gi-docgen

Release 2021.1

Emmanuele Bassi

Sep 27, 2021

CONTENTS:

CHAPTER

ONE

USING GI-DOCGEN

In order to use GI-DocGen, you will need:

- a library using GObject and generating introspection data as part of its build process
- a project configuration file

For the former, you should read the gobject-introspection documentation, which includes all the details on how to write introspectable API.

1.1 Writing a project configuration file

The project configuration file provides some basic information describing your project, expressed in key/value pairs, and will be exposed to the template system used when generating the API reference through gi-docgen. Not every key is mandatory, and the template will decide whether or not use its value when generating the API reference. For simplicity, we're going to assume you're using the "basic" template that is part of gi-docgen.

The project configuration file is written using ToML, and you can use the `--config` command line option for gi-docgen.

We begin with the `library` preamble:

```
[library]
description = "The GTK toolkit"
authors = "GTK Development Team"
license = "GPL-2.1-or-later"
browse_url = "https://gitlab.gnome.org/GNOME/gtk/"
repository_url = "https://gitlab.gnome.org/GNOME/gtk.git"
website_url = "https://www.gtk.org"
```

The keys above will be used in the main landing page for the library.

If your project has dependencies, and you wish to display them or cross-link types and symbols from your API reference, you will need to describe them using the `dependencies` key, for instance:

```
# List the dependencies using their GIR namespace
dependencies = [
  "GObject-2.0",
  "Graphene-1.0",
  "Pango-1.0",
  "Gdk-4.0",
  "Gsk-4.0",
]
```

Each dependency will need its own object, for instance:

```
[dependencies."GObject-2.0"]
name = "GObject"
description = "The base type system library"
docs_url = "https://developer.gnome.org/gobject/stable"
```

The `name`, `description`, and `docs_url` keys will be used when generating the list of dependencies on the main landing page.

If you wish to add links to the source code repository for type and symbol declarations, as well as the location of the documentation source, you will need a `source-location` section:

```
[source-location]
# The base URL for the web UI
base_url = "https://gitlab.gnome.org/GNOME/gtk/-/blob/master/"
# The format for links, using "filename" and "line" for the format
file_format = "{filename}#L{line}"
```

If your library has additional content, in the form of Markdown files that you wish to include in the generated API reference, you can use the `extra` section:

```
[extra]
# A list of Markdown files; they will be parsed using the
# same rules as the documentation coming from the introspection
# data. The path of each file is relative to the content
# directory specified on the command line.
#
# The order in which they are included will be used when
# generating the index.
#
# The generated files will be placed in the root output directory
content_files = [
  "getting_started.md",
  "building.md",
  "compiling.md",
  "running.md",
  "question_index.md",
  ...
]
# Additional images referenced by the documentation; their path
# is relative to the content directory specified on the command
# line.
#
# The image files will be copied into the root documentation,
# without replicating the directory structure in which they
# are listed.
content_images = [
  "images/aboutdialog.png",
  "images/action-bar.png",
  "images/appchooserbutton.png",
  "images/appchooserdialog.png",
  ...
]
```

For more information about the project configuration, please see the :doc:`project-configuration` page.

1.2 Generating the API reference

Once you have a project configuration file, and the introspection data for the library you wish to document, all you need is to launch the `gi-docgen` command line tool.

You will need to provide:

- the location of the project configuration file
- the location of the additional content files
- additional search paths for the dependencies
- the output directory for the generated files
- the location of the introspection file

A simple invocation for the installed `Gtk-4.0.gir` file is:

```
gi-docgen generate -C gtk4.toml /usr/share/gir-1.0/Gtk-4.0.gir
```

This will generate the API reference for the `Gtk-4.0` namespace, and will put the generate files under the current directory.

CHAPTER

TWO

PROJECT CONFIGURATION

Projects using gi-docgen should provide their own configuration file to describe how to generate their API reference.

The configuration file format uses ToML to provide key and value pairs that will be used by gi-docgen and, optionally, by the templates themselves.

Project configuration takes precendence over gi-docgen's defaults, but can be overridden by command line options, where applicable.

2.1 Standard sections and keys

2.1.1 The `library` section

The `library` section is used to define the library configuration values that gi-docgen will pass to the templates, as well as configuration switches that control the files generated by gi-docgen.

The following keys are used, if found:

`version` = **`s`** The version of the library. This is the actual version of the shared library, as opposed to the version of the API as represented by the namespace.

`authors` = **`s`** The name of the authors of the library, as a string.

`license` = **`s`** The license of the documentation, as an SPDX identifier.

`website_url` = **`s`** The website for the library.

`browse_url` = **`s`** The website that can be used to browse the source code of the library.

`logo_url` = **`s`** The location of a logo image. This can be a local file, or a URL.

`description` = **`s`** A short description of the library.

`dependencies` = **`dict(s, dict(s, s)`** A dictionary of dependencies; each entry in the dictionary has a key in the form of `{namespace}-{version}`, and values in the form of a dictionary with the following keys: `name`, `description`, and `docs_url`.

`devhelp` = **`b`** Whether gi-docgen should generate a DevHelp file for the namespace.

`search_index` = **`b`** Whether gi-docgen should generate a search index file for the namespace.

2.1.2 The `theme` section

The `theme` section is used to define the theme being used by gi-docgen when generating the API reference of a project.

The following keys are used, if found:

`templates_dir` = **s** The directory that contains the templates to be used by gi-docgen. The default directory is inside the gi-docgen module directory. This key can be overridden by the `--templates-dir` command line argument.

`name` = **s** The name of the template to use. The name is a sub-directory of the `template_dir` directory, and will be used to load the template's configuration file. This key can be overridden by the `--theme-name` command line argument.

`show_index_summary` = **b** A boolean value that controls whether to show the summary of each symbol in the namespace index.

`show_class_hierarchy` = **b** A boolean value that controls whether to generate a class graph with the ancestors of a type, as well as the implemented interfaces. Requires the `dot` utility from GraphViz installed in the `PATH`.

2.1.3 The `source-location` section

The `source-location` section is used to define the location of the source code repository of a project to allow gi-docgen to create links from the API reference to the definition of symbols and the source of the documentation stanzas.

The following keys are used, if found:

`base_url` = **s** The base URL for accessing a file in the source code repository.

`file_format` = **s** The format string used to point to a file, and a line in that file; the string can contain the token `{filename}`, which will be replaced with the basename of the file; and the token `{line}`, which will be replaced with the line in the file. The default value for this key is: `{filename}#L{line}`.

2.1.4 The `extra` section

The `extra` section is used to define additional content used when generating the API reference of a project.

The following keys are used, if found:

`content_files` = **`list(s)`** A list of tuples. The first element of the tuple is a Markdown file name, relative to the directories specified by the `--content-dir` command line arguments; the second element of the tuple is the title used for the link to the content file. When generating the API reference, gi-docgen will transform the Markdown file into an HTML one, using the same pre-processing filters applied to the documentation blocks found in the introspection data. The generated HTML files will be placed in the root directory of the namespace.

`content_images` = **`list(s)`** A list of files, relative to the directories specified by the `--content-dir` command line arguments. The files will be copied in the root directory of the namespace.

`urlmap_file` = **s** Path of a JavaScript file that defines the mapping from namespaces to url prefixes for resolving links to external symbols, as a JavaScript map with the name *baseURLs*:

```
baseURLs = [
  [ 'Pango', 'https://gnome.pages.gitlab.gnome.org/pango/Pango/' ],
  [ 'PangoCairo', 'https://gnome.pages.gitlab.gnome.org/pango/PangoCairo/' ],
]
```

2.2 Symbol overrides

2.2.1 Visibility

It is possible to override the visibility of types, properties, and symbols in the introspection data from within the project configuration file.

The following example will hide the type `Protected`:

```
[[object]]
name = "Protected"
hidden = true
```

The type will be skipped when generating the API reference and the search index. This annotation applies to all possible top-level types:

- aliases
- bitfields
- callbacks
- classes
- domains
- enums
- functions
- function macros
- interfaces
- records
- unions

The `object` key is always an array of dictionaries; each element in the array can have a `name` key, used to match the object name exactly; or a `pattern` key, which uses a regular expression to match the object name.

Each object can contain the following keys:

- `name`: the name of the symbol to match exactly
- `pattern`: a regular expression to match the symbol name
- `hidden`: whether the symbol should be hidden from the documentation
- `check_ignore`: whether the symbol should be skipped when checking the documentation

Each element can also have the following sections:

- `property`
- `signal`
- `constructor`
- `method`
- `function`

Each one of these sections can contain array of objects.

The following example will hide the `backend` property on the `Printer` type:

```
[[object]]
name = "Printer"

  [[object.property]]
  name = "backend"
  hidden = true
```

The following example will hide the `private-changed` signal on the `StyleProvider` type:

```
[[object]]
name = "StyleProvider"

  [[object.signal]]
  name = "private-changed"
  hidden = true
```

The following example will skip the `quark` function on the `ParserError` type when checking the documentation:

```
[[object]]
name = "ParserError"

  [[object.function]]
  name = "quark"
  check_ignore = true
```

CHAPTER

THREE

LINKING ITEMS BY NAME

Gi-docgen is capable of linking symbols across the same introspected namespace, by using a qualifier fragment and the symbol name.

For instance:

```
/**
 * ExampleFoo:
 *
 * This structure is related to [struct@Bar].
 */

 /**
  * example_foo_set_bar:
  *
  * Sets [struct@Example.Bar] on an instance of `Foo`.
  */

 /**
  * ExampleFoo:bar:
  *
  * Sets an instance of [`Bar`](struct.Bar.html) on `Foo`.
  */
```

will all link to `Bar`.

Backticks will be stripped, so `` [`class@Foo`] `` will correctly link to `Foo`.

The link can either be a fully qualified name, which includes the namespace; or a name relative to the current namespace; for instance, both of the following links will point to `ExampleFoo` when generating the documentation for the "Example" namespace:

- `[class@Foo]`
- `[class@Example.Foo]`

The available qualifier fragments are:

Fragment	Description	Example
alias	An alias to another type	[alias@Allocation]
callback	A callback type	[callback@Gtk.ListBoxForeachFunc]
class	An object class	[class@Widget], [class@Gdk.Surface], [class@Gsk.RenderNode]
const	A constant or pre-processor symbol	[const@Gdk.KEY_q]
ctor	A constructor function	[ctor@Gtk.Box.new], [ctor@Button.new_with_label]
enum	A plain enumeration	[enum@Orientation]
error	A GError domain enumeration	[error@Gtk.BuilderParseError]
flags	A bitfield	[flags@Gdk.ModifierType]
func	A global or a type function	[func@Gtk.init], [func@show_uri], [func@Gtk.Window.list_toplevels]
iface	A GTypeInterface	[iface@Gtk.Buildable]
method	An instance or class method	[method@Gtk.Widget.show], [method@WidgetClass.add_binding]
property	A GObject property	[property@Gtk.Orientable:orientation]
signal	A GObject signal	[signal@Gtk.RecentManager::changed]
struct	A plain C structure or union	[struct@Gtk.TextIter]
vfunc	A virtual function in a class or interface	[vfunc@Gtk.Widget.measure]

The generic type fragment, followed by a type, will look up the given type and generate the appropriate link for it. The type can be fully qualified or relative to the current namespace:

```
// Equivalent to [class@Gtk.Window]
[type@Gtk.Window]

// Equivalent to [enum@Gtk.Orientation]
[type@Gtk.Orientation]
```

Anything that is a known type—aliases, callbacks, classes, constants, enumerations, interfaces, structures—can be linked using the type fragment.

Additionally, the id fragment, followed by a C symbol identifier, will try to link to the function; for instance:

```
// Equivalent to [func@Gtk.show_uri], will link to gtk_show_uri()
[id@gtk_show_uri]

// Equivalent to [method@Gtk.Widget.show], will link to gtk_widget_show()
[id@gtk_widget_show]
```

The id fragment can only be used for symbols within the current namespace.

It's important to note that the method and func fragments can have multiple meanings:

- the method fragment will match both instance and class methods, depending on the type used; for instance, to match an instance method you should use the type name, and to match a class method you should use the class name. The class method should not be confused with the vfunc fragment, which uses the type name and links to virtual methods defined in the class or interface structure. Class methods take the class pointer as their first argument, whereas virtual methods take the instance pointer as their first argument.

```
// will link to gtk_widget_show()
[method@Gtk.Widget.show]

```

(continues on next page)

(continued from previous page)

```
// will link to gtk_widget_class_add_binding()
[method@Gtk.WidgetClass.add_binding]

// will link to GtkWidgetClass.show
[vfunc@Gtk.Widget.show]
```

- similarly, the `func` fragment will match global functions and type functions, depending on whether the link contains a type or not. Additionally, `func` will match function macros, which are part of the global namespace.

```
// will link to gtk_show_uri()
[func@Gtk.show_uri]

// will link to gtk_window_list_toplevels()
[func@Gtk.Window.list_toplevels]

// will link to gtk_widget_class_bind_template_child()
[func@Gtk.widget_class_bind_template_child]
```

3.1 External Links

Gi-docgen can use the same syntax to point to symbols in other namespaces with gi-docgen-generated documentation, as long as you provide it with a mapping from the namespace names to a base url for the docs. This is done by defining a JavaScript map called `baseURLs` like this:

```
baseURLs = [
  [ 'Pango', 'https://gnome.pages.gitlab.gnome.org/pango/Pango/' ],
  [ 'PangoCairo', 'https://gnome.pages.gitlab.gnome.org/pango/PangoCairo/' ],
]
```

And specifying the path of the JavaScript file into the `extras` section of the project configuration, in the `urlmap_file` key.

CHAPTER

FOUR

INTROSPECTION ATTRIBUTES

GI-DocGen consumes the following attributes found in the introspection data when generating the API reference for that data.

4.1 Properties

The following attributes apply to properties.

`org.gtk.Property.get` = **s** Defines the getter method for a given property. The value of the attribute is the C symbol of the function.

`org.gtk.Property.set` = **s** Defines the setter method for a given property. The value of the attribute is the C symbol of the function.

4.2 Methods

The following attributes apply to methods of a classed type or interface.

`org.gtk.Method.set_property` = **s** Defines the property set by the function. The property name must be in the same type as the method

`org.gtk.Method.get_property` = **s** Defines the property retrieved by the function. The property name must be in the same type as the method

`org.gtk.Method.signal` = **s** Defines the signal emitted by the function. The signal name must be in the same type as the method

CHAPTER FIVE

TEMPLATES

The *generate* command of gi-docgen uses Jinja2 templates to generate the HTML pages of the API reference from the introspection data provided by a library.

5.1 Template configuration

Each template must contain a template configuration file, with the same name as the template all in lower case. The template configuration format is ToML.

The template configuration file can contain the following sections:

5.1.1 The `metadata` section

Contains template metadata, like licensing and author information:

`name` = **s** The name of the template

`author_name` = **s** The name of the author of the template

`author_email` = **s** The email of the author of the template

`copyright_year` = **s** The copyright year of the template

`license` = **s** The license of the template, as an SPDX identifier.

5.1.2 The `templates` section

Contains the template files for each section of the template. If the key is not present, the default file name is used.

`class` = **s** The class template file. Default: `class.html`

`interface` = **s** The interface template file. Default: `interface.html`

`property` = **s** The property template file. Default: `property.html`

`signal` = **s** The signal template file. Default: `signal.html`

`method` = **s** The method template file. Default: `method.html`

`vfunc` = **s** The virtual method template file. Defalt: `vfunc.html`

`type_func` = **s** The type function template file. Default: `type_func.html`

`ctor` = **s** The constructor function template file. Default: `type_func.html`

`class_method` = **s** The class method template file. Default: `class_method.html`

error = s The error domain template file. Default: `error.html`

flags = s The bitfield template file. Default: `flags.html`

enum = s The enumeration template file. Default: `enum.html`

record = s The record template file. Default: `record.html`

union = s The union template file. Default: `union.html`

alias = s The alias template file. Default: `alias.html`

function = s The function template file. Default: `function.html`

constant = s The constant template file. Default: `constant.html`

namespace = s The namespace template file. Default: `namespace.html`

content = s The template file for additional content. Default: `content.html`

5.1.3 The `css` section

Contains style related data.

style = s The main CSS file for the template

5.1.4 Th `extra_files` section

Contains additional files that must be copied into the output directory after generating the reference.

files = list(s) A list of files needed by the template. Each file is relative to the template's directory.

5.2 Template data

Each Jinja template file will be passed objects and additional data when gi-docgen renders the API reference.

All templates will receive:

- the `CONFIG` object, containing the project configuration
- the `namespace` object, containing the GIR namespace

Additionally, each template will receive a template object containing the information needed to render the template.

CHAPTER

SIX

COMMANDS

6.1 gi-docgen generate

6.1.1 Generating the API reference from introspection data

SYNOPSIS

gi-docgen generate [OPTIONS. . .] [GIRFILE]

DESCRIPTION

The **generate** command generates the API reference from a GIR file.

GIR files are XML files that describe an API in a machine readable way, and are typically provided by a GObject library.

OPTIONS

`--add-include--path DIR` Adds `DIR` to the list of paths used to find introspection data files included in the given `GIRFILE`. The default search path for GIR files is `$XDG_DATA_DIRS/gir-1.0` and `$XDG_DATA_HOME/gir-1.0`; this option is typically used to include uninstalled GIR files, or non-standard locations.

`-C, --config FILE` Loads a project configuration file.

`--dry-run` Only load the introspection data, without generating the reference.

`--templates-dir DIR` Look for templates under `DIR`. The default location for the templates directory is inside the `gi-docgen` installation.

`--content-dir DIR` The directories for extra content, like additional files and images specified in the project configuration file. This argument may be called multiple times to specify several lookup directories, content files will be looked up in the content directories in the same order they are added.

`--theme-name NAME` The name of the template to use. Overrides the name specified by the project configuration file.

`--output-dir DIR` Generates the reference under `DIR`.

`--no-namespace-dir` When specified, the files are directly generated under the output directory, instead of using a sub-directory based on the namespace name and version.

`--section NAME` Only generate the section `NAME` of the reference. Valid section names are: `aliases`, `bitfields`, `callbacks`, `classes`, `constants`, `domains`, `enums`, `functions`, `function_macros`, `interfaces`, `structs`, and `unions`. Additionally, `all` will generate all sections, and `none` will generate no section.

6.2 gi-docgen gen-index

6.2.1 Generating the symbols index from introspection data

SYNOPSIS

gi-docgen gen-index [OPTIONS…] [GIRFILE]

DESCRIPTION

The **gen-index** command generates a symbols index from introspection data. The symbols index can be used to efficiently search symbols and terms.

The generated index file is called `index.json`

OPTIONS

`--add-include--path DIR` Adds `DIR` to the list of paths used to find introspection data files included in the given `GIRFILE`. The default search path for GIR files is `$XDG_DATA_DIRS/gir-1.0` and `$XDG_DATA_HOME/gir-1.0`; this option is typically used to include uninstalled GIR files, or non-standard locations.

`-C, --config FILE` Loads a project configuration file.

`--dry-run` Only load the introspection data, without generating the index.

`--content-dir DIR` The directories for extra content, like additional files and images specified in the project configuration file. This argument may be called multiple times to specify several lookup directories, the content files will be looked these directories in the same order they are added.

`--output-dir DIR` Generates the index file under `DIR`.

INDEX FILE

The index file is in JSON format.

The index file contains a single object with the following members:

`meta` = **`object`** An object with metadata about the index.

`symbols` = **`array of objects`** An array of all the addressable symbols.

`terms` = **`object`** A dictionary of all terms.

The `meta` object contains the following members:

`ns` = **`s`** The namespace name.

`version` = **`s`** The namespace version.

`generator` = **`s`** The `gi-docgen` string.

`generator-version` = **`s`** The version of `gi-docgen`.

The `symbols` array contains objects with the following members:

`type` = **`s`** **(*mandatory*)** The type of symbol: `alias`, `bitfield`, `callback`, `class`, `class_method`, `ctor`, `domain`, `enum`, `function`, `function_macro`, `interface`, `method`, `property`, `signal`, `type_func`, `union`, `vfunc`.

`name` = **`s`** **(*mandatory*)** The name of the symbol.

`ctype` = **`s`** The base C type for identifiers; only available for types: `alias`, `bitfield`, `class`, `domain`, `enum`, `interface`, `union`.

`type_name` = **`s`** The type name related to a symbol; only available for types: `class_method`, `ctor`, `method`, `property`, `signal`, `type_func`, `vfunc`.

`ident` = **`s`** The C identifier for symbols; only available for types: `class_method`, `constant`, `ctor`, `function`, `function_macro`, `method`, `type_func`.

`struct_for` = **`s`** The C type related to a class structure; only available for the `class_method` type.

The `terms` dictonary contains all terms as members; each term is associated to an array of indices in the `symbols` array.

6.3 gi-docgen check

6.3.1 Check the documentation in the introspection data

SYNOPSIS

gi-docgen check [OPTIONS…] [GIRFILE]

DESCRIPTION

The **check** command runs a series of checks on the introspection file, to verify that public API is properly documented. It can be used as part of a test suite.

OPTIONS

`--add-include--path DIR` Adds `DIR` to the list of paths used to find introspection data files included in the given `GIRFILE`. The default search path for GIR files is `$XDG_DATA_DIRS/gir-1.0` and `$XDG_DATA_HOME/gir-1.0`; this option is typically used to include uninstalled GIR files, or non-standard locations.

`-C, --config FILE` Loads a project configuration file.

6.4 SYNOPSIS

gi-docgen COMMAND [OPTIONS…]

The `gi-docgen` command line utility has several commands, each with its own functionality and options.

6.5 COMMANDS

gi-docgen generate Generates the API reference

gi-docgen gen-index Generates the symbol indices for search

gi-docgen check Checks the documentation

6.6 OPTIONS

All commands support the following options:

`-q, --quiet` Do not emit any additional information message.

`--fatal-warnings` Make all warnings fatal, immediately terminating the process.

`--help` Show an help message.

6.7 ENVIRONMENT VARIABLES

All commands support the following environment variables:

`GIDOCGEN_DEBUG` If set, `gi-docgen` will emit debugging messages.

6.8 BUGS

Report bugs at https://gitlab.gnome.org/GNOME/gi-docgen/issues

6.9 HOMEPAGE and CONTACT

https://gnome.pages.gitlab.gnome.org/gi-docgen/

6.10 AUTHOR

Emmanuele Bassi

GI-DocGen is a document generator for GObject-based libraries. GObject is the base type system of the GNOME project. GI-Docgen reuses the introspection data generated by GObject-based libraries to generate the API reference of these libraries, as well as other ancillary documentation.

CHAPTER SEVEN

INSTALLATION

7.1 Running GI-DocGen uninstalled

You can run GI-DocGen from its repository, by calling:

```
./gi-docgen.py
```

GI-DocGen will automatically detect this case.

7.2 Installing GI-DocGen via pip

To install GI-DocGen, you will need to have the following pieces of software available on your computer:

- Python 3.6, or later
- pip

Run the following command:

```
pip3 install --user gi-docgen
```

After running the command above, make sure to have the `$HOME/.local/bin` directory listed in your `$PATH` environment variable.

To update GI-DocGen, run the following command:

```
pip3 install --user --upgrade gi-docgen
```

CHAPTER EIGHT

USAGE

First, read *Using GI-DocGen*.

Additional documentation on how to control the generation of your project's API reference is available in the *Project configuration* page.

CHAPTER

NINE

DISCLAIMER

GI-DocGen is **not** a general purpose documentation tool for C libraries.

While GI-DocGen can be used to generate API references for most GObject/C libraries that expose introspection data, its main goal is to generate the reference for GTK and its immediate dependencies. Any and all attempts at making this tool more generic, or to cover more use cases, will be weighted heavily against its primary goal.

If you need a general purpose documentation tool, I strongly recommend:

- HotDoc
- Doxygen
- GTK-Doc

CHAPTER

TEN

COPYRIGHT AND LICENSING TERMS