Theme-D-Intr User Manual

Tommi Höynälänmaa

Table of Contents

Introduction

Theme-D-Intr is a software library allowing the use of the introspected GObject modules in Theme-D.

Installation

Debian forky (testing) and sid (unstable) and Ubuntu Stonking

If you use the Synaptic Package Manager install the packages `theme-d-intr` and `theme-d-intr-dev` . If you want to install the example programs install one or both of the packages `theme-d-intr-gtk3-examples` and `theme-d-intr-gtk4-examples`. In order to install the packages from the command line give command

```
sudo apt install theme-d-intr theme-d-intr-dev
```

and optionally one or both of the commands

```
sudo apt install theme-d-intr-gtk3-examples
sudo apt install theme-d-intr-gtk4-examples
```

Older Versions of Debian and Ubuntu

If you are satisfied with an older version of the software follow the instructions in the previous paragraph. Otherwise, see Theme-D User Guide [https://www.iki.fi/tohoyn/theme-d/theme-d-user-guide-20260527-1.pdf] for instructions to build and install the software. The latter option is recommended.

Other Debian-based Systems

- Install Guile 3.0 is you don't have it already.
- Install Theme-D version 7.3.3 is you don't have it already. See Theme-D User Guide [https://www.iki.fi/tohoyn/theme-d/theme-d-user-guide-20260527-1.pdf] for instructions.
- Install G-Golf. See http://www.iki.fi/tohoyn/g-golf-debian/ .
- Create a directory for the build and download packages `theme-d-intr-1.0.2.tar.xz` and `theme-d-intr_1.0.2-1.debian.tar.xz` there.
- Execute the following commands

  ```
  tar xvf theme-d-intr-1.0.2.tar.xz
  ln -s theme-d-intr-1.0.2.tar.xz theme-d-intr_1.0.2.orig.tar.xz
  cd theme-d-intr-1.0.2
  tar xvf ../theme-d-intr_1.0.2-1.debian.tar.xz
  dpkg-buildpackage --no-sign -b
  ```

  in the directory.

- Give command `cd ..` and install the packages with commands

  ```
  sudo dpkg -i theme-d-intr_1.0.2-1_ARCH.deb
  sudo dpkg -i theme-d-intr-dev_1.0.2-1_all.deb
  ```

  The architecture ARCH can be computed by

  ```
  dpkg-architecture -q DEB_HOST_ARCH
  ```

  You can install the example programs by installing one or both of the packages `theme-d-intr-gtk4-examples_1.0.2-1_all.deb` and `theme-d-intr-gtk3-examples_1.0.2-1_all.deb`.

- Ensure that your Guile installation file `~/.guile` contains the following lines:

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

Other UNIX Systems

- Install the G-Golf library. See https://www.gnu.org/software/g-golf/install.html .
- Unpack package `theme-d-intr-1.0.2.tar.xz` into some directory.
- Change to the subdirectory `theme-d-intr-1.0.2` and give commands

```
./configure
make
sudo make install-complete
```

- Ensure that your Guile installation file `~/.guile` contains the following lines:

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

Using Theme-D-Intr without Installing

It is possible to use the software without installing it. First give the following commands in the subdirectory `theme-d-intr-1.0.2`:

```
./configure
make
```

Modify the file `~/.guile` as in a normal installation (see the previous section). Now you can launch the uninstalled environment by giving command

```
./uninstalled-env bash
```

in subdirectory `meta`. You also have to take care that the Theme-D module search path (option `-m` or `-M` for the Theme-D compiler and linker) contains the directory `xxx/theme-d-intr-1.0.2/theme-d-code`. See Theme-D User Guide for a description of the module search path.

Example Programs

It is recommended that you copy the example programs into your home directory for building them. If you have installed Theme-D-Intr normally go to directory `/usr/share/doc/theme-d-intr-gtk4-examples`. If you use Theme-D-Intr without installed it go to the subdirectory `examples` in the source code. Give command

```
cp -avi intr-gtk4-examples MYDIR
```

where `MYDIR` is the directory where you want to install the programs. Then, go to the directory `/usr/share/doc/theme-d-intr-gtk3-examples` or `examples` and give command

```
cp -avi intr-gtk3-examples MYDIR
```

Finally, go to the directory `/usr/share/doc/theme-d-intr-dev/examples` or `examples` and give command

```
cp -avi intr-dbus-examples MYDIR
```

The example programs `appwindow`, `hello`, `hello2`, `hello3`, `calc`, `calc2`, `custom-layout-example`, `flag-demo`, and `theme-d-intr-demo` for GTK 4 are located in directory `/usr/share/doc/theme-d-intr-gtk4-examples/intr-gtk4-examples` and in directory `examples/intr-gtk4-examples` in the source package. You can build the example programs with command `make -f user.mk` and run them with command

```
GUILE_LOAD_PATH=../..:$GUILE_LOAD_PATH run-theme-d-program \
       <program-name>
```

where `<program-name>` is the name of the target `.go` file of the program (`main.go` or `run-demo.go`). For program `hello` the command is

```
GUILE_LOAD_PATH=.:$GUILE_LOAD_PATH run-theme-d-program hello.go
```

Example program `intr-gtk4-examples/calc2` is linked modularly and it shall be run with command

```
run-theme-d-program-m -g ../..: main.go
```

or in uninstalled mode

```
run-theme-d-program-m -g ../..:../../../theme-d-code: main.go
```

Before you run it you have to install its configuration files with command

```
./install-calc2.sh
```

If you have a Debian-based Linux operating system you have to ensure that packages `gir1.2-gtk-4.0` and `gir1.2-adw-1` are installed in your system.

The example programs `hello`, `hello2`, `hello3`, `calc`, and `theme-d-intr-demo` for GTK 3 are located in directory `/usr/share/doc/theme-d-intr-gtk3-examples/intr-gtk3-examples` and in directory `examples/intr-gtk3-examples` in the source package. You can build the example programs with command `make -f user.mk` and run them with command

```
GUILE_LOAD_PATH=../..:$GUILE_LOAD_PATH run-theme-d-program \
         <program-name>
```

where `<program-name>` is the name of the target `.go` file of the program (`main.go` or `run-demo.go`). For program `hello` the command is

```
GUILE_LOAD_PATH=.:$GUILE_LOAD_PATH run-theme-d-program hello.go
```

If you have a Debian-based Linux operating system you have to ensure that package `gir1.2-gtk-3.0` is installed in your system.

The example programs `dbus-test-server1`, `dbus-test-client1`, `dbus-test-client1-2`, and `introspect2` are located in directory `/usr/share/doc/theme-d-intr-dev/examples/intr-dbus-examples` and in directory `examples/intr-dbus-examples` in the source package. You can build the example programs with command `make -f user.mk` The D-Bus test server and clients support also modular linking. In order to invoke this use command

```
LINK_MODULES=1 make -f user.mk
```

You can run the programs with command

```
GUILE_LOAD_PATH=../..:$GUILE_LOAD_PATH run-theme-d-program \
         <program-name>
```

In order to test the server and clients first launch the server in a terminal window and then the client in another terminal window. The functionality in `dbus-test-client1` and `dbus-test-client1-2` is identical but the latter program uses Theme-D wrapper library to access the server. Program `introspect2/main.go` takes two arguments: the D-Bus service name and object path, where the service has to implement interface `org.freedesktop.DBus.Introspectable`. The program prints the introspection data to the console.

Writing Programs Using Theme-D-Intr

It is recommended you create a new directory for your program. First you have to list the G-Golf classes and functions you use in file `intr-imports.scm`. Its format is:

```
(intr-entities
  (version namespace ver)
  ...
  (types
    (namespace type)
    ...)
  (only-types
    (namespace type)
    ...)
  (functions
    (namespace function)
    ...)
  (rejected-methods
    name ...)
  (overridden-functions
    (name (argument-type ...) result-type attributes)
    ...)
  (strip-boolean-result
    name ...)
  (ignore-slots
    (namespace class) slot-name ...)
    ...)
```

The order of the toplevel subexpressions has no effect. Symbol `classes` is accepted as an alias for `types`. Namespace is the library where the definitions are imported, such as `Gtk`. Class and type names are given in format `MyClassName`. Note that the namespace is not included in the class or function name (`Widget` instead of `GtkWidget`). Methods belonging to a type are imported automatically when the type is imported and they must not be listed in the `functions` section. If you want to import a type but not its methods list the type in section `only-types` instead of `types`. Rejected methods are generic function names for which we do not generate methods. There are two main reasons for rejecting a method:

- The method name overlaps a nongeneric function name, e.g. `append`.
- The methods of some name violate the covariance rule, see Theme-D Language Manual [https://www.iki.fi/tohoyn/theme-d/theme-d-language-manual-7.3.1.pdf].

The purpose of `overridden-functions` is to handle functions overridden in G-Golf. The argument types, the result type and the attributes are given in Theme-D format. The `version` specifies the version used for a namespace. The version must be enclosed in double quotes. The boolean function result of a function can be stripped by listing it under `strip-boolean-result`. See G-Golf documentation for this feature. Form `ignore-slots` can be used to prevent code generation for some specific slots. For example, `(ignore-slots (Gtk Label) xalign yalign)`. Here is an example import file:

```
(intr-entities
  (version Gtk "4.0")
  (types
    (Gdk Display)
    (Gtk Widget)
    (Gtk CssProvider)
    (Gtk StyleContext)
    (Gtk Application)
    (Gtk ApplicationWindow)
    (Gtk Button)
    (Gtk Box)
    (Gtk ScrolledWindow)
    (Gtk TextBuffer)
    (Gtk TextView)
    (Gtk Entry)
    (Gtk EntryBuffer)
    (Gtk Label)
    (Gtk Separator)
    (Gtk TextTagTable)
    (Gtk TextTag)
    (Gtk TextMark)))
```

and here is another:

```
(intr-entities
  (version Gtk "3.0")
  (types
    (Gtk Widget)
    (Gtk Window)
    (Gtk Button))
  (functions
    (Gtk init)
    (Gtk main)
    (Gtk main_quit))
  (rejected-methods
```

```
      append map get-style activate compare copy)
    (overridden-functions
      (gtk-container-child-get-property (<gtk-container>
        <gtk-widget> <string>)
        <object> nonpure)
      (gtk-container-child-set-property
        (<gtk-container> <gtk-widget> <string> <object>)
        <none> nonpure)))
```

The following three files are generated from the definition file:

- Interface file `_intr-imports.thi`
- Body file `_intr-imports.thb`
- Target module `_intr-imports-target.scm`

The program and modules using G-Golf should import the module `_intr-imports`. In case you have a multiple part program name, such as `(myexamples myprogram)`, module `_intr-imports` has a multiple part name too, e.g. `(myexamples _intr-imports)`. File `_intr-imports-target.scm` is used by the linked program to provide access to the introspected libraries.

The following commands are used to generate Theme-D-Intr files:

- `generate-intr-interface`
- `generate-intr-body`
- `generate-intr-target-module`

If you use a multiple part program name you must give the `-m` option for these commands. The option argument is the imports module name for the first two commands, e.g. `-m "(myexamples _intr-imports)"` and the setter module name for the fourth command, e.g. `-m "(myexamples _intr-imports-target)"`. If you want to generate accessor methods `NAMESPACE-CLASS-get-SLOT` and `NAMESPACE-CLASS-set-SLOT!` use option `--generate-accessors` for all of these programs. Note that you can access slots without accessor methods by using procedures `slot-ref` and `slot-set!`.

When you link a program using Theme-D-Intr you have to give the following options to the Theme-D linker:

- `-x "(g-golf)"`
- `-x "(guile-theme-d-intr support)"`
- `-x "<target-module>"`
- `--duplicates="merge-generics replace warn-override-core warn last"`

Here `<target-module>` is the Theme-D name of the generated target module. For a program with a single part name it is `(_intr-imports-target)`. If you use the extra support module give also option `-x "(guile-theme-d-intr support2-gtk3)"` or `-x "(guile-theme-d-intr support2-gtk4)"`.

You have to ensure that you have the introspection files for the external libraries your program uses (with G-Golf) installed in your system. If your program uses GTK and you have a Debian-based Linux system this can be ensured by having package `gir1.2-gtk-3.0` (for GTK 3.0) or `gir1.2-gtk-4.0` (for GTK 4.0) in your system.

See also the `user.mk` makefiles of the example programs. Example program `hello` has a single part program name and the other example programs multiple part program names.

When you run programs using Theme-D-Intr the environment variable `GUILE_LOAD_PATH` has to contain the root directory of your program. If you use a single part program name this is the directory of your program.

If you use modular linking you may have several introspection modules. You have to generate the interface file, body file, and target module for each of these modules. By convention the target module is named `__scm_`*`modulename`*`.scm`. You have to include all target modules needed when you link units modularly. Use linker option `-x` for this.

Distributing Programs Using Theme-D-Intr

If you use a Debian-based operating system the following packages are required to run compiled (built) Theme-D programs using Theme-D-Intr:

- `theme-d-rte`
- `th-scheme-utilities`
- `libthemedsupport`
- `theme-d-intr`

If you have a non-Debian operating system see the Theme-D User Guide for the files required to distribute a compiled Theme-D program. In addition to those, you need file `support.scm` from the Theme-D-Intr source package. If you use files `support2-gtk3.scm` or `support2-gtk4.scm` you need distribute that, too. These files have to be installed in subdirectory `guile-theme-d-support` somewhere in the Guile library search path. In UNIX systems this path usually contains directory `/usr/share/guile/site/3.0/` or `/usr/share/guile/site/2.2/`.
If your application uses modular linking you also have to distribute files

- `base/__intf_support.go`
- `base/__impl_support.go`
- `base/__intf_utils.go`
- `base/__impl_utils.go`
- `base/__intf_gobject.go`
- `base/__impl_gobject.go`
- `base/__intf_gobject.go`
- `base/__scm_gobject.go`
- `base/__intf_variant.go`
- `base/__impl_variant.go`
- `base/__scm_variant.go`

into subdirectory `theme-d-intr/base` of the Theme-D target library directory. If your application uses modular linking and D-Bus you also have to distribute files

- `dbus/__intf_connection.go`
- `dbus/__impl_connection.go`

- `dbus/__scm_connection.go`
- `dbus/__intf_node-info.go`
- `dbus/__impl_node-info.go`
- `dbus/__scm_node-info.go`
- `dbus/__intf_utils.go`
- `dbus/__impl_utils.go`
- `dbus/__scm_utils.go`

into subdirectory `theme-d-intr/dbus`. You can obtain the target library directory with command

```
get-theme-d-config-var target-lib-dir
```

Help with Callback Types

You can use command `describe-cb-type` to get the Theme-D name and Theme-D procedure type of a callback type. The syntax of the command is

```
describe-cb-type library callback-name [version]
```

Argument `library` is the library where the callback is defined. Argument `callback-name` is the name of the callback type. Optional argument `version` is the version of the library. Here is an example:

```
describe-cb-type Gtk CustomMeasureFunc 4.0
```

D-Bus Support

Wrappers to Introspected Methods and Signals

Theme-D-Intr implements wrappers for D-Bus methods and signals. They handle conversion between Theme-D objects and `<g-variant>`'s.

Theme-D wrapper modules are generated by programs `gen-dbus-theme-d-interface` and `gen-dbus-theme-d-body`. These programs take two arguments: the XML file containing the D-Bus interface description and the module path to be used in the generated modules. The XML file has to contain a single `node` element containing a single `interface` element. The module path has to be a list of symbols separated by spaces and enclosed in parentheses.

The wrappers are named as follows:

a synchronous method call:	*interface*`:`*method*
an asynchronous method call:	*interface*`:async:`*method*
signal subscription:	*interface*`:subscribe:`*signal*
property getter:	*interface*`:get:`*property*

property setter: *interface*`:set:`*property*

where *interface* is obtained by replacing points by underscores in the interface name.
The method calls and property getter calls unwrap the `<variant>` returned by the method and return the obtained value.
A synchronous method call wrapper takes the following arguments:

Name: `conn`
Type: `<g-dbus-connection>`
Description:
A D-Bus connection

Name: `str-bus-name`
Type: `<string>`
Description:
The bus name of the method call

Name: `str-object-path`
Type: `<string>`
Description:
The path of the object the method is applied to

Name: `flags`
Type: `<g-dbus-call-flags>`
Description:
Flags affecting the call.

Name: `i-timeout`
Type: `<integer>`
Description:
The timeout in milliseconds, -1 to use the default timeout. Use the value of Glib macro `G_MAXINT` for no timeout. This value is the largest value that C type `int` can contain and it is typically `0x7fffffff`.

Name: `cancellable`
Type: `(:alt-maybe <g-cancellable>)`
Description:
An object to control the cancellation of the method. The data is owned by the caller of the method. This argument can be `#f`.

Name: `l-args`
Type: `(:tuple ARG1 ... ARGN)` where `ARGK` are the argument types of the method.
Description:
Arguments to be passed to the method. If there are no arguments to pass this argument is omitted.

An asynchronous method call wrapper takes the following arguments:

Arguments `conn`, `str-bus-name`, `str-object-path`, `flags`, `i-timeout`, and `cancellable` as in the synchronous case.

Name: `proc-result`
Type: `(:alt-maybe (:procedure ((:alt-maybe <gobject>)` *result-type* `<object>) <none> nonpure)` where *result-type* is the Theme-D type of the method return value. If the method does not return any value the second argument is omitted.
Description:
A callback to handle the result value. Arguments:

Name: `source_object`
Type: `(:alt-maybe <gobject>)`

Description:
The object the asynchronous operation was started with

Name: `result`
Type: *result-type*
Description:
The result of the method call

Name: `data`
Type: `<object>`
Description:
Auxiliary data

Name: `l-args`
Type: `(:tuple ARG1 ... ARGN)` where `ARGK` are the argument types of the method.
Description:
Arguments to be passed to the method. If there are no arguments to pass this argument is omitted.

A signal subscription takes the following arguments:

Arguments `conn`, `str-bus-name`, and `str-object-path` as before.

Name: `str-arg0`
Type: `(:alt-maybe <string>)`
Description:
Contents of first string argument to match on or `#f` to match on all kinds of arguments.

Name: `flags`
Type: `<g-dbus-signal-flags>`
Description:
Flags.

Name: `proc-callback`
Type: `(:procedure (<g-dbus-connection> (:alt-maybe <string>) <string> <string> <string> (:tuple ARG1 ... ARGN) <object>) <none> nonpure)` where `ARGK` are the argument types (Theme-D types) of the signal. If the signal takes no arguments the sixth argument is omitted.
Description:
Callback to be called when the signal is emitted. Arguments:

Name: `connection`
Type: `<g-dbus-connection>`
Description:
A D-Bus connection

Name: `sender_name`
Type: `(:alt-maybe <string>)`
Description:
The unique bus name of the sender of the signal, or `#f` on a peer-to-peer D-Bus connection

Name: `interface_name`
Type: `<string>`
Description:
The name of the interface the signal belongs to

Name: `signal_name`
Type: `<string>`

Description:
The name of the signal

Name: `l-args`
Type: `(:tuple ARG1 ... ARGN)`
Description:
Arguments to the signal. If the signal takes no arguments this argument is omitted.

Name: `user_data`
Type: `<object>`
Description:
Auxiliary data

Name: `x-user-data`
Type: `<object>`
Description:
Auxiliary data.

Name: `proc-user-data-free`
Type: `(:alt-maybe <g-lib-destroy-notify>)`
Description:
Procedure to free user data.

A property getter takes arguments `conn`, `str-bus-name`, `str-object-path`, `flags`, `i-timeout`, and `cancellable` that are defined as before.

A property setter takes arguments `conn`, `str-bus-name`, `str-object-path`, `x-value`, `flags`, `i-timeout`, and `cancellable` where the type of `x-value` is the Theme-D type of the property and other arguments are defined as before. Argument `x-value` is used to set the property.

As an example, consider the example program `introspect2/main.go`. The introspection specification is

```
(intr-entities
  (types
    (GLib Variant)
    (GLib VariantType)
    (Gio DBusConnection)
    (Gio BusType)
    (Gio DBusMessage))
  (functions
    (Gio bus_get_sync)))
```

The generated method declaration is

```
(declare-simple-method org_freedesktop_DBus_Introspectable:Introspect
  (<g-dbus-connection>
   <string>
   <string>
   <g-dbus-call-flags>
   <integer>
   (:alt-maybe <g-cancellable>))
  <string>
  nonpure)
```

and the method call

```
(org_freedesktop_DBus_Introspectable:Introspect
 conn
 str-service
 str-object
 '()
 -1
 #f)
```

D-Bus API

See Gio documentation [https://docs.gtk.org/gio/index.html] for more detailed description of the GDBus API. See https://docs.gtk.org/glib/struct.Variant.html for description of variant API. Virtual methods with short names are also defined for the procedures. A short name is obtained from the long name by stripping the type prefix away, e.g. `g-dbus-method-invocation-return-value` becomes `return-value`.

Most Important D-Bus Types

<g-bus-name-owner-flags>

Metatype: flags
Description:
Flags used in `g-bus-own-name-with-closures`. Zero or more of symbols `none`, `allow-replacement`, `replace`, and `do-not-queue`.

<g-bus-type>

Metatype: enumeration
Description:
Bus type. Possible values are `starter`, `none`, `system`, and `session`.

<g-cancellable>

Metatype: class
Description:
An object controlling cancellation of operations.

<g-dbus-call-flags>

Metatype: flags
Description:
Flags used in `g-dbus-connection-call`. Zero or more of symbols `none`, `no-auto-start`, and `allow-interactive-authorization`.

<g-dbus-connection>

Metatype: class
Description:
A D-Bus connection

<g-dbus-interface-info>

Metatype: opaque struct
Description:
Information about a D-Bus interface

<g-dbus-method-invocation>

Metatype: class
Description:

Instances of the <g-dbus-method-invocation> class are used when handling D-Bus method calls. It provides a way to asynchronously return results and errors.

<g-dbus-node-info>

Metatype: opaque struct
Description:
Information about nodes in a remote object hierarchy

<g-dbus-signal-flags>

Metatype: flags
Description:
Flags used when subscribing to signals via g-dbus-connection-signal-subscribe. Zero or more of symbols none, no-match-rule, match-arg0-namespace, and match-arg0-path.

<g-lib-destroy-notify>

Metatype: procedure type
Description:
A procedure used to delete a data element. Equal to (:procedure (<object>) <none> nonpure).

<general-procedure>

Metatype: procedure type
Description:
Equal to (:procedure ((rest <object>)) <object> nonpure).

<gio-async-ready-callback>

Metatype: procedure type
Description:
Procedure to be called when an asynchronous operation has been completes. Equal to (:procedure ((:alt-maybe <gobject>) <object> <object>) <none> nonpure).
The arguments are:

Name: source_object
Type: (:alt-maybe <gobject>)
Description:
The object the asynchronous operation was started with

Name: res
Type: <object>
Description:
An async ready object to be used in g-dbus-connection-call-finish.

Name: data
Type: <object>
Description:
Auxiliary data

<gio-d-bus-signal-callback>

Metatype: procedure type
Description:
A callback procedure to be used in g-dbus-connection-signal-subscribe. Equal to (:procedure (<g-dbus-connection> (:alt-maybe <string>)

`<string> <string> <string> <g-variant> <object>) <none> nonpure)`.
The arguments are:

Name: `connection`
Type: `<g-dbus-connection>`
Description:
A D-Bus connection

Name: `sender_name`
Type: `(:alt-maybe <string>)`
Description:
The unique bus name of the sender of the signal, or `#f` on a peer-to-peer D-Bus connection

Name: `interface_name`
Type: `<string>`
Description:
The name of the interface the signal belongs to

Name: `signal_name`
Type: `<string>`
Description:
The name of the signal

Name: `parameters`
Type: `<g-variant>`
Description:
A `<g-variant>` tuple with parameters for the signal.

Name: `user_data`
Type: `<object>`
Description:
Auxiliary data

Most Important D-Bus Procedures

g-bus-get-sync

Synchronously connects to the message bus specified by `bus_type`.
Arguments:

Name: `bus_type`
Type: `<g-bus-type>`
Description:
The type of bus to own a name on

Name: `cancellable`
Type: `(:alt-maybe <g-cancellable>)`
Description:
An object to control the cancellation of the connection. The data is owned by the caller of the method. This argument can be `#f`.

Result value: A D-Bus connection
Result type: `<g-dbus-connection>`

g-bus-own-name-with-closures

Requests ownership of a name
Arguments:

Name: `bus_type`
Type: `<g-bus-type>`
Description:
The type of bus to own a name on

Name: `name`
Type: `<string>`
Description:
The name to own

Name: `flags`
Type: `<g-bus-name-owner-flags>`
Description:
Flags with ownership options

Name: `bus_acquired_closure`
Type: `(:alt-maybe <general-procedure>)`
Description:
Closure to invoke when connected to the bus of type bus_type, or `#f` to ignore. This procedure shall take the following arguments:

Name: `connection`
Type: `<g-dbus-connection>`
Description:
A D-Bus connection

Name: `name`
Type: `<string>`
Description:
The name that is requested to be owned

Name: `name_acquired_closure`
Type: `(:alt-maybe <general-procedure>)`
Description:
Procedure to invoke when name is acquired, or `#f` to ignore. This procedure shall take the following arguments:

Name: `connection`
Type: `<g-dbus-connection>`
Description:
A D-Bus connection

Name: `name`
Type: `<string>`
Description:
The name being owned

Name: `name_lost_closure`
Type: `(:alt-maybe <general-procedure>)`
Description:
Procedure to invoke when name is lost, or `#f` to ignore. This procedure shall take the following arguments:

Name: `connection`
Type: `<g-dbus-connection>`
Description:
A D-Bus connection

Name: `name`

Type: `<string>`
Description:
The name being owned

Result value: An identifier (never 0) that can be used with `g-bus-unown-name` to stop owning the name.
Result type: `<integer>`

g-dbus-connection-call

This procedure calls a method asynchronously.
Arguments:

Name: `connection`
Type: `<g-dbus-connection>`
Description:
A D-Bus connection

Name: `bus_name`
Type: `<string>`
Description:
The bus name of the method call

Name: `object_path`
Type: `<string>`
Description:
The path of the object the method is applied to

Name: `interface_name`
Type: `<string>`
Description:
The name of the interface containing the method

Name: `method_name`
Type: `<string>`
Description:
The name of the method

Name: `parameters`
Type: `(:alt-maybe <g-variant>)`
Description:
The parameters to be passed to the method. This argument has to be a `<g-variant>` tuple or `#f`.

Name: `reply_type`
Type: `(:alt-maybe <g-variant-type>)`
Description:
The type of the result value of the method. This value has to be either a tuple variant type or `#f`.

Name: `flags`
Type: `<g-dbus-call-flags>`
Description:
Flags affecting the call.

Name: `timeout_msec`
Type: `<integer>`
Description:

The timeout in milliseconds, -1 to use the default timeout. Use the value of Glib macro `G_MAXINT` for no timeout. This value is the largest value that C type `int` can contain and it is typically 0x7fffffff.

Name: `cancellable`
Type: `(:alt-maybe <g-cancellable>)`
Description:
An object to control the cancellation of the method. The data is owned by the caller of the method. This argument can be `#f`.

Name: `callback`
Type: `(:alt-maybe <gio-async-ready-callback>)`
Description:
A callback to handle the result of the method.

Name: `user_data`
Type: `<object>`
Description:
Data to be passed to the callback.

Result value: None
Result type: `<none>`

g-dbus-connection-call-finish

This procedure finishes the method call started by `g-dbus-connection-call`
Arguments:

Name: `connection`
Type: `<g-dbus-connection>`
Description:
A D-Bus connection

Name: `res`
Type: `<object>`
Description:
An object obtained from the callback passed to `g-dbus-connection-call`.

g-dbus-connection-call-sync

This procedure calls a method synchronously.
Arguments:

Name: `connection`
Type: `<g-dbus-connection>`
Description:
A D-Bus connection

Name: `bus_name`
Type: `<string>`
Description:
The bus name of the method call

Name: `object_path`
Type: `<string>`
Description:
The path of the object the method is applied to

Name: `interface_name`
Type: `<string>`

Description:
The name of the interface containing the method

Name: `method_name`
Type: `<string>`
Description:
The name of the method

Name: `parameters`
Type: `(:alt-maybe <g-variant>)`
Description:
The parameters to be passed to the method. This argument has to be a `<g-variant>` tuple or `#f`.

Name: `reply_type`
Type: `(:alt-maybe <g-variant-type>)`
Description:
The type of the result value of the method. This value has to be either a tuple variant type or `#f`.

Name: `flags`
Type: `<g-dbus-call-flags>`
Description:
Flags affecting the call.

Name: `timeout_msec`
Type: `<integer>`
Description:
The timeout in milliseconds, -1 to use the default timeout. Use the value of Glib macro `G_MAXINT` for no timeout. This value is the largest value that C type `int` can contain and it is typically 0x7fffffff.

Name: `cancellable`
Type: `(:alt-maybe <g-cancellable>)`
Description:
An object to control the cancellation of the method. The data is owned by the caller of the method. This argument can be `#f`.

Result value: The value returned from the method
Result type: `<g-variant>`

g-dbus-connection-emit-signal

This procedure emits a signal.
Arguments:

Name: `connection`
Type: `<g-dbus-connection>`
Description:
A D-Bus connection

Name: `destination_bus_name`
Type: `(:alt-maybe <string>)`
Description:
The unique bus name for the destination for the signal or `#f` to emit to all listeners.

Name: `object_path`
Type: `<string>`
Description:

The object emitting the signal.

Name: `interface_name`
Type: `<string>`
Description:
D-Bus interface to emit a signal on.

Name: `signal_name`
Type: `<string>`
Description:
The name of the signal.

Name: `parameters`
Type: `(:alt-maybe <g-variant>)`
Description:
The parameters to the signal. This argument has to be a `<g-variant>` tuple or `#f` if there are no parameters.

Result value: `#t` if there were no errors emitting the signal.
Result type: `<boolean>`

g-dbus-connection-register-object-with-closures2

Registers callbacks for exported objects.
Arguments:

Name: `connection`
Type: `<g-dbus-connection>`
Description:
A D-Bus connection

Name: `object_path`
Type: `<string>`
Description:
The object path to register at.

Name: `interface_info`
Type: `<g-dbus-interface-info>`
Description:
Introspection data for the interface.

Name: `method_call_closure`
Type: `(:alt-maybe <general-procedure>)`
Description:
Procedure for handling incoming method calls. This procedure shall take the following arguments:

Name: `connection`
Type: `<g-dbus-connection>`
Description:
A D-Bus connection

Name: `sender`
Type: `(:alt-maybe <string>)`
Description:
The unique bus name of the remote caller, or `#f` if not specified by the caller, e.g. on peer-to-peer connections

Name: `object_path`

Type: `<string>`
Description:
The object path that the method was invoked on.

Name: `interface_name`
Type: `(:alt-maybe <string>)`
Description:
The D-Bus interface name the method was invoked on, or `#f` if not specified by the sender

Name: `method_name`
Type: `<string>`
Description:
The name of the method

Name: `parameters`
Type: `<g-variant>`
Description:
A `<g-variant>` tuple with parameters to the method

Name: `invocation`
Type: `<g-dbus-method-invocation>`
Description:
A `<g-dbus-method-invocation>` object that must be used to return a value or error.

Name: `get_property_closure`
Type: `(:alt-maybe <general-procedure>)`
Description:
Procedure for getting a property. This procedure shall take the following arguments:

Name: `connection`
Type: `<g-dbus-connection>`
Description:
A D-Bus connection

Name: `sender`
Type: `(:alt-maybe <string>)`
Description:
The unique bus name of the remote caller, or `#f` if not specified by the caller, e.g. on peer-to-peer connections

Name: `object_path`
Type: `<string>`
Description:
The object path that the method was invoked on.

Name: `interface_name`
Type: `(:alt-maybe <string>)`
Description:
The D-Bus interface name the method was invoked on, or `#f` if not specified by the sender

Name: `property_name`
Type: `<string>`
Description:
The name of the property

Name: `set_property_closure`
Type: `(:alt-maybe <general-procedure>)`
Description:
Procedure for setting a property. This procedure shall take the following arguments:

Name: `connection`
Type: `<g-dbus-connection>`
Description:
A D-Bus connection

Name: `sender`
Type: `(:alt-maybe <string>)`
Description:
The unique bus name of the remote caller, or `#f` if not specified by the caller, e.g. on peer-to-peer connections

Name: `object_path`
Type: `<string>`
Description:
The object path that the method was invoked on.

Name: `interface_name`
Type: `(:alt-maybe <string>)`
Description:
The D-Bus interface name the method was invoked on, or `#f` if not specified by the sender

Name: `property_name`
Type: `<string>`
Description:
The name of the property

Name: `value`
Type: `<g-variant>`
Description:
The value to set the property to.

Result value: 0 if there was an error, otherwise a registration ID (never 0)
Result type: `<integer>`

g-dbus-connection-signal-subscribe

Subscribes to signal on the connection and invokes a callback whenever the signal is received.
Arguments:

Name: `connection`
Type: `<g-dbus-connection>`
Description:
A D-Bus connection

Name: `sender`
Type: `(:alt-maybe <string>)`
Description:
Sender name to match on or `#f` to match all senders

Name: `interface_name`
Type: `(:alt-maybe <string>)`
Description:
Interface name to match on or `#f` to match all interfaces

Name: `member`
Type: `(:alt-maybe <string>)`
Description:
Signal name to match on or `#f` to match all signals

Name: `object_path`
Type: `(:alt-maybe <string>)`
Description:
Object paths to match on or `#f` to match all object paths

Name: `arg0`
Type: `(:alt-maybe <string>)`
Description:
Contents of the first string argument to match on or `#f` to match all kinds of arguments

Name: `flags`
Type: `<g-dbus-signal-flags>`
Description:
Flags affecting the subscription

Name: `callback`
Type: `<gio-d-bus-signal-callback>`
Description:
Callback to invoke when there is a signal matching the requested data.

Name: `user_data`
Type: `<object>`
Description:
Data to be passed to the callback.

Name: `user_data_free_func`
Type: `(:alt-maybe <g-lib-destroy-notify>)`
Description:
Procedure to free the user data.

Result value: A subscription identifier that can be used to unsubscribe from the signal with `g-dbus-connection-signal-unsubscribe`.
Result type: `<integer>`

g-dbus-connection-signal-unsubscribe

Ubsubscribes from signals.
Arguments:

Name: `connection`
Type: `<g-dbus-connection>`
Description:
A D-Bus connection

Name: `subscription_id`
Type: `<integer>`
Description:
A subscription id obtained from `g-dbus-connection-signal-subscribe`

Result value: None
Result type: `<none>`

g-dbus-connection-unregister-object

Ubsubscribes from signals.

Arguments:

Name: `connection`
Type: `<g-dbus-connection>`
Description:
A D-Bus connection

Name: `registration_id`
Type: `<integer>`
Description:
A registration id obtained from `g-dbus-connection-register-object-with-closures2`

Result value: `#t` if the object was unregistered, `#f` otherwise
Result type: `<boolean>`

g-dbus-method-invocation-return-value

Finishes handling of a method call by returning parameters.
Arguments:

Name: `invocation`
Type: `<g-dbus-method-invocation>`
Description:
D-Bus method invocation object

Name: `parameters`
Type: `(:alt-maybe <g-variant>)`
Description:
The parameters to return. This value has to be a `<g-variant>` tuple or `#f`.

Result value: None
Result type: `<none>`

g-dbus-node-info-lookup-interface

Looks up interface from node info.
Arguments:

Name: `info`
Type: `<g-dbus-node-info>`
Description:
D-Bus node info

Name: `name`
Type: `<string>`
Description:
D-Bus interface name

Result value: A `<g-dbus-interface-info>` or `#f` if no interface found
Result type: `(:alt-maybe <g-dbus-interface-info>)`

g-dbus-node-info-new-for-xml

Creates node info representing XML code.
Arguments:

Name: `xml_data`
Type: `<string>`
Description:
XML code

Result value: A `<g-dbus-node-info>` structure
Result type: `<g-dbus-node-info>`

Other Things

Command

```
dbus-introspect bus-name object
```

prints an XML specification of the D-Bus interfaces supported by a D-Bus object.