GNU G-Golf

Edition 0.1.0, revision 1, for use with GNU G-Golf 0.1.0

The GNU G-Golf Developers
This manual documents GNU G-Golf version 0.1.0.

Copyright (C) 2016 - 2018 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
The G-Golf License ......................... 1

1 Introduction ................................. 1
1.1 About G-Golf ......................... 1
1.2 Obtaining and Installing G-Golf .......... 2
1.3 Contact .......................... 4
1.4 Reporting Bugs ................. 4

2 Using G-Golf ......................... 5

3 API Reference ............................. 5
3.1 API Overview .................................... 5
3.1.1 Naming Conventions .................. 6
3.1.2 Project and Documentation Structure .... 7
3.1.3 GOOPS Notes and Conventions ...... 10
3.2 Glib .......................................... 11
3.2.1 Memory Allocation ............... 11
3.2.2 The Main Event Loop ............ 12
3.2.3 IO Channels ..................... 15
3.2.4 UNIX-specific utilities and integration ... 17
3.2.5 Doubly-Linked Lists .......... 17
3.2.6 Singly-Linked Lists ............ 18
3.2.7 Quarks ..................................... 19
3.3 GObject ........................................ 20
3.3.1 Type Information ............... 20
3.3.2 GObject ........................ 23
3.3.3 Enumeration and Flag Types ...... 25
3.3.4 Generic Values .................. 25
3.3.5 Parameters and Values .......... 26
3.3.6 GParamSpec .................... 29
3.3.7 Closures ....................... 31
3.3.8 Signals .......................... 33
3.4 Gdk .......................................... 35
3.4.1 Events .......................... 35
3.4.2 Event Structures .......... 41
3.4.3 Key Values ...................... 45
3.5 GObject Introspection ............. 46
3.5.1 Repository .................... 46
3.5.2 Common Types .................. 48
3.5.3 Base Info ...................... 49
3.5.4 Callable Info ........................................ 52
3.5.5 Signal Info ........................................ 53
3.5.6 Function Info ...................................... 54
3.5.7 Registered Type Info ............................... 56
3.5.8 Enum Info .......................................... 57
3.5.9 Struct Info ......................................... 59
3.5.10 Union Info ......................................... 60
3.5.11 Object Info ........................................ 62
3.5.12 Arg Info ........................................... 65
3.5.13 Constant Info ..................................... 69
3.5.14 Field Info ......................................... 70
3.5.15 Property Info ..................................... 70
3.5.16 Type Info .......................................... 71
3.5.17 Typelib ............................................. 73
3.5.18 Utilities ............................................ 74

3.6 Support ................................................ 77
3.6.1 Modules ............................................ 77
3.6.2 Goops ................................................ 77
3.6.3 G-export .......................................... 78
3.6.4 Enum ................................................ 78
3.6.5 Flag .................................................. 80
3.6.6 Struct .............................................. 80
3.6.7 Union .............................................. 82
3.6.8 Utilities ............................................ 82

3.7 G-Golf High Level API ............................... 84
3.7.1 GType ............................................... 85
3.7.2 GObject ........................................... 86
3.7.3 Closure ............................................ 86
3.7.4 Function ........................................... 88
3.7.5 Import .............................................. 94

Appendix A GNU Free Documentation License .......... 98

Concept Index ............................................ 106

Procedure Index .......................................... 107

Variable Index ........................................... 111

Type Index ................................................. 112
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.

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 98).

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

1 Introduction

1.1 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 Introspection (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://developer.gnome.org/gi/stable/overview.html) - a binary, readonly, an interpreter and compiler for the Scheme (http://schemers.org) programming language.

\(^1\) GNU Guile (http://www.gnu.org/software/guile)
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).

1.2 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 >= 2.69
- Automake >= 1.14
- Makeinfo >= 6.6
- Guile (http://www.gnu.org/software/guile) >= 2.0.14 [allows 2.2 3.0]
- Guile-Lib (http://www.nongnu.org/guile-lib) >= 0.2.5
- Glib-2.0 (https://developer.gnome.org/glib/stable/) >= 2.48.0
- Gobject-2.0 (https://developer.gnome.org/gobject/stable/) >= 2.48.0
- Glib+3.0 (https://developer.gnome.org/gobject/stable/) >= 3.24.0
- GObject-Introspection-1.0 (https://developer.gnome.org/stable/gi) >= 1.48.0

Install from the tarball
Assuming you have satisfied the dependencies, open a terminal and proceed with the following steps:

cd <download-path>
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 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 %load-compiled-path at run time if you prefer2 (See Environment Variables (https://www.gnu.org/software/guile/manual/guile.html#Environment-Variables))

---

2 In this case, you may as well decide to either alter your $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)". 
1.3 Contact

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/).

IRC

Most of the time you can find me on irc, channel #guile, #guix and #scheme on irc.freenode.net, #clutter and #introspection on irc.gnome.org, under the nickname daviid.

1.4 Reporting Bugs

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

3 Contact your administrator if you opt for the second solution but don’t have write priviledges on your system.
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).

2 Using G-Golf

G-Golf Reference Manual still is a mock-up: any help is more then welcome to improve this situation, thanks!

3 API Reference

3.1 API Overview

G-Golf Low Level API

The G-Golf low level API provides a one to one binding to (some of) the Glib, GObject and (most of) the GObject Introspection functions, using the ‘original’ name, except that _ are replaced by -.

At this low level API, both arguments and returned values are pointers, except for enums, strings, numbers and booleans. This is also true for the minimal ‘bootstrap’ bindings G-golf must implement for Glib and GObject.

Here is an example:

```
,use (g-golf)
 (g-irepository-require "Clutter")
 preca $2 = #<pointer 0x1681600>
```

In addition, other procedures, syntax, variables, classes and methods are also provided and part of the so called low level API, that use the above mentioned bindings in order to import, build and make GIR libraries functionality available from Guile (http://www.gnu.org/software/guile).

G-Golf High Level API

G-Golf will also provide a high level API, mostly based on GOOPS (see Section “GOOPS” in The GNU Guile Reference Manual), largely inspired by Guile-Gnome (https://www.gnu.org/software/guile-gnome).

Once G-Golf has its high level API ready, the idea is that you should be able to use just any GIR library, such as Clutter. Here is a simple ‘simulation’ example4:

```
,use (g-golf)
```

---

4 You won’t be able to try this example now, since there is no high level API in G-Golf yet, but so you get the idea . . .
(gi-import "Clutter")

(make <clutter-stage>)
-| $3 = #<<clutter-stage> 22782a0>

3.1.1 Naming Conventions

Glib and GObject functions
G-Golf low level API procedure names that bind a Glib or GObject function always use the ‘original’ name, except that _ are replaced by -. For example:

\[ g\_main\_loop\_new \rightarrow [g\_main\_loop\_new], \text{page 12} \]

GObject Introspection Function
G-Golf low level API procedure names that bind a GI function also always use the ‘original’ name, except that _ are replaced by -. For example:

\[ g\_irepository\_get\_loaded\_namespaces \rightarrow [g\_irepository\_get\_loaded\_namespaces], \text{page 47} \]

Types and Values
G-Golf low level API variables that bind Glib, GObject and GI types and values use the exact same convention as for procedures, except that they always start with %%. For example, from the GIBaseInfo section:

\[ \text{GIInfoType} \rightarrow [%\text{gi-info-type}], \text{page 52} \]

G-Golf
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 GI, Glib or GObject type, such as:

  \[ [\text{call-with-input-typelib}], \text{page 73} \]

- syntax names that start as with- followed by a GI, Glib or GObject type, such as:

  \text{with-gerror}
When an ‘obvious’ name can’t be find ‘on its own’, or to avoid possible conflict outside G-Golf\textsuperscript{5}, then the name starts using the gi- prefix, and equally for variables, using %gi-.

**Gnome Libraries**

Imported Gnome libraries will follow Guile-Gnome (https://www.gnu.org/software/guile-gnome)’s naming conventions, from which it will actually reuse the functionality developed at the time (see 9.1.1 Mapping class libraries to scheme (https://www.gnu.org/software/guile-gnome/docs/gobject/html/gnome-gobject-generics.html#gnome-gobject-generics) and 10.2 Usage (https://www.gnu.org/software/guile-gnome/docs/gobject/html/gnome-gobject-utils.html#gnome-gobject-utils) in the GNU Guile-Gnome: GObject Reference Manual). For example, the following names would be transformed like this:

\begin{verbatim}
ClutterActor -> clutter-actor
clutter_actor_new -> clutter-actor-new
clutter_actor_hide -> clutter-actor-hide
\end{verbatim}

The following class names would be transformed like this:

\begin{verbatim}
ClutterActor -> <clutter-actor>
GtkWindow -> <gtk-window>
\end{verbatim}

Once G-Golf high level API is also implemented, Gnome libraries GObject classes and methods will become GOOPS citizen (see Section “GOOPS” in The GNU Guile Reference Manual), and in the land of generic function multi methods polimorphic object oriented systems, users just have fun:

\begin{verbatim}
,use (g-golf)
  (g-golf-import "Clutter")
- $2 = #<<g-golf-clutter-typelib> 18237a0>

  (make <clutter-actor>)
- $3 = #<<clutter-actor> 32124b0>
\end{verbatim}

### 3.1.2 Project and Documentation Structure

The project and documentation structure and naming is, whenever it is possible, based on the ‘original’ documentation structure and naming of the corresponding library (among others, you will find exceptions, such as for support and utils modules, but these are exceptions and their common denominator is that the functionality they provide is ‘additional’ and (never) part of Glib, GObject or GObject Introspection.

**Glib**

In the table below we list respectively the Glib section, and, when applicable, its G-Golf section and the corresponding G-Golf source code module.

\textsuperscript{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.
### Glib Section

Memory Allocation Section 3.2.1 [Memory Allocation](https://developer.gnome.org/glib/stable/glib-Memory-Allocation.html)

The Main Event Loop Section 3.2.2 [The Main Event Loop](https://developer.gnome.org/glib/stable/glib-The-Main-Event-Loop.html)

### GObject

In the table below we list respectively the GObject section, and, when applicable, its G-Golf section and the corresponding G-Golf source code module.

<table>
<thead>
<tr>
<th>GObject Section</th>
<th>G-Golf Section</th>
<th>G-Golf module</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type Information</td>
<td>Section 3.3.1 <a href="https://developer.gnome.org/gobject/stable/gobject-Type-Information.html">Type Information</a></td>
<td>(g-golf gobject type-info)</td>
</tr>
<tr>
<td>GObject</td>
<td>Section 3.3.2 <a href="https://developer.gnome.org/gobject/stable/gobject-The-Base-Object-Type.html">GObject</a></td>
<td>(g-golf gobject gobject)</td>
</tr>
<tr>
<td>Enumeration and Flag Types</td>
<td>Section 3.3.3 <a href="https://developer.gnome.org/gobject/stable/gobject-Enumeration-and-Flag-Types.html">Enumeration and Flag Types</a></td>
<td>(g-golf gobject enum-flags)</td>
</tr>
<tr>
<td>Parameters and Values</td>
<td>Section 3.3.5 <a href="https://developer.gnome.org/gobject/stable/gobject-Parameters-and-Values.html">Parameters and Values</a> page 26,</td>
<td>(g-golf gobject params-vals)</td>
</tr>
<tr>
<td>GParamSpec</td>
<td>Section 3.3.6 <a href="https://developer.gnome.org/gobject/stable/gobject-GParamSpec.html">GParamSpec</a></td>
<td>(g-golf gobject param-spec)</td>
</tr>
</tbody>
</table>

### GObject Introspection

In the table below we list respectively the GObject Introspection section, and, when applicable, its G-Golf section and the corresponding G-Golf source code module.

<table>
<thead>
<tr>
<th>GI Section</th>
<th>G-Golf Section</th>
<th>G-Golf module</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIRepository</td>
<td>Section 3.5.1 <a href="https://developer.gnome.org/gi/stable/GIRepository.html">Reposi-tory</a></td>
<td>(g-golf gi repository)</td>
</tr>
<tr>
<td>common types</td>
<td>Section 3.5.2 <a href="https://developer.gnome.org/gi/stable/gi-common-types.html">Common</a></td>
<td>(g-golf gi common-types)</td>
</tr>
<tr>
<td>GIBaseInfo</td>
<td>Section 3.5.3 <a href="https://developer.gnome.org/gi/stable/gi-GIBaseInfo.html">Base Info</a></td>
<td>(g-golf gi base-info)</td>
</tr>
</tbody>
</table>
Chapter 3: API Reference

GICallableInfo  Section 3.5.4 [Callable (https://developer.gnome.org/gi/stable/gi-GICallableInfo.html)

GIFunctionInfo  Section 3.5.6 [Function (https://developer.gnome.org/gi/stable/gi-GIFunctionInfo.html)


GIRegisteredTypeInfo  Section 3.5.7 [Registered (https://developer.gnome.org/gi/stable/gi-GIRegisteredTypeInfo.html)


GITypeInfo  Section 3.5.16 [Type (https://developer.gnome.org/gi/stable/gi-GITypeInfo.html)
3.1.3 GOOPS Notes and Conventions

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).

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

**Slots are not Immutable**

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 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.

**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):

- 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 your 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 extremely 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.

---

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 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 Slot names tends to be extremely 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.
• Accessors may always be used to mutate a slot value (except for virtual slots, for which you can ‘block’ that feature), like in \(\text{set! (!name an-actor) "Mike"}\). 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 \text{set!}. Therefore, prefixing makes sense (and preserves the first reason announced here, where postfixing 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.

### 3.2 Glib

G-Golf Glib modules are defined in the \text{glib} subdirectory, such as \text{(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:

\[
\text{(use-modules (g-golf glib))}
\]

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

#### 3.2.1 Memory Allocation

G-Golf Glib Memory Allocation low level API.

Memory Allocation — general memory-handling

**Procedures**

- \text{[g-malloc]}, page 11
- \text{[g-malloc0]}, page 11
- \text{[g-free]}, page 12
- \text{[g-memdup]}, page 12

**Description**

These functions provide support for allocating and freeing memory.


**Procedures**

- \text{g-malloc \text{n-bytes}} \hspace{1cm} [Procedure]
- \text{g-malloc0 \text{n-bytes}} \hspace{1cm} [Procedure]

Returns a pointer to the allocated memory, or \#f.

Allocates \text{n-bytes} of memory. If \text{n-bytes} is 0 it returns \#f. When using \text{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.

### 3.2.2 The Main Event Loop

G-Golf Glib Main Event Loop low level API.  
The Main Event Loop — manages all available sources of events

**Procedures**

- [g-main-loop-new], page 12  
- [g-main-loop-run], page 13  
- [g-main-loop-ref], page 13  
- [g-main-loop-unref], page 13  
- [g-main-loop-quit], page 13  
- [g-main-context-new], page 13  
- [g-main-context-default], page 13  
- [g-timeout-source-new], page 13  
- [g-timeout-source-new-seconds], page 14  
- [g-idle-source-new], page 14  
- [g-source-ref-count], page 14  
- [g-source-ref], page 14  
- [g-source-unref], page 14  
- [g-source-free], page 14  
- [g-source-attach], page 14  
- [g-source-destroy], page 14  
- [g-source-is-destroyed?], page 15  
- [g-source-set-priority], page 15  
- [g-source-get-priority], page 15  
- [g-source-remove], page 15

**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.
Chapter 3: API Reference

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

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

Returns loop.

Increases the loop reference count by one.

**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 13, 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 13, 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 14, 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 14, 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 14, 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 14, 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 14.

**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

[Procedure]

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

[Procedure]

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

[Procedure]

Returns an integer.

Obtains and returns the source priority.

g-source-remove id

[Procedure]

Returns #t.

Removes the source with the given id from the default main context. You must use [g-source-destroy], page 14, 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.

3.2.3 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 16
[g-io-channel-ref], page 16
[g-io-channel-unref], page 16
[g-io-create-watch], page 16

Types and Values

[%g-io-condition], page 16

Description

The GI0Channel 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 \texttt{fd}, \textit{channel} and \textit{condition} arguments are [must be] respectively an integer (a `valid' file descriptor), a pointer to a \texttt{GIOChannel} and a list of one or more \texttt{[%g-io-condition]}, page 16, flags.

\texttt{g-io-channel-unix-new \texttt{fd}} \hfill [Procedure]

Returns a pointer.

Creates and returns a pointer to a new \texttt{GIOChannel} for \texttt{fd} (file descriptor). On UNIX systems this works for plain files, pipes, and sockets.

The newly created \texttt{GIOChannel} has a reference count of 1.

The default encoding for \texttt{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 \texttt{g_io_channel_set_encoding}).

\texttt{g-io-channel-ref \texttt{channel}} \hfill [Procedure]

Returns \texttt{channel}.

Increments the \texttt{channel} reference count.

\texttt{g-io-channel-unref \texttt{channel}} \hfill [Procedure]

Returns nothing.

Decrements the \texttt{channel} reference count.

\texttt{g-io-create-watch \texttt{channel condition}} \hfill [Procedure]

Returns a pointer.

Creates and returns a pointer to a \texttt{GSource} that’s dispatched when condition is met for the given \texttt{channel}. For example, if condition is \texttt{'(in)}, the source will be dispatched when there’s data available for reading.

Types and Values

\texttt{[%g-io-condition}} \hfill [Instance Variable of \texttt{<gi-flag>)}

An instance of \texttt{<gi-flag>}, who’s members are the scheme representation of the \texttt{GIOCondition} flags:

\begin{itemize}
  \item \texttt{gi-name}: GIOCondition
  \item \texttt{scm-name}: gio-condition
  \item \texttt{enum-set}:
    \begin{itemize}
      \item \texttt{in} \hfill There is data to read.
      \item \texttt{out} \hfill Data can be written (without blocking).
      \item \texttt{pri} \hfill There is urgent data to read.
      \item \texttt{err} \hfill Error condition.
      \item \texttt{hup} \hfill Hung up (the connection has been broken, usually for pipes and sockets).
      \item \texttt{nval} \hfill Invalid request. The file descriptor is not open.
    \end{itemize}
\end{itemize}
3.2.4 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 17

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 \textit{fd} and \textit{condition} arguments are [must be] respectively an integer
(a ‘valid’ file descriptor) and a list of one or more \%g-io-condition, page 16, flags.

\texttt{g-unix-fd-source-new fd condition} \hspace{1cm} [Procedure]

Returns a pointer.

Creates and returns a pointer to a new \texttt{GSource} to watch for a particular IO \textit{condition}
on \textit{fd}.

The source will never close the file descriptor, you must do it yourself.

3.2.5 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 18
[g-list-next], page 18
[g-list-prev], page 18
[g-list-free], page 18
[g-list-length], page 18
[g-list-nth-data], page 18

Description

The \texttt{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 \texttt{GSList}, which only allows movement
through the list in the forward direction).

section from the Glib reference manual for a complete description.
Procedures

**g-list-data g-list**

[Procedure]

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**

[Procedure]

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**

[Procedure]

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**

[Procedure]

Returns nothing.

Frees all of the memory used by *g-list*.

**g-list-length g-list**

[Procedure]

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 n**

[Procedure]

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.

### 3.2.6 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 19
- [g-slist-next], page 19
- [g-slist-free], page 19
- [g-slist-length], page 19
- [g-slist-nth-data], page 19

**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 Section 3.2.5 [Doubly-Linked Lists], page 17, 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-free g-slist**
  Returns nothing.
  Frees all of the memory used by g-slist.

- **g-slist-length g-slist**
  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**
  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.

**3.2.7 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 20
- [g-quark-to-string], page 20

**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

\texttt{g-quark-from-string str}  
Returns an integer.

Obtains and returns the \texttt{GQuark} identifying the string given by \texttt{str}. If the string does not currently have an associated \texttt{GQuark}, a new \texttt{GQuark} is created, using a copy of the string.

\texttt{g-quark-to-string g-quark}  
Returns a string.

Obtains and returns the string associated with the \texttt{GQuark} given by \texttt{g-quark}.

3.3 GObject

G-Golf GObject modules are defined in the \texttt{gobject} subdirectory, such as \texttt{(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 \texttt{(oop goops)}, \texttt{(system foreign)}, all G-Golf support and G-Golf GObject modules:

\begin{verbatim}
(use-modules (g-golf gobject))
\end{verbatim}

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

3.3.1 Type Information

G-Golf GObject Type Information low level API.

Type Information — The GLib Runtime type identification and management system

Procedures

\begin{verbatim}
[g-type->symbol], page 21  
[symbol->g-type], page 21  
[g-type-name], page 21  
[g-type-fundamental], page 21
\end{verbatim}

Types and Values

\begin{verbatim}
[%g-type-fundamental-flags], page 21  
[%g-type-fundamental-types], page 22
\end{verbatim}

Object Hierarchy

\begin{verbatim}
gpointer  
\hfill \rightarrow \texttt{GType}
\end{verbatim}

Description

The \texttt{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](https://developer.gnome.org/gobject/stable/gobject-Type-Information.html)) section from the GObject reference manual for a complete description.

### Procedures

**g-type->symbol**  
Returns a symbol.  
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**  
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**  
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-fundamental**  
Returns a type ID.  
Extracts the fundamental type ID portion for `g-type`.

### Types and Values

**%g-type-fundamental-flags**  
Bit masks used to check or determine specific characteristics of a fundamental type.  
An instance of `<gi-enum>`, who’s members are the scheme representation of the `GTypeFundamentalFlags`:

- `gi-name`: `GTypeFundamentalFlags`
- `scm-name`: `g-type-fundamental-flags`
- `enum-set`:

  - `classed`: Indicates a classed type
  - `instantiable`: Indicates an instantiable type (implies classed)
  - `derivable`: Indicates a flat derivable type
  - `deep-derivable`: Indicates a deep derivable type (implies derivable)
A instance of `<gi-enum>`, who’s members are the scheme representation of the `GType` obtained from the fundamental types defined using `G_TYPE_MAKE_FUNDAMENTAL`, which starts with `G_TYPE_INVALID` and ends with `G_TYPE_OBJECT`.

**gi-name:** `#f`

**scm-name:** `g-type-fundamental-types`

**enum-set:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>invalid</td>
<td>An invalid <code>GType</code> used as error return value in some functions which return a <code>GType</code>.</td>
</tr>
<tr>
<td>none</td>
<td>A fundamental type which is used as a replacement for the C void return type.</td>
</tr>
<tr>
<td>interface</td>
<td>The fundamental type from which all interfaces are derived.</td>
</tr>
<tr>
<td>char</td>
<td>The fundamental type corresponding to <code>gchar</code>. It is unconditionally an 8-bit signed integer. This may or may not be the same type as the C type &quot;<code>gchar</code>&quot;.</td>
</tr>
<tr>
<td>uchar</td>
<td>The fundamental type corresponding to <code>guchar</code>.</td>
</tr>
<tr>
<td>boolean</td>
<td>The fundamental type corresponding to <code>gboolean</code>.</td>
</tr>
<tr>
<td>int</td>
<td>The fundamental type corresponding to <code>gint</code>.</td>
</tr>
<tr>
<td>uint</td>
<td>The fundamental type corresponding to <code>guint</code>.</td>
</tr>
<tr>
<td>long</td>
<td>The fundamental type corresponding to <code>glong</code>.</td>
</tr>
<tr>
<td>ulong</td>
<td>The fundamental type corresponding to <code>gulong</code>.</td>
</tr>
<tr>
<td>int64</td>
<td>The fundamental type corresponding to <code>gint64</code>.</td>
</tr>
<tr>
<td>uint64</td>
<td>The fundamental type corresponding to <code>guint64</code>.</td>
</tr>
<tr>
<td>enum</td>
<td>The fundamental type from which all enumeration types are derived.</td>
</tr>
<tr>
<td>flags</td>
<td>The fundamental type from which all flags types are derived.</td>
</tr>
<tr>
<td>float</td>
<td>The fundamental type corresponding to <code>gfloat</code>.</td>
</tr>
<tr>
<td>double</td>
<td>The fundamental type corresponding to <code>gdouble</code>.</td>
</tr>
<tr>
<td>string</td>
<td>The fundamental type corresponding to null-terminated C strings.</td>
</tr>
<tr>
<td>pointer</td>
<td>The fundamental type corresponding to <code>gpointer</code>.</td>
</tr>
<tr>
<td>boxed</td>
<td>The fundamental type from which all boxed types are derived.</td>
</tr>
</tbody>
</table>

---

8 There is no corresponding `enum` in GObject. These fundamental types (in GObject) are defined using a macro, `G_TYPE_MAKE_FUNDAMENTAL`, that applies bitwise arithmetic shift given by `G_TYPE_FUNDAMENTAL_SHIFT` (which we also have to apply, to get to the type ID for the fundamental number x).
param The fundamental type from which all Section 3.3.6 [GParam-Spec], page 29, types are derived.

object The fundamental type for Section 3.3.2 [GObject_], page 23.

3.3.2 GObject

G-Golf GObject low level API.

GObject — The base object type

Procedures

[g-object-new], page 23
[g-object-new-with-properties], page 24
[g-object-ref], page 24
[g-object-unref], page 24
[g-object-ref-sink], page 24
[g-object-ref-count], page 24
[g-object-is-floating], page 24
[g-object-type], page 24
[g-object-type-name], page 24
[g-object-get-property], page 24
[g-object-set-property], page 25

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-new gtype
[Procedure]
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  
\texttt{gtype n-prop names g-values}  
[Procedure]  
Returns a pointer.  
Creates and returns a (pointer to) a new instance of a GObject subtype \texttt{gtype}. The other arguments are \texttt{n-prop} the number of properties, \texttt{names} a pointer to an array of pointers to strings with the names of each property to be set and \texttt{values} an array of \texttt{GValue} containing the values of each property to be set.  
Properties that are not explicitly specified are set to their default values.

\begin{verbatim}
g-object-ref \texttt{object}  
[Procedure]  
Returns a pointer.  
Increases the reference count of \texttt{object}.  
\end{verbatim}

\begin{verbatim}
g-object_unref \texttt{object}  
[Procedure]  
Returns nothing.  
Decreases the reference count of \texttt{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 \texttt{g-clear-object} for this.  
\end{verbatim}

\begin{verbatim}
g-object-ref-sink \texttt{object}  
[Procedure]  
Returns a pointer.  
If \texttt{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 \texttt{object} is not floating, then this call adds a new normal reference increasing the reference count by one.  
\end{verbatim}

\begin{verbatim}
g-object-ref-count \texttt{object}  
[Procedure]  
Returns an integer.  
Obtains and returns the (public GObject struct field) \texttt{ref_count} value for \texttt{object}.  
\end{verbatim}

\begin{verbatim}
g-object-is-floating \texttt{object}  
[Procedure]  
Returns \#t if \texttt{object} has a floating reference, otherwise it returns \#f.  
\end{verbatim}

\begin{verbatim}
g-object-type \texttt{object}  
[Procedure]  
Returns the \texttt{GType} (the type id) for \texttt{object}.  
\end{verbatim}

\begin{verbatim}
g-object-type-name \texttt{object}  
[Procedure]  
Returns the \texttt{GType} name for \texttt{object}.  
\end{verbatim}

\begin{verbatim}
g-object-get-property \texttt{object property \[g-type \#f\]}  
[Procedure]  
Returns the property value for \texttt{object}.  
The property argument is (must be) a pointer to a valid \texttt{GIPROPERTYInfo} (property must point to one of the properties infos of the class of \texttt{object}). The g-type argument
Chapter 3: API Reference

must be a valid GType value. If #f, which is the default, [gi-property-g-type], page 71, is called.

\[
\text{g-object-set-property} \ \text{object} \ \text{property} \ \text{value} \ [g\text{-type} \ #f] \quad \text{[Procedure]}
\]

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 71, is called.

3.3.3 Enumeration and Flag Types

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

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.

3.3.4 Generic Values

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

Procedures

\[
\text{[g-value-size], page 26} \\
\text{[g-value-init], page 26} \\
\text{[g-value-unset], page 26}
\]

Object Hierarchy

GBoxed

\[\rightarrow \text{GValue}\]

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 create a zero-filled GValue structure. To initialize the GValue, use the [g-value-init], page 26, 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-init g-type`  
Returns a pointer to a `GValue`.  
Creates and initializes a `GValue` with the default value for `g-type`.

`g-value-unset g-value`  
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.

3.3.5 Parameters and Values

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

Procedures

[g-value->g-type-id], page 27
[g-value->g-type], page 27
[g-value-ref], page 27
[g-value-set!], page 27
[g-value-get-boolean], page 27
[g-value-set-boolean], page 27
[g-value-get-int], page 27
[g-value-set-int], page 27
[g-value-get-uint], page 28
[g-value-set-uint], page 28
[g-value-get-float], page 28
[g-value-set-float], page 28
[g-value-get-double], page 28
[g-value-set-double], page 28
[g-value-get-enum], page 28
[g-value-set-enum], page 28
[g-value-get-flags], page 28
[g-value-set-flags], page 28
[g-value-get-string], page 29
[g-value-set-string], page 29
[g-value-get-boxed], page 29
[g-value-set-boxed], page 29
[g-value-get-pointer], page 29
[g-value-set-pointer], page 29
[g-value-get-object], page 29
[g-value-set-object], page 29
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 '-' . All other characters are replaced by a '-' during construction.

Procedures and Methods

Note: in this section, the g-value arguments are [must be] pointers to a GValue.

\[ \text{Procedure} \]
\[ \text{g-value->g-type-id} \ g-value \]

Returns a integer.

Gets and returns the GType for g-value.

\[ \text{Procedure} \]
\[ \text{g-value->g-type} \ g-value \]

Returns a symbol.

Gets and returns the scheme representation (a symbol) of the GType for g-value.

\[ \text{Procedure} \]
\[ \text{g-value-ref} \ g-value \]

Returns the content of g-value.

Gets and returns the content of g-value. Supported GType (their scheme representation) for g-value are: boolean, int, uint, float, string, pointer, object.

\[ \text{Procedure} \]
\[ \text{g-value-set!} \ g-value \ \text{value} \]

Returns nothing.

Sets the content of g-value to value. Supported GType (their scheme representation) for g-value are: boolean, int, uint, float, string, pointer, object.

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.

\[ \text{Procedure} \]
\[ \text{g-value-get-boolean} \ g-value \]

Returns #t or #f.

Gets the content of g-value and returns #f if it is 0, otherwise it returns #t.

\[ \text{Procedure} \]
\[ \text{g-value-set-boolean} \ g-value \ \text{val} \]

Returns nothing.

Sets the content of g-value to 0 if val is #f, otherwise sets the content to 1.

\[ \text{Procedure} \]
\[ \text{g-value-get-int} \ g-value \]

Returns a integer.

Gets and returns the content of g-value.

\[ \text{Procedure} \]
\[ \text{g-value-set-int} \ g-value \ \text{int} \]

Returns nothing.
Sets the content of \textit{g-value} to \textit{int}.

\textbf{g-value-get-uint} \textit{g-value} \hfill [Procedure]

Returns an unsigned integer.

Gets and returns the content of \textit{g-value}.

\textbf{g-value-set-uint} \textit{g-value uint} \hfill [Procedure]

Returns nothing.

Sets the content of \textit{g-value} to \textit{uint}.

\textbf{g-value-get-float} \textit{g-value} \hfill [Procedure]

Returns a float.

Gets and returns the content of \textit{g-value}.

\textbf{g-value-set-float} \textit{g-value float} \hfill [Procedure]

Returns nothing.

Sets the content of \textit{g-value} to \textit{float}.

\textbf{g-value-get-double} \textit{g-value} \hfill [Procedure]

Returns a double.

Gets and returns the content of \textit{g-value}.

\textbf{g-value-set-double} \textit{g-value double} \hfill [Procedure]

Returns nothing.

Sets the content of \textit{g-value} to \textit{double}.

\textbf{g-value-get-enum} \textit{g-value} \hfill [Procedure]

Returns a symbol.

Gets and returns the (registered) enum type info symbol for \textit{g-value}.

\textbf{g-value-set-enum} \textit{g-value} (id <integer>) \hfill [Method]
\textbf{g-value-set-enum} \textit{g-value} (sym <symbol>) \hfill [Method]

Returns nothing.

Sets the content of \textit{g-value} to \textit{id}, or to the \textit{id} corresponding to \textit{sym} respectively. The \textit{id} or the \textit{sym} must be valid (as in being a valid member of the (registered) enum type info for \textit{g-value}), otherwise an exception is raised.

\textbf{g-value-get-flags} \textit{g-value} \hfill [Procedure]

Returns a list.

Gets and returns the (registered) list of flags for \textit{g-value}.

\textbf{g-value-set-flags} \textit{g-value} (val <integer>) \hfill [Method]
\textbf{g-value-set-flags} \textit{g-value} (flags <list>) \hfill [Method]

Returns nothing.

Sets the content of \textit{g-value} to \textit{val}, or to the value given by calling \texttt{[gi-gflags->integer]}, page 80, upon the list of \textit{flags}, respectively. The \textit{val} or the \textit{flags} must be valid (as
in being a valid member of the (registered) gflags type for \textit{g-value}), otherwise an exception is raised.

\textbf{g-value-get-string} \textit{g-value}  
\hspace{1em} [Procedure]  
\hspace{1em} Returns a string.  
\hspace{1em} Gets and returns the content of \textit{g-value}.

\textbf{g-value-set-string} \textit{g-value} \textit{str}  
\hspace{1em} [Procedure]  
\hspace{1em} Returns nothing.  
\hspace{1em} Sets the content of \textit{g-value} to \textit{str}.

\textbf{g-value-get-boxed} \textit{g-value}  
\hspace{1em} [Procedure]  
\hspace{1em} Returns either a list of values, or a pointer.  
\hspace{1em} Gets and returns the content of \textit{g-value}. If the boxed type \([\text{!is-opaque?}], \text{page 81}, \text{or } [\text{!is-semi-opaque?}], \text{page 82}\), it \textit{blindingly} returns the boxed instance \textit{g-value} pointer. Otherwise, the boxed instance is \textit{decoded}, and a list of its field values is returned.

\textbf{g-value-set-boxed} \textit{g-value} \textit{boxed}  
\hspace{1em} [Procedure]  
\hspace{1em} Returns nothing.  
\hspace{1em} Sets the content of \textit{g-value} to \textit{boxed}. If the boxed type \([\text{!is-opaque?}], \text{page 81}, \text{or } [\text{!is-semi-opaque?}], \text{page 82}\), then \textit{boxed} is (supposed to be) a pointer, used to \textit{blindingly} set \textit{g-value}. Otherwise, the boxed instance is (supposed to be) a list of values, that are \textit{encoded}, and its (newly created) pointer is used to set \textit{g-value}.

\textbf{g-value-get-pointer} \textit{g-value}  
\hspace{1em} [Procedure]  
\hspace{1em} Returns a pointer.  
\hspace{1em} Gets and returns the content of \textit{g-value}.

\textbf{g-value-set-pointer} \textit{g-value} \textit{pointer}  
\hspace{1em} [Procedure]  
\hspace{1em} Returns nothing.  
\hspace{1em} Sets the content of \textit{g-value} to \textit{pointer}.

\textbf{g-value-get-object} \textit{g-value}  
\hspace{1em} [Procedure]  
\hspace{1em} Returns a pointer.  
\hspace{1em} Gets and returns the content of \textit{g-value}.

\textbf{g-value-set-object} \textit{g-value} \textit{object}  
\hspace{1em} [Procedure]  
\hspace{1em} Returns nothing.  
\hspace{1em} Sets the content of \textit{g-value} to \textit{object} (a pointer to a \texttt{GObject} instance) and increases the \textit{object} reference count.

### 3.3.6 GParamSpec

G-Golf \texttt{GObject} GParamSpec low level API.  
GParamSpec — Metadata for parameter specifications.
Types and Values

[%g-param-flags], page 30

Description

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

Types and Values

%g-param-flags [Instance Variable of <gi-enum>]

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

- type-name: GParamFlags
- scm-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
the parameter is deprecated and will be removed in a future version. A warning will be generated if it is used while running with G_ENABLE_DIAGNOSTIC=1. Since 2.26

3.3.7 Closures
G-Golf GObject Closures low level API.

Closures - Functions as first-class objects

Procedures

[g-closure-size], page 31
[g-closure-ref-count], page 32
[g-closure-ref], page 32
[g-closure-sink], page 32
[g-closure_unref], page 32
[g-closure-free], page 32
[g-closure-invoke], page 32
[g-closure-add-invalidate-notifier], page 32
[g-closure-new-simple], page 33
[g-closure-set-marshal], page 33
[g-source-set-closure], page 33

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.

g-closure-size

[Procedure]

Returns an integer.

Obtains and returns the size (the number of bytes) that a GClosure occupies in memory.
g-closure-ref-count closure
[Procedure]
Returns an integer.
Obtains and returns the reference count of closure.

g-closure-ref closure
[Procedure]
Returns a pointer.
Increments the reference count of closure, to force it staying alive while the caller holds a pointer to it.

g-closure-sink closure
[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 32, 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 32, does nothing.
Because [g-closure-sink], page 32, may decrement the reference count of closure (if it hasn’t been called on closure yet) just like [g-closure-unref], page 32, [g-closure-ref], page 32, should be called prior to this function.

g-closure-unref closure
[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
[Procedure]
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 invocation-hit
[Procedure]
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-hit (a context dependent invocation hint).

g-closure-add-invalidate-notifier closure data function
[Procedure]
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).
Chapter 3: API Reference

3.3.8 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 33
[g-signal-lookup], page 34
[g-signal-list-ids], page 34

Types and Values

[g-signal-flags], page 34

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\textsuperscript{9}, flags, return-type, number of arguments and their types. For example\textsuperscript{10}:

\begin{verbatim}
,use (g-golf)
  (gi-import "Clutter")

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

  (!gtype-id (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))
\end{verbatim}

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 \texttt{GClosure}, callbacks must assume that the instance upon which a signal is called is always the first argument of the callback.

**g-signal-lookup** \texttt{name g-type}  
[Procedure]  
Returns an integer.  
Obtains and returns the signal’s identifying integer, given the \texttt{name} of the signal and the object \texttt{g-type} it connects to. If a signal identifier can’t be find for the given \texttt{name} and \texttt{g-type}, an exception is raised.

**g-signal-list-ids** \texttt{g-type}  
[Procedure]  
Returns a list of integers.  
Obtains and returns the list of signal’s identifying integers for \texttt{g-type} (Note that at least one \texttt{g-type} instance must have been created prior to attempt to list or query signal’s identifying integers for a given \texttt{g-type}).

**Types and Values**

\texttt{\%g-signal-flags}  
[Instance Variable of \texttt{<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 \texttt{<gi-enum>}, who’s members are the scheme representation of the \texttt{GSignalFlags}:

\textsuperscript{9} Within this context, the interface-type is the \texttt{GType} of the \texttt{GObject} subclass the signal is ‘\texttt{attached to}’ - knowing that signals are inhereted.

\textsuperscript{10} At least one \texttt{GObject} subclass instance must have been created prior to attempt to query any of its class signal(s).
3.4 Gdk

G-Golf Gdk modules are defined in the gdk subdirectory, such as (g-golf gdk events).

Where you may load these modules individually, the easiest way to use G-Golf Gdk 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 Gdk modules:

( use-modules (g-golf gdk) )

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

3.4.1 Events

G-Golf Gdk Events low level API.

Events — Functions for handling events from the window system
Class

[<gdk-event>], page 36

Procedures, Accessors and Methods

[!event], page 36
[gdk-event-get-button], page 36
[gdk-event-get-click-count], page 37
[gdk-event-get-coords], page 37
[gdk-event-get-keycode], page 37
[gdk-event-get-keyval], page 37
[gdk-event-get-root-coords], page 37
[gdk-event-get-state], page 37
[gdk-event-get-time], page 37
[gdk-event-get-window], page 37
[gdk-event-get-event-type], page 37

Types and Values

[%gdk-event-type], page 38

Description

This section describes functions dealing with events from the window system.

Classes

<gdk-event>

The superclass of all Gdk type of event. Its slots are:

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

A pointer to a GdkEvent.

Procedures, Accessors and Methods

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

!event inst

[Accessor]

Returns the content of the event slot for inst.

gdk-event:type inst

[Method]

Returns a symbol.

Obtains and returns the event type (symbol) name for inst.

gdk-event-get-button event

[Procedure]

Returns an integer or #f.

Extracts and returns the button number from event. If event did not deliver a button number, it returns #f.
gdk-event-get-click-count event  
Returns an integer or #f.  
Extracts and returns the click count from event. If event did not deliver a click count, it returns #f.

gdk-event-get-coords event  
Returns a list or #f.  
Extracts and returns a list of the x and y window relative coordinates from event. If event did not deliver window coordinates, it returns #f.

gdk-event-get-keycode event  
Returns an integer or #f.  
Extracts and returns the hardware keycode from event. If event did not deliver a hardware keycode, it returns #f.

gdk-event-get-keyval event  
Returns an integer or #f.  
Extracts and returns the keyval from event. If event did not deliver a key symbol, it returns #f.

gdk-event-get-root-coords event  
Returns a list or #f.  
Extracts and returns a list of the x and y root window relative coordinates from event. If event did not deliver root window coordinates, it returns #f.

gdk-event-get-state event  
Returns a (possibly empty) list of GdkModifierType\(^\text{11}\) or #f.  
Obtains and returns the list of (the scheme representation of) GdkModifierType for event. If event did not contain a ‘state’ field, it returns #f.

gdk-event-get-time event  
Returns an integer.  
Obtains and returns the time stamp for event, if there is one, otherwise, it returns GDK_CURRENT_TIME (and so does it if event is %null-pointer).

gdk-event-get-window event  
Returns a pointer.  
Extracts and returns (a pointer to) the GdkWindow associated with event.

gdk-event-get-event-type event  
Returns the event type name.  
Obtains and returns the [%gdk-event-type], page 38, (symbol) name for event.

---

\(^\text{11}\) G-Golf imports the GdkModifierType from the Gdk namespace as a <gi-flag> instance, which you may get executing (gi-cache-ref 'flag 'gdk-modifier-type), then visualize running describe on the former result. Please refer to the enum GdkModifierType (https://developer.gnome.org/gdk3/stable/gdk3-Windows.html#GdkModifierType) section of the Gdk Reference Manual for a complete list and description of all modifier bit-flags.
Types and Values

%gdk-event-type

Specifies the type of the event.

Do not confuse these events with the signals that GTK+ widgets emit. Although many of these events result in corresponding signals being emitted, the events are often transformed or filtered along the way.

An instance of <gi-ennum>, who’s members are the scheme representation of the GdkEventType.

\textit{gi-name}: GdkEventType
\textit{scm-name}: gdk-event-type
\textit{enum-set}:

\begin{itemize}
  \item \texttt{nothing} A special code to indicate a null event.
  \item \texttt{delete} The window manager has requested that the toplevel window be hidden or destroyed, usually when the user clicks on a special icon in the title bar.
  \item \texttt{destroy} The window has been destroyed.
  \item \texttt{expose} All or part of the window has become visible and needs to be redrawn.
  \item \texttt{motion-notify} The pointer (usually a mouse) has moved.
  \item \texttt{button-press} A mouse button has been pressed.
  \item \texttt{2button-press} A mouse button has been double-clicked (clicked twice within a short period of time). Note that each click also generates a button-press event.
  \item \texttt{double-button-press} Alias for 2button-press, added in 3.6.
  \item \texttt{3button-press} A mouse button has been clicked 3 times in a short period of time. Note that each click also generates a button-press event.
  \item \texttt{triple-button-press} Alias for 3button-press, added in 3.6.
  \item \texttt{button-release} A mouse button has been released.
  \item \texttt{key-press} A key has been pressed.
\end{itemize}
key-release
  A key has been released.

enter-notify
  The pointer has entered the window.

leave-notify
  The pointer has left the window.

focus-change
  The keyboard focus has entered or left the window.

configure
  The size, position or stacking order of the window has changed. Note that GTK+ discards these events for window-child windows.

map
  The window has been mapped.

unmap
  The window has been unmapped.

property-notify
  A property on the window has been changed or deleted.

selection-clear
  The application has lost ownership of a selection.

selection-request
  Another application has requested a selection.

selection-notify
  A selection has been received.

proximity-in
  An input device has moved into contact with a sensing surface (e.g. a touchscreen or graphics tablet).

proximity-out
  An input device has moved out of contact with a sensing surface.

drag-enter
  The mouse has entered the window while a drag is in progress.

drag-leave
  The mouse has left the window while a drag is in progress.

drag-motion
  The mouse has moved in the window while a drag is in progress.

drag-status
  The status of the drag operation initiated by the window has changed.

drop-start
  A drop operation onto the window has started.
The drop operation initiated by the window has completed.

A message has been received from another application.

The window visibility status has changed.

The scroll wheel was turned

The state of a window has changed. See GdkWindowState for the possible window states

A setting has been modified.

The owner of a selection has changed. This event type was added in 2.6

A pointer or keyboard grab was broken. This event type was added in 2.8.

The content of the window has been changed. This event type was added in 2.14.

A new touch event sequence has just started. This event type was added in 3.4.

A touch event sequence has been updated. This event type was added in 3.4.

A touch event sequence has finished. This event type was added in 3.4.

A touch event sequence has been canceled. This event type was added in 3.4.

A touchpad swipe gesture event, the current state is determined by its phase field. This event type was added in 3.18.

A touchpad pinch gesture event, the current state is determined by its phase field. This event type was added in 3.18.

A tablet pad button press event. This event type was added in 3.22.
pad-button-release
A tablet pad button release event. This event type was added in 3.22.

pad-ring
A tablet pad axis event from a "ring". This event type was added in 3.22.

pad-strip
A tablet pad axis event from a "strip". This event type was added in 3.22.

pad-group-mode
A tablet pad group mode change. This event type was added in 3.22.

event-last
marks the end of the GdkEventType enumeration. Added in 2.18

3.4.2 Event Structures
G-Golf Gdk Events Structures low level API.
Events Structures — Data structures specific to each type of event

Class

[gdk-event-key], page 42
[gdk-event-button], page 42
[gdk-event-motion], page 42
Procedures and Methods

[make-gdk-event], page 42
[make-gdk-event-key:window], page 43
[make-gdk-event-key:time], page 43
[make-gdk-event-key:state], page 43
[make-gdk-event-key:keydown], page 43
[make-gdk-event-key:keyname], page 43
[make-gdk-event-key:hardware-keycode], page 43
[make-gdk-event-button:time], page 43
[make-gdk-event-button:state], page 43
[make-gdk-event-button:button], page 43
[make-gdk-event-button:click-count], page 43
[make-gdk-event-button:coords], page 43
[make-gdk-event-button:x], page 43
[make-gdk-event-button:y], page 43
[make-gdk-event-button:root-coords], page 43
[make-gdk-event-button:x-root], page 43
[make-gdk-event-button:y-root], page 43
[make-gdk-event-motion:time], page 44
[make-gdk-event-motion:state], page 44
[make-gdk-event-motion:x], page 44
[make-gdk-event-motion:y], page 44
[make-gdk-event-motion:root-coords], page 44
[make-gdk-event-motion:x-root], page 44
[make-gdk-event-motion:y-root], page 44

Description
The event structures contain data specific to each type of event in GDK.

Classes

<gdk-event-key> ([<gdk-event>], page 36)  [Class]
Describes a key press or key release event.

<gdk-event-button> ([<gdk-event>], page 36)  [Class]
Describes a button-press, button-release, 2button-press, 2button-release, 3button-press and 3button-release event.

<gdk-event-motion> ([<gdk-event>], page 36)  [Class]
Describes a motion-notify event.

Procedures and Methods

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

make-gdk-event event  [Procedure]
Returns an instance.
Creates and returns an instance of the gdk event subclass for event, according to its event type.
gdk-event-key:window (inst <gdk-event-key>)  [Method]
gdk-event-key:time (inst <gdk-event-key>)  [Method]
gdk-event-key:state (inst <gdk-event-key>)  [Method]
gdk-event-key:keydown (inst <gdk-event-key>)  [Method]
gdk-event-key:keyup (inst <gdk-event-key>)  [Method]
gdk-event-key:hardware-keycode (inst <gdk-event-key>)  [Method]

Respectively returns the scheme representation of the content of the inst event key element, refered to by its name - the part of the method name that follows the semi-colon, which are:

**window**

A pointer to the *GdkWindow* that received the event.

**time**

The time of the event in milliseconds.

**state**

A list representing the state of the modifier keys (e.g. Control, Shift and Alt) and the pointer buttons\(^\text{12}\) Please refer to the enum *GdkModifierType* ([https://developer.gnome.org/gdk3/stable/gdk3-Windows.html#GdkModifierType](https://developer.gnome.org/gdk3/stable/gdk3-Windows.html#GdkModifierType)) section of the Gdk Reference Manual for a complete list and description of all modifier bit-flags.

**keyval**

The key value (an unsigned integer) that was pressed or released.

**keyname**

The key (symbol) name as returned by calling *(gdk-keyval-name keyval)*.

**hardware-keycode**

The raw code of the key that was pressed or released.

---

\(^{12}\) The *GdkModifierType* is obtained by introspection, from the "Gdk" namespace, when the (g-golf) module is imported (used). You may obtain its *<gi-flag>* instance doing *(gi-cache-ref 'flag 'gdk-modifier-type)*, and visualize its content using *describe*. 
**time**

The time of the event in milliseconds.

**state**

A list representing the state of the modifier keys (e.g., Control, Shift and Alt) and the pointer buttons. Please refer to the enum GdkModifierType (https://developer.gnome.org/gdk3/stable/gdk3-Windows.html#GdkModifierType) section of the Gdk Reference Manual for a complete list and description of all modifier bit-flags.

**button**

The button number of the event.

**click-count**

The click-count of the event.

**coords**

The list of the x and y window relative coordinates of the event.

**x**

The x window relative coordinate of the event.

**y**

The y window relative coordinate of the event.

**root-coords**

The list of the x and y root window relative coordinates of the event.

**x-root**

The x root window relative coordinate of the event.

**y-root**

The y root window relative coordinate of the event.

gdk-event-motion:time (inst <gdk-event-motion>) [Method]
gdk-event-motion:state (inst <gdk-event-motion>) [Method]
gdk-event-motion:coords (inst <gdk-event-motion>) [Method]
gdk-event-motion:x (inst <gdk-event-motion>) [Method]
gdk-event-motion:y (inst <gdk-event-motion>) [Method]
gdk-event-motion:root-coords (inst <gdk-event-motion>) [Method]
gdk-event-motion:x-root (inst <gdk-event-motion>) [Method]
gdk-event-motion:y-root (inst <gdk-event-motion>) [Method]

Respectively returns the scheme representation of the content of the inst event motion element, referred to by its name - the part of the method name that follows the semi-colon, which are:

**time**

The GdkModifierType is obtained by introspection, from the "Gdk" namespace, when the (g-golf) module is imported (used). You may obtain its <gi-flag> instance doing (gi-cache-ref 'flag 'gdk-modifier-type), and visualize its content using describe.
The time of the event in milliseconds.

**state**

A list representing the state of the modifier keys (e.g., Control, Shift and Alt) and the pointer buttons. Please refer to the enum GdkModifierType (https://developer.gnome.org/gdk3/stable/gdk3-Windows.html#GdkModifierType) section of the Gdk Reference Manual for a complete list and description of all modifier bit-flags.

**coords**

The list of the x and y window relative coordinates of the event.

**x**

The x window relative coordinate of the event.

**y**

The y window relative coordinate of the event.

**root-coords**

The list of the x and y root window relative coordinates of the event.

**x-root**

The x root window relative coordinate of the event.

**y-root**

The y root window relative coordinate of the event.

### 3.4.3 Key Values

G-Golf Gdk Key Values low level API.

**Key Values — Functions for manipulating keyboard codes**

**Procedures**

[gdk-keyval-name], page 46

**Struct Hierarchy**

GObject
  ├── GdkKeymap

**Description**

Key values are the codes which are sent whenever a key is pressed or released. They appear in the GdkEventKey.keyval field of the GdkEventKey structure, which is passed to signal handlers for the 'key-press-event' and 'key-release-event' signals.

---

14 The GdkModifierType is obtained by introspection, from the "Gdk" namespace, when the (g-golf) module is imported (used). You may obtain its <gi-flag> instance doing (gi-cache-ref 'flag 'gdk-modifier-type), and visualize its content using describe.

**Procedures**

**gdk-keyval-name keyval**

Returns a symbol or #f.

Obtains and returns the symbol name for `keyval`. If `keyval` is not a valid key, it returns #f.

### 3.5 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 (manua- nal) section, but there are some exceptions, such as init and utils ...  

#### 3.5.1 Repository

G-Golf Introspection Repository low level API.

**GIRepository** — GObject Introspection repository manager.

**Procedures**

- [g-irepository-get-default], page 47
- [g-irepository-get-dependencies], page 47
- [g-irepository-get-loaded-namespaces], page 47
- [g-irepository-get-n-infos], page 47
- [g-irepository-get-info], page 47
- [g-irepository-get-typelib-path], page 47
- [g-irepository-require], page 47
- [g-irepository-get-c-prefix], page 48
- [g-irepository-get-shared-library], page 48
- [g-irepository-get-version], page 48
- [g-irepository-find-by-gtype], page 48
- [g-irepository-find-by-name], page 48

**Object Hierarchy**

- GObject
  - GIRepository
Description

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

Procedures

Note: in this section, the repository optional keyword argument is [must be] a pointer to a GIRepository or %null-pointer.

**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 %null-pointer, meaning this singleton process-global default GIRepository.

**g-irepository-get-dependencies namespace #:key repository**

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 #:key repository**

Return the list of currently loaded namespaces.

**g-irepository-get-n-infos namespace #:key repository**

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 #:key repository**

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-get-typelib-path namespace #:key repository**

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 #:key version repository**

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 #:key repository  
[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 #:key repository  
[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 #:key repository  
[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 #:key repository  
[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 #:key repository  
[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.

### 3.5.2 Common Types

G-Golf Common Types low level API.

common types - TODO

#### Types and Values

[%gi-type-tag], page 48  
[%gi-array-type], page 49

**%gi-type-tag**  
[Instance Variable of <gi-enum>]  
An instance of <gi-enum>, who’s members are the type tag of a GITypeInfo:

- *gi-name*: GITypeTag  
- *scm-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

%gi-array-type

[Instance Variable of <gi-enum>]
An instance of <gi-enum>, who’s members are the type of array in a GTypeInfo:

\[ gi-name: GIArrayType \\
\text{scm-name: } \text{gi-array-type} \\
\text{enum-set:} \]

\text{c} \\
array \\
ptr-array \\
byte-array

### 3.5.3 Base Info

G-Golf Base Info low level API.

GIBaseInfo — Base struct for all GITypelib structs.
Chapter 3: API Reference

Procedures

[g-base-info-ref], page 50
[g-base-info-unref], page 51
[g-base-info-equal], page 51
[g-base-info-get-type], page 51
[g-base-info-get-typelib], page 51
[g-base-info-get-namespace], page 51
[g-base-info-get-name], page 51
[g-base-info-get-attribute], page 51
[g-base-info-iterate-attributes], page 51
[g-base-info-get-container], page 51
[g-base-info-is-deprecated], page 51

Types and Values

[%gi-info-type], page 52

Struct Hierarchy

GIBaseInfo
  +— GIArgInfo
  +— GICallableInfo
  +— GIConstantInfo
  +— GIFieldInfo
  +— GIPropertyInfo
  +— 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 51, has to be called when done accessing the data. GIBaseInfos are normally accessed by calling either [g-repository-find-by-name], page 48, [g-repository-find-by-gtype], page 48, or [g-repository-get-info], page 47.

Example: Getting the Button of the Gtk typelib

(use (g-golf gi)
  (g-repository-require "Gtk")
  (g-repository-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.
**Chapter 3: API 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 instances 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

%gi-info-type

[Instance Variable of <gi-enum>]

An instance of <gi-enum>, who’s members are the scheme representation of the type of a GIBaseInfo struct:

gi-name: GIInfoType

scm-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

3.5.4 Callable Info

G-Golf Callable Info low level API.

GICallableInfo — Struct representing a callable.

Procedures

[g-callable-info-get-n-args], page 53
[g-callable-info-get-arg], page 53
[g-callable-info-get-instance-ownership-transfer], page 53
[g-callable-info-get-caller-owns], page 53
[g-callable-info-get-return-type], page 53
[g-callable-info-may-return-null], page 53

Struct Hierarchy

GIBaseInfoInfo
  +— GICallableInfo
    +— GIFunctionInfo
Description

GICallableInfo represents an entity which is callable. Currently a function (GIFunctionInfo), virtual function, (GIVFuncInfo) or callback (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 n\text{th} GIArgInfo of info.
  
  It must be freed by calling [g-base-info-unref], page 51, when done accessing the data.

- **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 68, for the list of possible values.

- **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 68, 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 51, when done accessing the data.

- **g-callable-info-may-return-null** info
  
  Returns \#t if the callable info could return NULL.
  
  See if a callable could return NULL.

3.5.5 Signal Info

G-Golf Signal Info low level API.

GISignalInfo — Struct representing a signal.

Procedures

- [g-signal-info-get-flags], page 54
Description
GISignalInfo represents a signal. It’s a sub-struct of GICallableInfo and contains a set of flags and a class closure.

See also Section 3.5.4 [Callable Info], page 52, 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.

\texttt{g-signal-info-get-flags info} \text{ [Procedure]}

Returns a list of [\%g-signal-flags], page 34.

Obtain the flags for this signal info. See [\%g-signal-flags], page 34, for more information about possible flag values.

3.5.6 Function Info

G-Golf Function Info low level API.
GIFunctionInfo — Struct representing a function.

Procedures

[is-method?], page 55
[g-function-info-get-flags], page 55
[g-function-info-get-property], page 55
[g-function-info-get-symbol], page 55
[g-function-info-get-vfunc], page 55
[g-function-info-invoke], page 55

Types and Values

[\%g-function-info-flags], page 56

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 55.
See also Section 3.5.4 [Callable Info], page 52, 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`.

**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 55.

**g-function-info-get-flags**  
`info`  
[Procedure]  
Returns a list of [%g-function-info-flags], page 56.  
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 51, when done.

**g-function-info-get-symbol**  
`info`  
[Procedure]  
Returns a string.  
Obtain the ‘symbol’ of the function\(^{15}\).

**g-function-info-get-vfunc**  
`info`  
[Procedure]  
Returns a pointer or `#f`.  
Obtains the GIVFUNCINFO associated with `info`. Only GIFunctionInfo with the flag `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 51, when done.

**g-function-info-invoke**  
`info in-args n-in out-args n-out r-val g-error`  
[Procedure]  
Returns `#t` if the function has been invoked, `#f` if an error occurred.  
Invokes the function described in `info` with the given arguments. Note that `inout` parameters must appear in both argument lists. The arguments are:

- `info`: a pointer to a GIFunctionInfo describing the function to invoke.
- `in-args`: a pointer to an array of GIArguments, one for each `in` and `inout` parameter of `info`. If there are no `in` parameter, `in-args` must be the %null-pointer.

---

\(^{15}\) As you have noticed already, since `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, ‘suitable to be used as an argument to g_module_symbol()’
the length of the \textit{in-args} array.

\textbf{out-args} \hspace{1em} a pointer to an array of \texttt{GIArguments}, one for each \texttt{out} and \texttt{inout} parameter of \texttt{info}. If there are no \texttt{out} parameter, \texttt{out-args} must be the \texttt{%null-pointer}.

\textbf{n-out} \hspace{1em} the length of the \texttt{out-args} array.

\textbf{r-val} \hspace{1em} a pointer to a \texttt{GIArguments}, the return location for the return value of the function. If the function returns \texttt{void}, \texttt{r-val} must be the \texttt{%null-pointer}.

\textbf{g-error} \hspace{1em} a pointer to a newly allocated (and ‘empty’) \texttt{GError} (the recommended way for procedure calls that need such a pointer is to ‘surround’ the call using \texttt{[with-gerror]}, page 75).

\section*{Types and Values}

\%g-function-info-flags \hspace{1em} [Instance Variable of \texttt{<gi-flag>}]

An instance of \texttt{<gi-flag>}, who’s members are the scheme representation of the \texttt{GIFunctionInfoFlags}:

\begin{itemize}
  \item \texttt{gi-name}: \texttt{GIFunctionInfoFlags}
  \item \texttt{scm-name}: \texttt{gi-function-info-flags}
  \item \texttt{enum-set}:
    \begin{itemize}
      \item \texttt{is-method} \hspace{1em} Is a method.
      \item \texttt{is-constructor} \hspace{1em} Is a constructor.
      \item \texttt{is-getter} \hspace{1em} Is a getter of a \texttt{GIPROPERTYINFO}.
      \item \texttt{is-setter} \hspace{1em} Is a setter of a \texttt{GIPROPERTYINFO}.
      \item \texttt{wraps-vfunc} \hspace{1em} Represent a virtual function.
      \item \texttt{throws} \hspace{1em} The function may throw an error.
    \end{itemize}
\end{itemize}

\subsection*{3.5.7 Registered Type Info}

G-Golf Registered Type Info low level API.

\texttt{GIRegisteredTypeInfo} — Struct representing a struct with a \texttt{GType}.

\section*{Procedures}

\begin{itemize}
  \item \texttt{[g-registered-type-info-get-type-name]}, page 57
  \item \texttt{[g-registered-type-info-get-type-init]}, page 57
  \item \texttt{[g-registered-type-info-get-g-type]}, page 57
\end{itemize}
Structured Hierarchy

GIBaseInfo
  +---GIRegisteredTypeInfo
    +---GIEnumInfo
    +---GIInterfaceInfo
    +---GIOBJECTInfo
    +---GITypeInfo
    +---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.

**g-registered-type-info-get-type-name** info
  Returns the type name.
  Obtain the type name of the struct within the GObject type system. This type can be passed to g_type_name() to get a GType.

**g-registered-type-info-get-type-init** info
  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
  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.

3.5.8 Enum Info

G-Golf Enum Info low level API.

GIEnumInfo — Structs representing an enumeration and its values.

Procedures

[gi-enum-import], page 58
[gi-enum-value-values], page 58
[g-enum-info-get-n-values], page 58
[g-enum-info-get-value], page 58
[g-enum-info-get-n-methods], page 58
[g-enum-info-get-method], page 58
[g-value-info-get-value], page 59
**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 59: just like a `GIStructInfo` holds a list of pointers to `GIFieldInfo` from which we get the (field) type, a `GIEnumInfo` 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.

- **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 \textit{info}

Returns the enumeration value.

Obtains a value of the GIValueInfo.
\textit{info} is [must be] a pointer to a GIValueInfo.

3.5.9 Struct Info

G-Golf Struct Info low level API.
GIStructInfo — Structs representing a C structure.

Procedures

[gi-struct-import], page 59
[gi-struct-field-types], page 59
[g-struct-info-get-alignment], page 59
[g-struct-info-get-size], page 60
[g-struct-info-is-gtype-struct], page 60
[g-struct-info-is-foreign], page 60
[g-struct-info-get-n-fields], page 60
[g-struct-info-get-field], page 60
[g-struct-info-get-n-methods], page 60
[g-struct-info-get-method], page 60

Struct Hierarchy

GIBaseInfo
  +— GIRegisteredTypeInfo
    +— GIStructInfo

Description

GIStructInfo represents a generic C structure type.

A structure has methods and fields.

Procedures

Note: in this section, unless otherwise specified, the \textit{info} argument is [must be] a pointer to a GIStructInfo.

\textbf{gi-struct-import} \textit{info} \hspace{1cm} [Procedure]

Returns a <gi-struct> instance.

Obtains the list of (field) types the C struct GI definition pointed by \textit{info} contains, then makes and returns a <gi-struct> instance.

\textbf{gi-struct-field-types} \textit{info} \hspace{1cm} [Procedure]

Returns a list.

Obtains and returns the list of (field) types the C struct GI definition pointed by \textit{info} contains.

\textbf{g-struct-info-get-alignment} \textit{info} \hspace{1cm} [Procedure]

Returns an integer.
Obtains and returns the required alignment for info.

\textbf{g-struct-info-get-size info} \hspace{1cm} \text{[Procedure]}
\begin{itemize}
  \item Returns an integer.
  \item Obtains and returns the total size of the structure specified info.
\end{itemize}

\textbf{g-struct-info-is-gtype-struct info} \hspace{1cm} \text{[Procedure]}
\begin{itemize}
  \item Returns #t or #f.
  \item Return true if the structure specified by info represents the "class structure" for some GObject or GInterface.
\end{itemize}

\textbf{g-struct-info-is-foreign info} \hspace{1cm} \text{[Procedure]}
\begin{itemize}
  \item Returns #t or #f.
  \itemFIXME. No upstream documentation, though the procedure works.
\end{itemize}

\textbf{g-struct-info-get-n-fields info} \hspace{1cm} \text{[Procedure]}
\begin{itemize}
  \item Returns an integer.
  \item Obtains the number of fields for info.
\end{itemize}

\textbf{g-struct-info-get-field info n} \hspace{1cm} \text{[Procedure]}
\begin{itemize}
  \item Returns a pointer.
  \item Obtains and returns the info type information (a pointer to a GIFieldInfo) for the field at the specified n index.
  \item The GIFieldInfo must be freed by calling [g-base-info-unref], page 51, when done.
\end{itemize}

\textbf{g-struct-info-get-n-methods info} \hspace{1cm} \text{[Procedure]}
\begin{itemize}
  \item Returns an integer.
  \item Obtains the number of methods for info.
\end{itemize}

\textbf{g-struct-info-get-method info n} \hspace{1cm} \text{[Procedure]}
\begin{itemize}
  \item Returns a pointer.
  \item Obtains and returns the info type information (a pointer to a GIFunctionInfo) for the method at the specified n index.
  \item The GIFunctionInfo must be freed by calling [g-base-info-unref], page 51, when done.
\end{itemize}

\subsection*{3.5.10 Union Info}
G-Golf Union Info low level API.
GIUnionInfo — Struct representing a C union.
Chapter 3: API Reference

Procedures

[g-union-info-get-n-fields], page 61
[g-union-info-get-field], page 61
[g-union-info-get-n-methods], page 61
[g-union-info-get-method], page 61
[g-union-info-is-discriminated?], page 61
[g-union-info-get-discriminator-offset], page 62
[g-union-info-get-discriminator-type], page 62
[g-union-info-get-discriminator], page 62
[g-union-info-get-size], page 62
[g-union-info-get-alignment], page 62

Description

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

GBaseInfo
  +— GIRegisteredTypeInfo
    +— GIUnionInfo

Procedures

Note: in this section, unless otherwise specified, the info argument is [must be] a pointer to a GIUnionInfo.

g-union-info-get-n-fields info
    Returns an integer.
    Obtains and returns the number of fields the info union has.

[g-union-info-get-field], page 61

[g-union-info-get-n-methods], page 61

[g-union-info-get-method], page 61

[g-union-info-is-discriminated?], page 61

[g-union-info-get-discriminator-offset], page 62

[g-union-info-get-discriminator-type], page 62

[g-union-info-get-discriminator], page 62

[g-union-info-get-size], page 62

[g-union-info-get-alignment], page 62
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 GTypeInfo for info, which must be free’d by calling [g-base-info-unref], page 51, 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 51, 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.

---

### 3.5.11 Object Info
G-Golf Object Info low level API.
GIOBJECTINFO — Structs representing a GObject.
Procedures

[g-object-info-get-abstract], page 63
[g-object-info-get-parent], page 63
[g-object-info-get-type-name], page 64
[g-object-info-get-type-init], page 64
[g-object-info-get-n-constants], page 64
[g-object-info-get-constant], page 64
[g-object-info-get-n-fields], page 64
[g-object-info-get-field], page 64
[g-object-info-get-n-interfaces], page 64
[g-object-info-get-interface], page 64
[g-object-info-get-n-methods], page 64
[g-object-info-get-method], page 64
[g-object-info-find-method], page 64
[g-object-info-get-n-properties], page 65
[g-object-info-get-property], page 65
[g-object-info-get-n-signals], page 65
[g-object-info-get-signal], page 65
[g-object-info-find-signal], page 65
[g-object-info-get-n-vfuncs], page 65
[g-object-info-get-vfunc], page 65
[g-object-info-get-class-struct], page 65

Struct Hierarchy

GIBaseInfo
   — GIRegisteredTypeInfo
      — GIOBJECTInfo

Description

GIOBJECTInfo represents a GObject (https://developer.gnome.org/gobject/stable/gobject-The-Base-Object-Type.html#GObject-struct). 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.

[g-object-info-get-abstract info] [Procedure]
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] [Procedure]
Returns a pointer to the info’s parent GIOBJECTInfo.
Obtain the parent of the object type.
**g-object-info-get-type-name info**

[Procedure]

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**

[Procedure]

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**

[Procedure]

Returns the number of constants for *info*.

Obtain the number of constants that this object type has.

---

**g-object-info-get-constant info n**

[Procedure]

Returns a pointer to the *n*th GIConstantInfo of *info*.

It must be freed by calling [g-base-info-unref], page 51, when done accessing the data.

---

**g-object-info-get-n-fields info**

[Procedure]

Returns the number of fields for *info*.

Obtain the number of fields that this object type has.

---

**g-object-info-get-field info n**

[Procedure]

Returns a pointer to the *n*th GIFieldInfo of *info*.

It must be freed by calling [g-base-info-unref], page 51, when done accessing the data.

---

**g-object-info-get-n-interfaces info**

[Procedure]

Returns the number of interfaces for *info*.

Obtain the number of interfaces that this object type has.

---

**g-object-info-get-interface info n**

[Procedure]

Returns a pointer to the *n*th GIInterfaceInfo of *info*.

It must be freed by calling [g-base-info-unref], page 51, when done accessing the data.

---

**g-object-info-get-n-methods info**

[Procedure]

Returns the number of methods for *info*.

Obtain the number of methods that this object type has.

---

**g-object-info-get-method info n**

[Procedure]

Returns a pointer to the *n*th GIFunctionInfo of *info*.

It must be freed by calling [g-base-info-unref], page 51, when done accessing the data.

---

**g-object-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 51, when done accessing the data.
g-object-info-get-n-properties info

[Procedure]

Returns the number of properties for info.

Obtain the number of properties that this object type has.

---

g-object-info-get-property info n

[Procedure]

Returns a pointer to the nth GIPROPERTYInfo of info.

It must be freed by calling [g-base-info-unref], page 51, when done accessing the data.

---

g-object-info-get-n-signals info

[Procedure]

Returns the number of signals for info.

Obtain the number of signals that this object type has.

---

g-object-info-get-signal info n

[Procedure]

Returns a pointer to the nth GISIGNALInfo of info.

It must be freed by calling [g-base-info-unref], page 51, when done accessing the data.

---

g-object-info-find-signal info name

[Procedure]

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 51, when done accessing the data.

---

g-object-info-get-n-vfuncs info

[Procedure]

Returns the number of vfuncs for info.

Obtain the number of vfuncs that this object type has.

---

g-object-info-get-vfunc info n

[Procedure]

Returns a pointer to the nth GIVFUNCInfo of info.

It must be freed by calling [g-base-info-unref], page 51, when done accessing the data.

---

g-object-info-get-class-struct info

[Procedure]

Returns a pointer to the nth GIStructInfo of info, or #f.

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 51, when done accessing the data.

---

### 3.5.12 Arg Info

G-Golf Arg Info low level API.

GIArgInfo — Struct representing an argument.
Procedures

[g-arg-info-get-closure], page 66
[g-arg-info-get-destroy], page 66
[g-arg-info-get-direction], page 66
[g-arg-info-get-ownership-transfer], page 66
[g-arg-info-get-scope], page 67
[g-arg-info-get-type], page 67
[g-arg-info-may-be-null], page 67
[g-arg-info-is-caller-allocates], page 67
[g-arg-info-is-optional], page 67
[g-arg-info-is-return-value], page 67
[g-arg-info-is-skip], page 67

Types and Values

[%gi-direction], page 67
[%gi-scope-type], page 68
[%gi-transfer], page 68

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 67, of the argument.

g-arg-info-get-ownership-transfer info  [Procedure]
  Returns a symbol.
  Obtains and returns the [%gi-transfer], page 68, for this argument.
g-arg-info-get-scope info
Returns a symbol.
Obtains and returns the [%gi-scope-type], page 68, 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
Returns a pointer.
Obtains the GITypeInfo holding the type information for info. Free it using [g-base-info-unref], page 51, when done.

g-arg-info-may-be-null info
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
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
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.

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:

  gi-name: GIDirection
  scm-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.

  gi-name: GIScopeType
  scm-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:

  gi-name: GITransfer
  scm-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
transfer everything, e.g., 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.

### 3.5.13 Constant Info

G-Golf Constant Info low level API.

**GIConstantInfo** — Struct representing a constant.

**Procedures**

- [g-constant-info-free-value], page 69
- [g-constant-info-get-type], page 69
- [g-constant-info-get-value], page 69

**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 69, and a value, which can be obtained by calling [g-constant-info-get-value], page 69.

**Procedures**

Note: in this section, the *info* and *value* arguments are [must be] pointers to a **GIConstantInfo** and a **GIArgument**, respectively.

- **g-constant-info-free-value** *info* *value*  
  Returns nothing.  
  Frees the value returned from [g-constant-info-get-value], page 69.

- **g-constant-info-get-type** *info*  
  Returns a pointer.  
  Obtains and returns a pointer to the **GITypeInfo** for *info*. Free it using [g-base-info-unref], page 51, when done.

- **g-constant-info-get-value** *info* *value*  
  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 69.
3.5.14 Field Info
G-Golf Field Info low level API.
GIFieldInfo — Struct representing a struct or union field.

Procedures
[g-field-info-get-type], page 70

Struct Hierarchy
GIBaseInfo
+— GIFieldInfo

Description
A GIFieldInfo struct represents a field of a struct (see Section 3.5.9 [Struct Info], page 59),
union (see GIUnionInfo) or an object (see Section 3.5.11 [Object Info], page 62). The
GIFieldInfo is fetched by calling [g-struct-info-get-field], page 60, g-union-info-get-field
or [g-object-info-get-field], page 64. A field has a size, type and a struct offset associated
and a set of flags, which is 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-type info
Returns a pointer. [Procedure]

Obtains and returns the GITypeInfo for info.

The GITypeInfo must be freed by calling [g-base-info-unref], page 51, when done.

3.5.15 Property Info
G-Golf Property Info low level API.
GIPropertyInfo — Struct representing a property.

Procedures
[gi-property-g-type], page 71
[g-property-info-get-flags], page 71
[g-property-info-get-ownership-transfer], page 71
[g-property-info-get-type], page 71

Struct Hierarchy
GIBaseInfoInfo
+— GIPropertyInfo

Description
GIPropertyInfo 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.

\textbf{gi-property-g-type info} \hfill [Procedure]

Returns an integer.

Obtains and returns the GType value of the property.

\textbf{g-property-info-get-flags info} \hfill [Procedure]

Returns a list of [%g-param-flags], page 30.

Obtain the flags for this property info. See Section 3.3.6 [GParamSpec], page 29, for the list of possible flag values.

\textbf{g-property-info-get-ownership-transfer info} \hfill [Procedure]

Returns the ownership transfer for this property.

Obtain the ownership transfer for this property. See [%gi-transfer], page 68, for more information about transfer values.

\textbf{g-property-infoxs-get-type info} \hfill [Procedure]

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.

3.5.16 Type Info

G-Golf Type Info low level API.

\textbf{GITypeInfo — Struct representing a type.}

Procedures

\texttt{[g-type-tag-to-string]}, page 72
\texttt{[g-info-type-to-string]}, page 72
\texttt{[g-type-info-is-pointer]}, page 72
\texttt{[g-type-info-get-tag]}, page 72
\texttt{[g-type-info-get-param-type]}, page 72
\texttt{[g-type-info-get-interface]}, page 72
\texttt{[g-type-info-get-array-length]}, page 73
\texttt{[g-type-info-get-array-fixed-size]}, page 73
\texttt{[g-type-info-is-zero-terminated]}, page 73
\texttt{[g-type-info-get-array-type]}, page 73

Struct Hierarchy

\texttt{GIBaseInfoInfo}
\hfill ➝ \texttt{GITypeInfo}

Description

\texttt{GITypeInfo} represents a type. You can retrieve a type info from an argument (see Section 3.5.12 [Arg Info], page 65), a functions return value (see Section 3.5.6 [Function
INFO], page 54), a field (see GIFIELDINFO), a property (see SECTION 3.5.15 [PROPERTY INFO], page 70), a constant (see GIConstantInfo) or for a union discriminator (see GIUnionInfo). 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 GITypeInfo.

g-type-tag-to-string type-tag
    Returns a string or #f.
    Obtains the string representation for type-tag or #f if it does not exists.
    type-tag can either be a symbol or an id, a member of the enum-set of [%gi-type-tag], page 48, (otherwise, #f is returned).

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 52, (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 67) 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 48, 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 48) 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 51.
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 48), or -1 will be 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 48), or -1 will be 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 48), 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 49). If the type tag of this type is not array, #f will be returned.

---

3.5.17 Typelib

G-Golf Typelib low level API.

GITypelib — Layout and accessors for typelib.

Procedures

- [g-golf-typelib-new], page 73
- [call-with-input-typelib], page 73
- [g-typelib-new-from-memory], page 74
- [g-typelib-free], page 74
- [g-typelib-get-namespace], page 74

Description

TODO.

Procedures

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

- g-golf-typelib-new file
  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 74.

- call-with-input-typelib file proc
  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

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

Returns nothing.

Free a GITypelib.

g-typelib-get-namespace typelib

Returns the namespace of typelib.

3.5.18 Utilities

G-Golf GObject Introspection Utilities low level API.

Procedures and Syntax

[gi-pointer-new], page 75
[gi-pointer-inc], page 75
[gi-attribute-iter-new], page 75
[with-gerror], page 75
[gi->scm], page 75
[gi-boolean->scm], page 76
[gi-string->scm], page 76
[gi-strings->scm], page 76
[gi-csv-string->scm], page 76
[gi-pointer->scm], page 76
[gi-glist->scm], page 76
[gi-gslist->scm], page 76

Types and Values

[%gi-pointer-size], page 77

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 GIAttributeIter (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
Returns the scheme representation of value.
The accepted type symbols (also called tags in the GI terminology) are:

- **boolean**  
The GType of value is gboolean. The returned value is #t or #f.

- **string**  
The GType of value is *gchar. The returned value is a string or #f (when value is the %null-pointer).

- **strings**  
The GType of value is **gchar (an array of pointers to *gchar), returned as a list of strings, which can be empty (when value is the %null-pointer).

- **csv-string**  
The GType of value is *gchar. The returned value is a list of strings, obtained by splitting value using #\, as the char-pred, or #f (when value is the %null-pointer).

- **pointer**  
The GType of value is gpointer. The returned value is a pointer or #f (when value is the %null-pointer).

- **glist**  
The GType of value is glist. The returned value is a (possibly empty) list of items contained in the glist pointed by value.

Note: currently, the only supported item type is **object** (which implies that the type argument is a type description as documented below (see [gi-glist->scm], page 76, [gi-gslist->scm], page 76). G-Golf will of course provide translation for all other glist gslist item types, but in the mean time, if you encounter such a missing glist

---

16 All (G-Golf) procedures that accept a GI tag as an argument will accept the tag as an integer (the GI representation) or a symbol (the scheme representation). All (G-Golf) procedures that return a tag will always return the scheme representation.
gslist The GType of value is gslist. The returned value is a (possibly empty) list of items contained in the glist pointed by value.

Note: currently, the only supported item type is object (which implies that the type argument is a type description as documented below (see [gi-glist->scm], page 76, [gi-gslist->scm], page 76). G-Golf will of course provide translation for all other glist gslist item types, but in the mean time, if you encounter such a missing glist gslist item type, please Section 1.3 [Contact], page 4, us so we immediately fix that for you.

### gi-boolean->scm value

[Procedure]

Returns #t or #f.

The GType of value is gboolean.

### gi-string->scm value

[Procedure]

Returns a string or #f.

The GType of value is gchar*. When value is the %null-pointer, the procedure returns #f.

### gi-strings->scm value

[Procedure]

Returns a list of strings (which can be empty).

The GType of value is gchar** (an array of pointers to *gchar).

### gi-csv-string->scm value

[Procedure]

Returns a list of strings or #f.

The GType of value is gchar*. When the returned value is a list of strings, it is obtained by splitting value using #\, as the char-pred. When value is the %null-pointer, the procedure returns #f.

### gi-pointer->scm value

[Procedure]

Returns a pointer or #f.

The GType of value is gpointer. When value is the %null-pointer, the procedure returns #f.

### gi-glist->scm g-list type-desc

[Procedure]

### gi-gslist->scm g-slist type-desc

[Procedure]

Returns a list.

Obtains and returns a (possibly empty) list composed of the scheme representation of the data field of each element of g-list or g-slist respectively.

The type-desc is a list composed of the g-list or g-slist type tag, ‘interface if the list element param-tag is interface or #f otherwise, the interface description (if the previous is not #f), followed by a boolean which is #t if the g-list (g-slist) data field is a pointer, #f otherwise.
An interface description is a list composed of the interface GI type tag, the G-Golf type name, the G-Golf type, the g-type id and a boolean which is #t if the last three values have been verified\(^{17}\). As an example, here is the type description of the returned value for `clutter-actor-get-children`:

```
(glist
  interface
  (object
    <clutter-actor>
    #<<gobject-class> <clutter-actor> 561c2e9c2960>
    94679044057632
    #f)
  #f)
```

### Types and Values

#### %gi-pointer-size

[Variable]

The size (the number of bytes) that a (Glib) pointer occupies in memory (which is architecture dependent).

### 3.6 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)), ...

#### 3.6.1 Modules

[re-export-public-interface], page 77

**re-export-public-interface . args**

[Special Form]

Re-export the public interface of a module or modules. Invoked as (re-export-modules (mod1) (mod2)...).

#### 3.6.2 Goops

**Syntax, Procedures and Methods**

[class-direct-virtual-slots], page 77

[class-virtual-slots], page 78

[class-direct-g-property-slots], page 78

[class-g-property-slots], page 78

[mslot-set!], page 78

**class-direct-virtual-slots (self <class>)**

[Method]

Returns a list.

---

\(^{17}\) It happens, due to the (unspecified) order at which ‘things’ are imported from a GI typelib, that a G-Golf class or instance hasn’t been finalized when the procedure or method is being imported, hence these must be verified, which is done at the first (and only once) procedure or method call.
Obtains and returns the list of the class direct slots for self that satisfy the \((\text{eq}\?)\ (\text{slot-definition-allocation slot}) \#:.virtual)\) predicate.

```scheme
class-virtual-slots (self <class>)
```

[Method]

Returns a list.

Obtains and returns the list of the class slots for self that satisfy the \((\text{eq}\?)\ (\text{slot-definition-allocation slot}) \#:.virtual)\) predicate.

```scheme
class-direct-g-property-slots (self <class>)
```

[Method]

Returns a list.

Obtains and returns the list of the class direct slots for self that satisfy the \((\text{eq}\?)\ (\text{slot-definition-allocation slot}) \#:.g-property)\) predicate.

```scheme
class-g-property-slots (self <class>)
```

[Method]

Returns a list.

Obtains and returns the list of the class slots for self that satisfy the \((\text{eq}\?)\ (\text{slot-definition-allocation slot}) \#:.g-property)\) predicate.

```scheme
mslot-set! inst s1 v1 s2 v2 s3 v3 ...
```

[Procedure]

Returns nothing.

Performs a multiple `slot-set!` for `inst`, setting its slot named `s1` to the value `v1`, `s2` to `v2`, `s3` to `v3` . . .

### 3.6.3 G-export

### 3.6.4 Enum

#### Classes

- `<enum>`, page 78
- `<gi-enum>`, page 79

#### Procedures and Methods

- `![enum-set]`, page 79
- `enum->value`, page 79
- `enum->values`, page 79
- `enum->symbol`, page 79
- `enum->symbols`, page 79
- `enum->name`, page 79
- `enum->names`, page 79
- `!gtype-id_`, page 80
- `!gi-name`, page 80
- `!scm-name`, page 80

#### `<enum>`

The `<enum>` class is for enumerated values. Its (unique) slot is:

```scheme
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 Section 3.1.3 [GOOPS Notes and Conventions], page 10, ’Slots are not Immutable’).

!enum-set enum

Returns the content of the enum-set slot for enum.

enum->value enum symbol
enum->values enum

Returns the enum value for symbol (or #f if it does not exists), or the list of all values for enum, respectively.

enum->symbol enum value
enum->symbols enum

Returns the enum symbol for value (or #f if it does not exists), or the list of all symbols for enum, respectively.

enum->name enum value
enum->names enum

Returns the enum name (the string representation of the symbol) for value (or #f if it does not exists), or the list of all names for enum, respectively.

value can either be a symbol or an id.

<gi-enum>

The <gi-enum> class is a subclass of <enum>. Its class-direct-slots are:

gtype-id #:accessor !gtype-id
#:init-keyword #:gtype-id
#:init-value #f

gi-name #:accessor !gi-name
#:init-keyword #:gi-name

scm-name #:accessor !scm-name

The scm-name slot is automatically initialized.

Instances of the <gi-enum> class are immutable (to be precise, there are not meant to be mutated, see Section 3.1.3 [GOOPS Notes and Conventions], page 10, ’Slots are not Immutable’).
3.6.5 Flag

Classes

[<gi-flag>], page 80

Procedures

[gi-integer->gflags], page 80
[gi-gflags->integer], page 80

Description

<gi-flag>  
The <gi-flag> class is a subclass of <gi-enum>. Its has no direct slots.

gi-integer->gflags gflags n  
[Procedure]  
Returns a possibly empty) list of symbol(s).
Obtains and returns the list of (symbol) flags for the given <gi-flag> instance gflags and its integer representation n.

gi-gflags->integer gflags flags  
[Procedure]  
Returns an integer.
Compute and returns the integer representation for the list of (symbol(s)) given by flags and the given <gi-flag> instance gflags.

3.6.6 Struct

Classes

[<gi-struct>], page 80

Procedures and Accessors

[!gi-name_], page 81
[!scm-name_], page 81
[alignment], page 81
[size], page 81
[is-gtype-struct?], page 81
[is-foreign?], page 81
[field-types], page 81
[scm-types], page 81
[is-init-vals], page 81
[is-opaque?], page 81
[is-semi-opaque?], page 82
The `<gi-struct>` class is a subclass of `<struct>`. Its `class-direct-slots` are:

- `gi-name`: accessor `!gi-name`
- `#:accessor #:gi-name`
- `#:init-keyword #:gi-name`
- `scm-name`: accessor `!scm-name`
- `alignment`: accessor `!alignment`
- `#:accessor #:alignment`
- `#:init-keyword #:alignment`
- `size`: accessor `!size`
- `#:accessor #:size`
- `#:init-keyword #:size`
- `is-gtype-struct?`: accessor `!is-gtype-struct?`
- `#:accessor #:is-gtype-struct?`
- `field-types`: accessor `!field-types`
- `#:accessor #:field-types`
- `scm-types`: accessor `!scm-types`
- `#:accessor #:scm-types`
- `init-vals`: accessor `!init-vals`
- `#:accessor #:init-vals`
- `is-opaque?`: accessor `!is-opaque?`
- `is-semi-opaque`: accessor `!is-semi-opaque`

The `scm-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 Section 3.1.3 [GOOPS Notes and Conventions], page 10, 'Slots are not Immutable').

- `!gi-name`: accessor `gi-struct`
- `#:accessor #:gi-name`
- `#:init-keyword #:gi-name`
- `!scm-name`: accessor `gi-struct`
- `#:accessor #:scm-name`
- `alignment`: accessor `gi-struct`
- `#:accessor #:alignment`
- `#:init-keyword #:alignment`
- `size`: accessor `gi-struct`
- `#:accessor #:size`
- `#:init-keyword #:size`
- `is-gtype-struct?`: accessor `gi-struct`
- `#:accessor #:is-gtype-struct?`
- `field-types`: accessor `gi-struct`
- `#:accessor #:field-types`
- `scm-types`: accessor `gi-struct`
- `#:accessor #:scm-types`
- `init-vals`: accessor `gi-struct`
- `#:accessor #:init-vals`
- `is-opaque?`: accessor `gi-struct`
- `#:accessor #:is-opaque?`
- `is-semi-opaque`: accessor `gi-struct`
- `#:accessor #:is-semi-opaque`

Returns the content of their respective slot for `gi-struct`.

- `!is-opaque?`: accessor `gi-struct`
- `#:accessor #:is-opaque?`

Returns `#t` if `gi-struct` 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 GStructInfo 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 `blindingly` receive, pass and/or return their pointer(s).

`is-semi-opaque? gi-struct`          [Accessor]
Returns `#t` if `gi-struct` 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 `blindingly` receive, pass and/or return their pointer(s).

### 3.6.7 Union
G-Golf procedures to deal with C unions.

#### Procedures

- `[make-c-union]`, page 82
- `[c-union-ref]`, page 82
- `[c-union-set!]`, page 82

#### Description
G-Golf procedures to deal with C unions.

#### Procedures

`make-c-union types [type #f] [val #f]`             [Procedure]
Returns a pointer.

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](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 foreign size type`             [Procedure]
Returns the content of the C union pointed by `foreign`, for the given `size` and `type`.

`c-union-set! foreign size type val`             [Procedure]
Returns nothing.

Sets the content of the C union pointed by `foreign` to `val`, given its `size` and `type`.

### 3.6.8 Utilities
Procedures

[g-studly-caps-expand], page 83
[g-name->scm-name], page 83
[g-name->class-name], page 83
[gi-type-tag->ffi], page 84
[gi-type-tag->init-val], page 84

Types and Values

[%g-name-transform-exceptions], page 84
[%g-studly-caps-expand-token-exceptions], page 84

Description

G-Golf utilities low level API.

Procedures

g-studly-caps-expand str

Returns a string.

Given a ‘Camel Case (https://en.wikipedia.org/wiki/Camel_case)’ string, this procedure returns a new string, with all ‘_’ transformed into ‘-’, uppercase letters are transformed into their corresponding lowercase letter, and with a #\- is inserted in between occurrences of two consecutive uppercase letters, unless the sequence analysed is part of a prefix defined in the [%g-studly-caps-expand-token-exceptions], page 84, alist. Here are two examples:

(g-studly-caps-expand "GStudlyCapsExpand")

$2 = "g-studly-caps-expand"

(g-studly-caps-expand "WebKitWebContext")

$3 = "webkit-web-context"

[g-name->scm-name] name

Returns a string - a symbol - respectively.

The former obtains the scheme name for name, by looking for a possible entry in [%g-name-transform-exceptions], page 84, then, if it failed, by calling [g-studly-caps-expand], page 83.

The later uses the former, surrounds the result using #\< and #\> characters then calls string->symbol. For example:

(g-name->class-name "ClutterActor")

18 This procedure, as well as [g-name->scm-name], page 83, and [g-name->class-name], page 83, 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 the (Guile-Gnome) module (gnome gobject utils). In G-Golf, these will also be used to transform other (Gobject Inptrospection given) names, such as function names, hence their g-name-> prefix instead.
$2 = <\text{clutter-actor}>$

**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 48. 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.

### Types and Values

%g-name-transform-exceptions  
[Variable]  
Contains an alist where each key is a GType name exception for the [g-name->scm-name], page 83, procedure, and the corresponding value the string that [g-name->scm-name], page 83, will return instead. As an example, it could be defined as:

```
(define %g-name-transform-exceptions
  '((GEnum . genum)))
```

However currently it is an empty list\(^{19}\).

%g-studly-caps-expand-token-exceptions  
[Variable]  
Contains an alist where each key is a token exception for the [g-studly-caps-expand], page 83, procedure, and the corresponding value the string that [g-studly-caps-expand], page 83, will use for that token transformation instead. It is currently defined as:

```
(define %g-studly-caps-expand-token-exceptions
  '(("WebKit" . "webkit")))
```

Users may add or remove alist pairs to satisfy their needs.

### 3.7 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))
```

\(^{19}\)Guile-Gnome ([https://www.gnu.org/software/guile-gnome](https://www.gnu.org/software/guile-gnome)) defines a relatively long list of those GType name exceptions, including GEnum, but I'm still not sure about what G-Golf will do and currently decided not to apply any exception. Will this change in the future? Maybe, but it will become stable before G-Golf 1.0.
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).

### 3.7.1 GType

G-Golf GType high level API.
The base of the GObject type system.

#### Classes

[<gtype-class>], page 85
[<gtype-instance>], page 86

#### Accessors and Methods

[!info], page 85
[!namespace], page 85
[!gtype-id], page 85
[!gtype-name], page 85
[!scm-name], page 85
[!g-inst], page 86
[unref], page 86

**<gtype-class>**

The metaclass of all GType classes. Ensures that GType classes have an info slot, holding a pointer to a GIObjectInfo. Its slots are:

```lisp
(info #:accessor !info #:init-keyword #:info)
```

```lisp
(namespace #:accessor !namespace #:allocation #:virtual)
```

```lisp
(gtype-id #:accessor !gtype-id #:allocation #:virtual)
```

```lisp
(gtype-name #:accessor !gtype-name #:allocation #:virtual)
```

```lisp
(scm-name #:accessor !scm-name #:allocation #:virtual)
```

**!info class**  
**!namespace class**  
**!gtype-id class**  
**!gtype-name class**  
**!scm-name class**

Returns the content of their respective slot for class.
The root class of all instantiatable GType classes. Adds a slot, \texttt{g-inst}, to instances, which holds a pointer to the C value

\begin{verbatim}
g-inst #:accessor !g-inst
\end{verbatim}

The \texttt{g-inst} slot is initialized automatically and immutable (to be precise, it is not meant to be mutated, see Section 3.1.3 [GOOPS Notes and Conventions], page 10, 'Slots are not Immutable').

\begin{verbatim}
!g-inst instance
Returns the content of the \texttt{g-inst} slot for \texttt{instance}.
\end{verbatim}

\begin{verbatim}
unref instance
Returns nothing.
This method calls \texttt{[g-object-unref]}, page 24, on the \texttt{g-inst} of \texttt{instance}.
When the reference count for the \texttt{g-inst} reaches 0 (zero), it sets the \texttt{g-inst} slot value for \texttt{instance} to \texttt{#f} and removes \texttt{instance} from the \texttt{%g-inst-cache}.
This method must be called upon instances that are not referenced anywhere anymore, so that their memory can be freed by the next gc occurrence.
\end{verbatim}

### 3.7.2 GObject
G-Golf GObject high level API.
The G-Golf integration with the GObject object system.

**Classes**

- \texttt{[<gobject>], page 86}

**Procedures**

- \texttt{[gobject-class?], page 86}

**Description**

**Classes**

\texttt{<gobject>}

The base class for GLib’s default object system.

\texttt{gobject-class? val}

Returns \texttt{#t} if \texttt{val} is a member of the class-subclasses for \texttt{<gobject>}. Otherwise, it returns \texttt{#f}.

### 3.7.3 Closure
G-Golf closure high level API.
The G-Golf integration with GObject Closures.
Classes

[<closure>], page 87

Procedures and Methods

[!g-closure], page 88
[!function], page 88
[!return-type], page 88
[!param-types], page 88
[invoke], page 88

Description

The GLib/GObject type system supports the creation and invocation of ‘Closures’, which represents a callback supplied by the programmer (see Section 3.3.7 [Closures], page 31, 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 88, for example:

```scheme
(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

<closure>

Its slots are:

```scheme
<closure>

  g-closure #:accessor !g-closure

  function #:accessor !function
             #:init-keyword #:function

  return-type #:accessor !return-type
              #:init-keyword #:return-type
```
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 22.

Instances of the <closure> class are immutable (to be precise, there are not meant to be mutated, see Section 3.1.3 [GOOPS Notes and Conventions], page 10, 'Slots are not Immutable').

Accessors and Methods

Note: in this section, the closure argument is [must be] a <closure> instance.

### Accessors

- ![g-closure](closure) [Accessor]
- ![function](closure) [Accessor]
- ![return-type](closure) [Accessor]
- ![param-types](closure) [Accessor]

Returns the content of their respective slot for closure.

### Method

- ![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.\(^{20}\)

### 3.7.4 Function

G-Golf GI function and argument high level API.

The G-Golf GI function and argument high level API.

### Classes

- ![function>], page 89
- ![argument>], page 90

\(^{20}\) From scheme, you would ‘immediately’ call the procedure instead of course.
Accessors and Methods

[[info_], page 92
[!name], page 92
[!flags], page 92
[!n-arg], page 92
[!caller-owns], page 92
[!return-type_], page 92
[!type-desc], page 92
[!may-return-null], page 92
[!arguments], page 92
[!n-gi-arg-in], page 92
[!args-in], page 92
[!gi-args-in], page 92
[!n-gi-arg-out], page 92
[!args-out], page 92
[!gi-args-out], page 92
[!gi-arg-result], page 92
[!name_], page 92
[!closure], page 92
[!destroy], page 92
[!direction], page 92
[!transfert], page 92
[!scope], page 92
[!type-tag], page 92
[!type-desc_], page 92
[!forced-type], page 92
[!string-pointer], page 92
[!is-pointer?], page 92
[!may-be-null?], page 92
[!is-caller-allocate?], page 92
[!is-optional?], page 92
[!is-return-value?], page 92
[!is-skip?], page 92
[!gi-argument-in], page 92
[!gi-argument-out], page 92
[!gi-argument-field], page 92
[is-interface?], page 92

Variables

[gi-strip-boolean-result], page 93

Classes

Class

<function>

Its slots are:

info #:accessor !info
Instances of the `<function>` class are immutable (to be precise, there are not meant to be mutated, see Section 3.1.3 [GOOPS Notes and Conventions], page 10, 'Slots are not Immutable').

<argument>  
Its slots are:

- name #:accessor !name  
  #:init-keyword #:name
- closure #:accessor !closure
- destroy #:accessor !destroy
- direction #:accessor !direction  
  #:init-keyword #:direction
Instances of the `<argument>` class are immutable (to be precise, there are not meant to be mutated, see Section 3.1.3 [GOOPS Notes and Conventions], page 10, '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
!name function
!flags function
!n-arg function
!caller-owns function
!return-type function
!type-desc function
!may-return-null function
!arguments function
!n-gi-arg-in function
!args-in function
!gi-args-in function
!n-gi-arg-out function
!args-out function
!gi-args-out function
!gi-arg-result function

Returns the content of their respective slot for function.

!name argument
!closure argument
!destroy argument
!direction argument
!transfert argument
!scope argument
!type-tag argument
!type-desc argument
!forced-type argument
!string-pointer argument
!is-pointer? argument
!may-be-null? argument
!is-caller-allocate? argument
!is-optional? argument
!is-return-value? argument
!is-skip? argument
!gi-argument-in argument
!gi-argument-out argument
!gi-argument-field argument

Returns the content of their respective slot for argument.

is-interface? argument

Returns #t if the type-tag slot value for argument is 'interface. Otherwise, it returns #f.
Variables

%gi-strip-boolean-result

A list of procedure and method names that have at least one `inout` or `out` argument(s) and return either `#t` or `#f`, solely to indicate that the procedure or method call was successful or not.

These procedures and methods, if (and only if) their name is a member of `%gi-strip-boolean-result`, will see their returned value eluded if it is `#t`, otherwise, an exception will be raised\(^{21}\).

Initially, `%gi-strip-boolean-result` is empty, and it is a user responsibility to fill it appropriately, for each namespace they are importing.

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

```
(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 this procedure name to `%gi-strip-boolean-result`:

```
(push! 'clutter-color-from-string %gi-function-call-strip-boolean-result)
 $6 = (clutter-color-from-string)
```

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

And call it with an undefined color name:

```
(clutter-color-from-string "Bluee")
 |- scm-error" "clutter-color-from-string" failed."
```

Entering a new prompt. Type `\',bt` for a backtrace or `\',q` to continue.

---

\(^{21}\) In any other situation, but void, the returned value comes first, then in order, if any, the `inout` and/or `out` argument(s).
3.7.5 Import

G-Golf Import high level API.
The G-Golf GIR namespace (Typelib) import high level API.

Procedures

- [gi-import], page 94
- [gi-import-by-name], page 94
- [gi-import-info], page 95
- [gi-import-enum], page 95
- [gi-import-flag], page 95
- [gi-import-struct], page 95
- [gi-import-function], page 96
- [gi-import-constant], page 97

Variables

- [%gi-base-info-types], page 97
- [%gi-imported-base-info-types], page 97

Procedures

**gi-import namespace**

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*.

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 a 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 94, here below.

**gi-import-by-name namespace name [#:recur #t]**

Returns the object or constant returned by [gi-import-info], page 95, 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 keyword #:recur argument - which is #t by default - is passed to the gi-import-enum, gi-import-flag and gi-import-struct. When #:recur is #f, then the enum, flag 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,
gflag 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" #:recur #f)
```

As mentioned above, 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 here below.

**gi-import-info info**[Procedure]

Returns the object or constant returned by the one of the gi-import-enum, gi-import-flag, ..., 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 94, 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-flag> 5642cb13c5d0>

(describe $4)
#<<gi-flag> 5642cb13c5d0> is an instance of class <gi-flag>
Slots are:
enum-set = ((mapped 2) (realized 4) (reactive 8) (visible 16) (no-layout 32))
gtype-id = 94844874149456
gi-name = "ClutterActorFlags"
scm-name = "clutter-actor-flags"
```

**gi-import-enum info [#:recur #t]**[Procedure]

**gi-import-flag info [#:recur #t]**[Procedure]

**gi-import-struct info [#:recur #t]**[Procedure]

Returns a [<gi-enum>], page 79, a [<gi-flag>], page 80, or a [<gi-struct>], page 80, instance, respectively.

The info argument is (must be) a pointer to GIEnumInfo, a GIEnumInfo for which ([g-base-info-get-type], page 51, info) returned 'flags and a GIStructInfo respectively. It is an error to call any of these procedures upon an invalid info argument.

The optional keyword #:recur argument - which is #t by default - is passed using #f, then info will be imported without its respective methods. A description and
chapter 3: api reference

an example ware also given here above, as part of the [gi-import-by-name], page 94, documentation entry.

Every imported [gi-enum], page 79, [gi-flag], page 80, and [gi-struct], page 80, instance is cached under the 'enum, 'flag and 'boxed main key (respectively), and using the symbol representation of the value of their scm-name slot as the secondary key. For example, reusing the "Clutter" "ActorFlags" namespace/name introduced above, you would retrieve its [gi-flag], page 80, instance as is:

\[
\text{(gi-cache-ref 'flag 'clutter-actor-flags)}
\]

\[
\text{\$6 = #<<gi-flag> 5642cb13c5d0>}
\]

\text{gi-import-function info } \text{[Procedure]}

Returns a [function], page 89, instance.

Imports info - a pointer to a GIFunctionInfo (see Section 3.5.6 [Function Info], page 54), 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 [function], page 89, instance name slot as the secondary key. Here is an example:

\[
,use (g-golf)
\text{(g-irepository-require "Clutter")}
\text{\$2 = #<pointer 0x55c191f3fe30>}
\]

\[
\text{(g-irepository-find-by-name "Clutter" "init")}
\text{\$3 = #<pointer 0x55c191f41de0>}
\]

\[
\text{(gi-import-function \$3)}
\text{\$4 = #<<function> 55c191e81510>}
\]

\[
\text{(describe \$4)}
\text{#<<function> 55c191e81510> is an instance of class <function>}
\text{Slots are:}
\text{info = #<pointer 0x55c191f41de0>}
\text{name = clutter-init}
\text{flags = ()}
\text{n-arg = 2}
\text{caller-owns = nothing}
\text{return-type = interface}
\text{...}
\]

\[
\text{(gi-cache-ref 'function 'clutter-init)}
\text{\$5 = #<<function> 55c191e81510>}
\]

\text{Returned value(s):}
In most situations, but when 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 `inout` and/or `out` argument(s).

However, some function and method, that have at least one `inout` or `out` argument(s), do return a `boolean`, but 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/or `out` argument(s) have been ’correctly’ set and may be safely 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 defines a [%gi-strip-boolean-result], page 93, variable, initially empty, that users may fill appropriately, using the function or method (symbol) name, as described in its documentation: make sure to carefully read and understand it.

**gi-import-constant info**  [Procedure]

Returns two values, the constant value and its name.

Obtains and returns the `info` constant value and its name. For example:

```lisp
,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"
```

Constants are curently not being automatically imported, though this will probably change in the near future, stay tuned.

**Variables**

%gi-base-info-types  [Variable]
%gi-imported-base-info-types  [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 94; (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 94, [gi-import-info], page 95, and/or [gi-import-by-name], page 94, 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 94, calls and how complete is \texttt{G-Golf}.

\section*{Appendix A GNU Free Documentation License}

\begin{flushright}
Version 1.3, 3 November 2008
\end{flushright}

\begin{flushleft}
\url{http://fsf.org/}
\end{flushleft}

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

\section*{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.

\section*{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
Appendix A: GNU Free Documentation License

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 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 60 days after your receipt of the notice.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

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 Multi Author 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 Multi Author 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.

C

G
GPL................................. 1

L

copying.............................. 1 license .................................... 1
Procedure Index

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>!alignment</td>
<td>81</td>
</tr>
<tr>
<td>!args-in</td>
<td>92</td>
</tr>
<tr>
<td>!args-out</td>
<td>92</td>
</tr>
<tr>
<td>!arguments</td>
<td>92</td>
</tr>
<tr>
<td>!caller-owns</td>
<td>92</td>
</tr>
<tr>
<td>!close</td>
<td>92</td>
</tr>
<tr>
<td>!destroy</td>
<td>92</td>
</tr>
<tr>
<td>!direction</td>
<td>92</td>
</tr>
<tr>
<td>!enum-set</td>
<td>79</td>
</tr>
<tr>
<td>!event</td>
<td>36</td>
</tr>
<tr>
<td>!field-types</td>
<td>81</td>
</tr>
<tr>
<td>!flags</td>
<td>92</td>
</tr>
<tr>
<td>!forced-type</td>
<td>92</td>
</tr>
<tr>
<td>!function</td>
<td>88</td>
</tr>
<tr>
<td>!g-closure</td>
<td>88</td>
</tr>
<tr>
<td>!g-inst</td>
<td>86</td>
</tr>
<tr>
<td>!gi-arg-result</td>
<td>92</td>
</tr>
<tr>
<td>!gi-args-in</td>
<td>92</td>
</tr>
<tr>
<td>!gi-args-out</td>
<td>92</td>
</tr>
<tr>
<td>!gi-argument-field</td>
<td>92</td>
</tr>
<tr>
<td>!gi-argument-in</td>
<td>92</td>
</tr>
<tr>
<td>!gi-argument-out</td>
<td>92</td>
</tr>
<tr>
<td>!gi-name</td>
<td>80, 81</td>
</tr>
<tr>
<td>!g-name</td>
<td>80, 85</td>
</tr>
<tr>
<td>!gtypedef-id</td>
<td>80, 85</td>
</tr>
<tr>
<td>!gtypedef-name</td>
<td>85</td>
</tr>
<tr>
<td>!info</td>
<td>85, 92</td>
</tr>
<tr>
<td>!init-vals</td>
<td>81</td>
</tr>
<tr>
<td>!is-caller-allocate</td>
<td>92</td>
</tr>
<tr>
<td>!is-closure-struct</td>
<td>92</td>
</tr>
<tr>
<td>!is-opaque</td>
<td>81</td>
</tr>
<tr>
<td>!is-optional</td>
<td>81</td>
</tr>
<tr>
<td>!is-pointer</td>
<td>92</td>
</tr>
<tr>
<td>!is-return-value</td>
<td>92</td>
</tr>
<tr>
<td>!is-semi-opaque</td>
<td>82</td>
</tr>
<tr>
<td>!is-skip</td>
<td>92</td>
</tr>
<tr>
<td>!may-be-null</td>
<td>92</td>
</tr>
<tr>
<td>!may-return-null</td>
<td>92</td>
</tr>
<tr>
<td>!n-arg</td>
<td>92</td>
</tr>
<tr>
<td>!n-gi-arg-in</td>
<td>92</td>
</tr>
<tr>
<td>!n-gi-arg-out</td>
<td>92</td>
</tr>
<tr>
<td>!name</td>
<td>92</td>
</tr>
<tr>
<td>!namespace</td>
<td>85</td>
</tr>
<tr>
<td>!param-types</td>
<td>88</td>
</tr>
<tr>
<td>!return-type</td>
<td>88, 92</td>
</tr>
<tr>
<td>!scm-name</td>
<td>80, 81, 85</td>
</tr>
<tr>
<td>!scm-types</td>
<td>81</td>
</tr>
<tr>
<td>!scope</td>
<td>92</td>
</tr>
<tr>
<td>!size</td>
<td>81</td>
</tr>
<tr>
<td>!string-pointer</td>
<td>92</td>
</tr>
<tr>
<td>!transferred</td>
<td>92</td>
</tr>
<tr>
<td>!type-desc</td>
<td>92</td>
</tr>
<tr>
<td>!type-tag</td>
<td>92</td>
</tr>
</tbody>
</table>

C

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>c-union-ref</td>
<td>82</td>
</tr>
<tr>
<td>c-union-set</td>
<td>82</td>
</tr>
<tr>
<td>call-with-input-typelib</td>
<td>73</td>
</tr>
<tr>
<td>class-direct-g-property-slots</td>
<td>78</td>
</tr>
<tr>
<td>class-direct-virtual-slots</td>
<td>77</td>
</tr>
<tr>
<td>class-g-property-slots</td>
<td>78</td>
</tr>
<tr>
<td>class-virtual-slots</td>
<td>78</td>
</tr>
</tbody>
</table>

E

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>enum-&gt;name</td>
<td>79</td>
</tr>
<tr>
<td>enum-&gt;names</td>
<td>79</td>
</tr>
<tr>
<td>enum-&gt;symbol</td>
<td>79</td>
</tr>
<tr>
<td>enum-&gt;symbols</td>
<td>79</td>
</tr>
<tr>
<td>enum-&gt;value</td>
<td>79</td>
</tr>
<tr>
<td>enum-&gt;values</td>
<td>79</td>
</tr>
</tbody>
</table>

G

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>g-arg-info-get-closure</td>
<td>66</td>
</tr>
<tr>
<td>g-arg-info-get-destroy</td>
<td>66</td>
</tr>
<tr>
<td>g-arg-info-get-direction</td>
<td>66</td>
</tr>
<tr>
<td>g-arg-info-get-ownership-transfer</td>
<td>66</td>
</tr>
<tr>
<td>g-arg-info-get-scope</td>
<td>67</td>
</tr>
<tr>
<td>g-arg-info-get-type</td>
<td>67</td>
</tr>
<tr>
<td>g-arg-info-is-caller-allocates</td>
<td>67</td>
</tr>
<tr>
<td>g-arg-info-is-optional</td>
<td>67</td>
</tr>
<tr>
<td>g-arg-info-is-return-value</td>
<td>67</td>
</tr>
<tr>
<td>g-arg-info-is-skipped</td>
<td>67</td>
</tr>
<tr>
<td>g-arg-info-may-be-null</td>
<td>67</td>
</tr>
<tr>
<td>g-base-info-equal</td>
<td>51</td>
</tr>
<tr>
<td>g-base-info-get-attribute</td>
<td>51</td>
</tr>
<tr>
<td>g-base-info-get-container</td>
<td>51</td>
</tr>
<tr>
<td>g-base-info-get-name</td>
<td>51</td>
</tr>
<tr>
<td>g-base-info-get-instance-</td>
<td>51</td>
</tr>
<tr>
<td>g-base-info-get-type</td>
<td>51</td>
</tr>
<tr>
<td>g-base-info-get-typelib</td>
<td>51</td>
</tr>
<tr>
<td>g-base-info-is-deprecated</td>
<td>51</td>
</tr>
<tr>
<td>g-base-info-iterate-attributes</td>
<td>51</td>
</tr>
<tr>
<td>g-base-info-ref</td>
<td>51</td>
</tr>
<tr>
<td>g-base-info-unsafe</td>
<td>51</td>
</tr>
<tr>
<td>g-callable-info-get-arg</td>
<td>53</td>
</tr>
<tr>
<td>g-callable-info-get-callers</td>
<td>53</td>
</tr>
<tr>
<td>g-callable-info-get-instance-</td>
<td>53</td>
</tr>
<tr>
<td>g-callable-info-get-instance-</td>
<td>53</td>
</tr>
<tr>
<td>g-callable-info-get-name</td>
<td>53</td>
</tr>
<tr>
<td>g-callable-info-get-n-args</td>
<td>53</td>
</tr>
<tr>
<td>g-callable-info-get-return-type</td>
<td>53</td>
</tr>
<tr>
<td>g-callable-info-get-return-type</td>
<td>53</td>
</tr>
<tr>
<td>g-callable-info-may-return</td>
<td>53</td>
</tr>
<tr>
<td>g-closure-add-invalidate-notifier</td>
<td>32</td>
</tr>
</tbody>
</table>
Procedure Index 108

g-closure-free .......................... 32
g-closure-involve ........................ 32
g-closure-new-simple .................... 33
g-closure-ref ............................ 32
g-closure-ref-count ...................... 32
g-closure-set-marshall .................. 33
g-closure-sink ........................... 32
g-closure-size ........................... 31
g-closure-unref .......................... 32
g-constant-info-free-value ............. 69
g-constant-info-get-type ............... 69
g-constant-info-get-value ............. 69
g-enum-info-get-method ................ 58
g-enum-info-get-n-methods ............. 58
g-enum-info-get-n-values .............. 58
g-enum-info-get-value .................. 58
g-field-info-get-type ................... 70
g-free .................................... 12
g-function-info-get-flags .............. 55
g-function-info-get-property .......... 55
g-function-info-get-symbol ............ 55
g-function-info-get-vfunc .............. 55
g-function-info-involve ............... 55
g-golf-typelib-new ....................... 73
g-idle-source-new ....................... 14
g-info-type-to-string ................... 72
g-io-channel-ref ........................ 16
g-io-channel-unix-new .................. 16
g-io-channel-unref ...................... 16
g-io-create-watch ....................... 16
g-irepository-find-by-gtype .......... 48
g-irepository-find-by-name .......... 48
g-irepository-get-c-prefix .......... 48
g-irepository-get-default .......... 47
g-irepository-get-dependencies ........ 47
g-irepository-get-info ............... 47
g-irepository-get-loaded-namespaces .. 47
g-irepository-get-n-consts .......... 47
g-irepository-get-shared-library ..... 48
g-irepository-get-typelib-path ....... 47
g-irepository-get-version ............. 48
g-irepository-require .................. 47
g-list-data ............................. 18
g-list-free .............................. 18
g-list-length ............................ 18
g-list-next ............................. 18
g-list-nth-data .......................... 18
g-list-prev .............................. 18
g-main-context-default ................. 13
g-main-context-new ..................... 13
g-main-loop-new ......................... 13
g-main-loop-quit ........................ 13
g-main-loop-ref ......................... 13
g-main-loop-run ......................... 13
g-main-loop-unref ...................... 13
g-malloc ................................ 11
g-malloc0 ................................ 11
g-memdup ................................ 12
g-name->class-name ..................... 83
g-name->scm-name ......................... 83
g-object-get-property .................. 24
g-object-info-find-method .............. 64
g-object-info-find-sign ................. 65
g-object-info-get-abstract .......... 63
g-object-info-get-class-struct ....... 65
g-object-info-get-constant .......... 64
g-object-info-get-field ............... 64
g-object-info-get-interface .......... 64
g-object-info-get-method .............. 64
g-object-info-get-n-constants ......... 64
g-object-info-get-n-fields .......... 64
g-object-info-get-n-interfaces ...... 64
g-object-info-get-n-methods .......... 64
g-object-info-get-n-properties ...... 65
g-object-info-get-n-signals .......... 65
g-object-info-get-n-vfuncs ........... 65
g-object-new ........................... 23
g-object-new-with-properties .......... 24
g-object-ref ............................ 24
g-object-ref-count ..................... 24
g-object-ref-sink ....................... 24
g-object-set-property .................. 25
g-object-type ......................... 24
g-object-type-name ..................... 24
g-object-unref .......................... 24
g-property-info-get-flags ............ 71
g-property-info-get-ownership-transfer 71
g-property-infos-get-type ............ 71
g-quark-from-string ................... 20
g-quark-to-string ...................... 20
g-registered-type-info-get-g-type .... 57
g-registered-type-info-get-g-type-init 57
g-registered-type-info-get-type-name 57
g-signal-info-get-flags .............. 54
g-signal-list-ids ..................... 34
g-signal-lookup ....................... 34
g-signal-query .......................... 33
g-slist-data ............................ 19
g-slist-free ............................ 19
g-slist-length ........................... 19
g-slist-next ............................ 19
g-slist-nth-data ....................... 19
g-source-attach ......................... 14
g-source-destroy ....................... 14
g-source-free ........................... 14
g-source-get-priority ................. 15
g-source-is-destroyed? ................. 15
g-source-ref ............................ 14
g-source-ref-count ..................... 14
Procedure Index

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>g-source-remove</td>
<td>15</td>
</tr>
<tr>
<td>g-source-set-closure</td>
<td>33</td>
</tr>
<tr>
<td>g-source-set-priority</td>
<td>15</td>
</tr>
<tr>
<td>g-source-unref</td>
<td>14</td>
</tr>
<tr>
<td>g-struct-info-get-alignment</td>
<td>59</td>
</tr>
<tr>
<td>g-struct-info-get-field</td>
<td>60</td>
</tr>
<tr>
<td>g-struct-info-get-method</td>
<td>60</td>
</tr>
<tr>
<td>g-struct-info-get-n-fields</td>
<td>60</td>
</tr>
<tr>
<td>g-struct-info-get-n-methods</td>
<td>60</td>
</tr>
<tr>
<td>g-struct-info-get-size</td>
<td>60</td>
</tr>
<tr>
<td>g-struct-info-is-foreign</td>
<td>60</td>
</tr>
<tr>
<td>g-struct-info-is-gtype-struct</td>
<td>60</td>
</tr>
<tr>
<td>g-studly-caps-expand</td>
<td>83</td>
</tr>
<tr>
<td>g-timeout-source-clean</td>
<td>13</td>
</tr>
<tr>
<td>g-timeout-source-new</td>
<td>14</td>
</tr>
<tr>
<td>g-&gt;symbol</td>
<td>21</td>
</tr>
<tr>
<td>g-&gt;fundamental</td>
<td>21</td>
</tr>
<tr>
<td>g-&gt;info-get-array-fixed-size</td>
<td>73</td>
</tr>
<tr>
<td>g-&gt;info-get-array-length</td>
<td>73</td>
</tr>
<tr>
<td>g-&gt;info-get-array-type</td>
<td>73</td>
</tr>
<tr>
<td>g-&gt;info-get-interface</td>
<td>72</td>
</tr>
<tr>
<td>g-&gt;info-get-param-type</td>
<td>72</td>
</tr>
<tr>
<td>g-&gt;info-get-tag</td>
<td>72</td>
</tr>
<tr>
<td>g-&gt;info-is-pointer</td>
<td>72</td>
</tr>
<tr>
<td>g-&gt;info-is-zero-terminated</td>
<td>73</td>
</tr>
<tr>
<td>g-&gt;name</td>
<td>21</td>
</tr>
<tr>
<td>g-&gt;tag-to-string</td>
<td>72</td>
</tr>
<tr>
<td>g-typelib-free</td>
<td>74</td>
</tr>
<tr>
<td>g-typelib-get-namespace</td>
<td>74</td>
</tr>
<tr>
<td>g-typelib-new-from-memory</td>
<td>74</td>
</tr>
<tr>
<td>g-union-info-get-alignment</td>
<td>62</td>
</tr>
<tr>
<td>g-union-info-get-discriminator</td>
<td>62</td>
</tr>
<tr>
<td>g-union-info-get-discriminator-offset</td>
<td>62</td>
</tr>
<tr>
<td>g-union-info-get-discriminator-type</td>
<td>62</td>
</tr>
<tr>
<td>g-union-info-get-field</td>
<td>61</td>
</tr>
<tr>
<td>g-union-info-get-method</td>
<td>61</td>
</tr>
<tr>
<td>g-union-info-get-n-fields</td>
<td>61</td>
</tr>
<tr>
<td>g-union-info-get-n-methods</td>
<td>61</td>
</tr>
<tr>
<td>g-union-info-get-size</td>
<td>62</td>
</tr>
<tr>
<td>g-union-info-is-discrimated?</td>
<td>61</td>
</tr>
<tr>
<td>g-union-info-is-discrimated?</td>
<td>61</td>
</tr>
<tr>
<td>g-union-info-new-create</td>
<td>17</td>
</tr>
<tr>
<td>g-value-&gt;g-type</td>
<td>27</td>
</tr>
<tr>
<td>g-value-&gt;g-type-id</td>
<td>27</td>
</tr>
<tr>
<td>g-value-get-boolean</td>
<td>27</td>
</tr>
<tr>
<td>g-value-get-boxed</td>
<td>29</td>
</tr>
<tr>
<td>g-value-get-double</td>
<td>28</td>
</tr>
<tr>
<td>g-value-get-enum</td>
<td>28</td>
</tr>
<tr>
<td>g-value-get-float</td>
<td>28</td>
</tr>
<tr>
<td>g-value-get-int</td>
<td>27</td>
</tr>
<tr>
<td>g-value-get-object</td>
<td>29</td>
</tr>
<tr>
<td>g-value-get-pointer</td>
<td>29</td>
</tr>
<tr>
<td>g-value-get-string</td>
<td>29</td>
</tr>
<tr>
<td>g-value-get-uint</td>
<td>28</td>
</tr>
<tr>
<td>g-value-info-get-value</td>
<td>59</td>
</tr>
<tr>
<td>g-value-init</td>
<td>26</td>
</tr>
<tr>
<td>g-value-ref</td>
<td>27</td>
</tr>
<tr>
<td>g-value-set!</td>
<td>27</td>
</tr>
<tr>
<td>Procedure Index</td>
<td>110</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----</td>
</tr>
<tr>
<td>gi-import</td>
<td>94</td>
</tr>
<tr>
<td>gi-import-by-name</td>
<td>94</td>
</tr>
<tr>
<td>gi-import-constant</td>
<td>97</td>
</tr>
<tr>
<td>gi-import-enum</td>
<td>95</td>
</tr>
<tr>
<td>gi-import-flag</td>
<td>95</td>
</tr>
<tr>
<td>gi-import-function</td>
<td>96</td>
</tr>
<tr>
<td>gi-import-info</td>
<td>95</td>
</tr>
<tr>
<td>gi-import-struct</td>
<td>95</td>
</tr>
<tr>
<td>gi-integer-&gt;gflags</td>
<td>80</td>
</tr>
<tr>
<td>gi-pointer-&gt;scm</td>
<td>76</td>
</tr>
<tr>
<td>gi-pointer-inc</td>
<td>75</td>
</tr>
<tr>
<td>gi-pointer-new</td>
<td>75</td>
</tr>
<tr>
<td>gi-property-g-type</td>
<td>71</td>
</tr>
<tr>
<td>gi-string-&gt;scm</td>
<td>76</td>
</tr>
<tr>
<td>gi-strings-&gt;scm</td>
<td>76</td>
</tr>
<tr>
<td>gi-struct-field-types</td>
<td>59</td>
</tr>
<tr>
<td>gi-struct-import</td>
<td>59</td>
</tr>
<tr>
<td>gi-type-tag-&gt;ffi</td>
<td>84</td>
</tr>
<tr>
<td>gi-type-tag-&gt;init-val</td>
<td>84</td>
</tr>
<tr>
<td>gobject-class?</td>
<td>86</td>
</tr>
<tr>
<td>invoke</td>
<td>88</td>
</tr>
<tr>
<td>is-interface?</td>
<td>92</td>
</tr>
<tr>
<td>is-method?</td>
<td>55</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>M</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>make-c-union</td>
<td>82</td>
</tr>
<tr>
<td>make-gdk-event</td>
<td>42</td>
</tr>
<tr>
<td>mslot-set!</td>
<td>78</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>R</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>re-export-public-interface</td>
<td>77</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>symbol-&gt;g-type</td>
<td>21</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>U</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>unref</td>
<td>86</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>W</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>with-gerror</td>
<td>75</td>
</tr>
</tbody>
</table>
Variable Index

This is an alphabetical list of all the important variables and constants in G-Golf.

%g-function-info-flags of <gi-flag> ........ 56
%g-io-condition of <gi-flag> ................ 16
%g-name-transform-exceptions ............... 84
%g-param-flags of <gi-enum> ............... 30
%g-signal-flags of <gi-enum> .............. 34
%g-studly-caps-expand-token-exceptions .... 84
%g-type-fundamental-flags of <gi-enum> ... 21
%g-type-fundamental-types of <gi-enum> ... 22
%gdk-event-type of <gi-enum> .............. 38
%gi-array-type of <gi-enum> ............... 49
%gi-base-info-types ...................... 97
%gi-direction of <gi-enum> ................ 67
%gi-imported-base-info-types ............. 97
%gi-info-type of <gi-enum> ................ 52
%gi-pointer-size .............. 77
%gi-scope-type of <gi-enum> .............. 68
%gi-type-tag of <gi-enum> ............... 48
Type Index

This is an alphabetical list of all the important data types defined in the G-Golf Programmers Manual.

<argument> ........................................ 90  <gdk-event> ...................................... 36
<closure> ......................................... 87  <gi-enum> ......................................... 79
<enum> ........................................... 78  <gi-flag> ........................................... 80
<function> ....................................... 89  <gi-struct> ....................................... 81
<gdk-event-button> .............................. 42  <gobject> .......................................... 86
<gdk-event-key> ................................. 42  <gtype-class> ...................................... 85
<gdk-event-motion> .............................. 42  <gtype-instance> ................................... 86