get-flags</td>
<td>84</td>
</tr>
<tr>
<td>g-signal-list-ids</td>
<td>73</td>
</tr>
<tr>
<td>g-signal-lookup</td>
<td>73</td>
</tr>
<tr>
<td>g-signal-parse-name</td>
<td>73</td>
</tr>
<tr>
<td>g-signal-query</td>
<td>72</td>
</tr>
<tr>
<td>g-slist-append</td>
<td>52</td>
</tr>
<tr>
<td>g-slist-data</td>
<td>52</td>
</tr>
<tr>
<td>g-slist-free</td>
<td>53</td>
</tr>
<tr>
<td>g-slist-length</td>
<td>53</td>
</tr>
<tr>
<td>g-slist-next</td>
<td>52</td>
</tr>
<tr>
<td>g-slist-prepare</td>
<td>52</td>
</tr>
<tr>
<td>g-source-attach</td>
<td>46</td>
</tr>
<tr>
<td>g-source-destroy</td>
<td>46</td>
</tr>
<tr>
<td>g-source-free</td>
<td>46</td>
</tr>
<tr>
<td>g-source-get-priority</td>
<td>47</td>
</tr>
<tr>
<td>g-source-get-field</td>
<td>46</td>
</tr>
<tr>
<td>g-source-get-interface</td>
<td>46</td>
</tr>
<tr>
<td>g-source-get-method</td>
<td>46</td>
</tr>
<tr>
<td>g-source-get-count</td>
<td>46</td>
</tr>
<tr>
<td>g-source-get-field</td>
<td>46</td>
</tr>
<tr>
<td>g-source-set-closure</td>
<td>71</td>
</tr>
<tr>
<td>g-source-set-priority</td>
<td>46</td>
</tr>
<tr>
<td>g-source-set-priority</td>
<td>46</td>
</tr>
<tr>
<td>g-source-unref</td>
<td>46</td>
</tr>
<tr>
<td>g-struct-info-get-alignment</td>
<td>91</td>
</tr>
<tr>
<td>g-struct-info-get-field</td>
<td>91</td>
</tr>
<tr>
<td>g-struct-info-get-method</td>
<td>92</td>
</tr>
<tr>
<td>g-struct-info-get-n-fields</td>
<td>91</td>
</tr>
<tr>
<td>g-struct-info-get-n-methods</td>
<td>91</td>
</tr>
<tr>
<td>g-struct-info-get-size</td>
<td>91</td>
</tr>
<tr>
<td>g-struct-info-get-unref</td>
<td>91</td>
</tr>
<tr>
<td>g-struct-info-is-foreign</td>
<td>91</td>
</tr>
<tr>
<td>g-struct-info-is-gtype-struct</td>
<td>91</td>
</tr>
</tbody>
</table>
Variable Index

This is an alphabetical list of all the important variables and constants in G-Golf.

<table>
<thead>
<tr>
<th>%g-function-info-flags of &lt;gi-flag&gt;</th>
<th>%g-cache</th>
<th>%g-io-condition of &lt;gi-flag&gt;</th>
<th>%g-direction of &lt;gi-enum&gt;</th>
<th>%g-param-flags of &lt;gi-enum&gt;</th>
<th>%g-imported-base-info-types</th>
<th>%g-signal-flags of &lt;gi-enum&gt;</th>
<th>%g-info-type of &lt;gi-enum&gt;</th>
<th>%g-type-fundamental-flags of &lt;gi-enum&gt;</th>
<th>%g-imported-base-info-types</th>
<th>%g-info-type of &lt;gi-enum&gt;</th>
<th>%g-array-type of &lt;gi-enum&gt;</th>
<th>%g-transfer of &lt;gi-enum&gt;</th>
<th>%g-base-info-types</th>
<th>%g-type-tag of &lt;gi-enum&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>86</td>
<td>32</td>
<td>48</td>
<td>103</td>
<td>68</td>
<td>136</td>
<td>73</td>
<td>82</td>
<td>56</td>
<td>136</td>
<td>79</td>
<td>103</td>
<td>104</td>
<td>136</td>
<td>78</td>
</tr>
</tbody>
</table>
Type Index

This is an alphabetical list of all the important data types defined in the G-Golf Programmers Manual.

<argument> ........................................... 131  <gi-struct> ........................................... 119
<closure> ........................................... 127  <gi-union> ........................................... 121
<enum> ........................................... 116  <ginterface> ........................................... 28
<flags> ........................................... 118  <gobject-class> ....................................... 28
<function> ........................................... 130  <gobject> ........................................... 28
<gdk-event> ........................................... 21  <gtype-class> ....................................... 29
<gi-enum> ........................................... 116  <gtype-instance> ..................................... 30
<gi-flags> ........................................... 118  <vfunc> ............................................. 40
List of Examples

Example 1: ................................................................. 12
Example 2: ................................................................. 17