Feedback from downstreams

Philip Withnall

2015-01-21

1 Introduction

A reasonable amount of the work Collabora does is to help downstream companies to use FOSS software correctly. They have written code using, for example, GLib and GTK+, but need a hand to finish a few tricky bits, train them in use of libraries, or review their code.

Here are two case studies of clients we've helped with their code, highlighting the areas they've had problems with, in the hope that we can improve this upstream.

The first client is a medical device manufacturer, using GTK+ (on Wayland!) in the UI for their proprietary devices. They wanted training and support on getting the most out of GTK+, especially since they wanted to customise the UI.

Client 1: Medical devices

- Training
- GTK+ customisation
- Using: GLib, GTK+ on Wayland, GStreamer, gtkmm

The second client is an automotive supplier, using GLib and Clutter to build a full OS for in-vehicle infotainment (IVI), which is the non-safety-critical side of the screens in a car. They are open-sourcing the entire platform (it will be released soon), and wanted code review and a mediator to handle licencing and upstream issues.

Client 2: In-vehicle infotainment (IVI)

- Code review
- Licencing and open sourcing
- Poking upstreams
- Using: GLib, Clutter, D-Bus, gtk-doc

2 Client 1: Medical devices

2.1 Problems

As Collabora's interaction with client 1 was mostly limited to a training week, we can only give examples of the problems seen that week.

Client 1: Problems

- SVGs in `GtkImage`s don't scale with the widget
- Some properties are confusing, such as `GtkCellRenderText:editable[-set]`
- Really want a way to make `GtkBuilder` behave like a factory
- Gtkmm's documentation, especially Devhelp integration, is poor
- Documentation of widget event masks and event propagation is opaque
- Want a horizontal accordion revealer widget
- No legal or tooling issues

- After creating a `GtkImage` backed by an SVG it seemed strange that the SVG didn't scale as a vector when the widget got resized.
- Properties such as `GtkCellRendererText:editable` and `:editable-set` are confusing.
- They really wanted a way to make `GtkBuilder` files act more like a factory. For example, defining a widget in a UI file then creating n instances of it, as opposed to always getting the same widget out. This is now supported by `GtkBuilder` templates, but the documentation is appalling and gives no hint it can be used for factorying.
- Gtkmm's documentation is poor in comparison to GTK+'s and it works really badly in Devhelp. Searching (an important feature we take for granted when using the C API) is especially bad.
- Documentation of widget event masks was a little odd. You can call `gtk_widget_add_events(w, GDK_KEY_PRESS_EVENT)` but perhaps it's not necessary because the widget is already set up to receive such an event. So is it more about ensuring the event and not depending on implementation details?
- Event propagation is documented well, but not so easily for a beginner. There was a problem with understanding why `GtkTextView` got `::key-press-event` before its parent `GtkWindow`.
- They wanted a new widget for horizontal accordion revealers, instead of packing two `GtkRevealer`s in a box.
- No problems were encountered with licencing or developer tooling.

2.2 Summary

Client 1: Summary

- Various specific problems or misunderstandings of the API
- Deprecations should always be documented with a rationale and replacement API — preferably also a porting guide
- New functionality needs documenting: `GtkBuilder` templates were blogged about extensively, but are given a single sentence in the manual
- New widget: Accordion revealer
- No legal or tooling issues

A lot of client 1's issues are specific bugs to be fixed; the only repeating issue is documentation, especially that of deprecations. There are also a few specific feature requests. Interestingly, there were no legal or tooling issues, though I don't know what their tooling setup is.

3 Client 2: In-vehicle infotainment (IVI)

Client 2 is a little different. Our project with them is much bigger and longer, and has a larger goal of open sourcing an entire platform. This means extensive code and licencing review, and training and bug fixing from the results of the code review. Client 2 have previously only made proprietary products, so this is a big shift for them, and gives some insight into what first-timers to open source think of our platform.

3.1 Legal

Client 2: Legal

- Choosing a licence to release under
- Checking for proprietary code
- Licence hygiene
- Project naming!

The first hurdle is choosing a licence. For us, this is typically a matter of preference; for a company, it involves multiple lawyers, forethought on how they want the project to be used, and discussions about liability, maintenance, ownership and patents. It drags on for a while.

The second is to check that the code being released is not proprietary and has the appropriate copyrights. It is not possible to automate this. Then comes licence hygiene, which is where our choice of licences (as upstream) has an impact. There have been dependencies removed from this project due to their licencing. (Some autoconf macros were incompatible with the parent project's licence.)

Finally, names need to be chosen for all the projects being released. This is not trivial, and involves a fair amount of bikeshedding. Client 2 has tried to avoid offending 'open source co-inhabitants' with their choice of project names.

3.2 Infrastructure

Client 2: Infrastructure

- git
- Bugzilla
- Website and wiki
- Code review system
- Packaging system

The big problems with infrastructure probably come from the interaction between a big company's IT department and Collabora, so are uninteresting. However, there is a lot of infrastructure to set up, and everyone needs to be trained in workflows for using it. We take a lot of this for granted. Packaging would be even more complex if we were targeting more than a single, constrained platform.

3.3 Information

Client 2: Information

- Is the project even interesting?
- Marketing the project
- Producing documentation
- Limitations and known issues

One key question client 2 had was: if we open source this, will anyone be interested? If so, how do we 'market' the project so it gets used and receives contributions? They have had problems producing documentation, eventually settling on gtk-doc and a wiki — but there have been problems getting people up to speed with gtk-doc (especially bootstrapping it).

Client 2 also has the mentality that all limitations and known issues of the software should be documented. This may be an automotive sector thing; it's certainly a change from normal FOSS development.

3.4 Interaction with upstreams

Client 2: Interaction with upstreams

- Insulated by Collabora
- Lacking documentation
- Worry about API deprecations

So far, Collabora has insulated client 2 from the upstream projects it uses, filing bugs and adding features as needed. A lot of work has been done on various upstream libraries as a result. Once client 2's project is released, they will interact more with upstreams. So far, their biggest worry is the lack of documentation for various libraries, particularly examples for APIs or configuration files.

They previously worried a lot about API stability, and what to do in the face of upstream API deprecations. They have now stabilised on a fixed core runtime on top of the base OS, analogous to the runtime concept used in GNOME sandboxing (though without any of the bind mounting). As long as the supported libraries in the runtime are used, API stability is guaranteed. The full sandboxing and runtime story would take this one step further in the right direction for client 2.

3.5 Problems

Collabora has performed a full review of client 2's code base, looking from the overall design down to memory management and code clarity. The issues we've seen can be split up into several sections.

Client 2: Module setup problems

- Project bootstrapping and directory layout best practices
- Bootstrapping problems with gettext, intltool and gtk-doc
- Indicating licencing in files
- Knowing which files to put in git
- Parallel installability of libraries
- Total confusion over LT versioning

A lot of client 2's problems came from their build system. Very few people really understand build systems, so we need to provide more helper macros and build system examples for things. For example, if the `GLIB_GSETTINGS` macro didn't automatically validate the schema files, I doubt anybody would remember to write rules for it.

Knowing which files to install, which to clean, and which to commit to git is a recurring problem. Macros should be designed and documented to make this obvious.

Another set of problems came from the naming and versioning of libraries, which were not parallel installable, and had inconsistent naming. This is definitely something to be solved by Builder or some other project bootstrapping tool.

Client 2: Coding conventions problems

- Namespacing and GObject modularity
- Little confusion over use of GError

- Repeated, inefficient use of GList
- Confusion between sync and async; over-use of threads

Client 2 wrote a number of custom GObjects, and generally got all the boilerplate correct. One recurring problem was putting private data in the public GObject struct, rather than the private one.

`GError` was generally used correctly, which I suspect reflects the quality of its documentation.

Synchronous methods were persistently called instead of the preferred asynchronous versions. This was a repeated theme; when the sync methods caused blocking of the main loop, client 2 started to move them out to custom-spawned `GThread`s, rather than using the async versions.

Client 2: Memory management problems

- Massive over-use of `g_strdup()`
- Persistent confusion over `g_free(NULL)`

Client 2's code had a few leaks, but there were two common problems: calling `g_strdup()` unnecessarily, and checking a variable against `NULL` before calling `g_free()` on it.

Client 2: Tooling problems

- No compiler, GIR or gtk-doc warnings enabled
- No gtk-doc tests enabled
- Confusion over documentation generation with `gdbus-codegen`

A number of client 2's problems would have been caught by compiler, GIR and gtk-doc warnings, but these are not enabled by default. Why? Neither were the gtk-doc unit tests enabled by default (though this has now changed).

A variety of approaches were taken to generating documentation using `gdbus-codegen` and integrating it into a gtk-doc manual. Perhaps we should provide a macro for this.

Client 2: Unit testing problems

- Tests were not automatable
- Tests were not hooked up to CI or other tooling
- APIs were untestable by design

Unit testing is hard, and `GTest` is still not the best API. Universally, the tests written by client 2 required manual interaction, and were not hooked up to continuous integration, code coverage or Valgrind. Some of their APIs were untestable by design (e.g. by relying on global state).

We can't fix design issues, but we could perhaps improve `GTest` to steer the user towards designing better unit tests which are automatically hooked up to various tools plus installed-tests.

Client 2: API-specific problems

- JSON reader function pairing on different code paths
- No validation of resource files (XML, JSON schemas)
- Confusion over GSettings schema internationalisation
- Confusion over relocatable GSettings schemas
- Confusion over GDBus vs. libdbus and libdbus-glib
- Poor D-Bus API design in places
- Problems with D-Bus build system integration
- Problems building SQL and SPARQL queries
- Use of POSIX APIs incompatible with GLib (e.g. `sigaction()`, `system()`, `sleep()`)

There were various API-specific problems. Most of these need fixing through documentation updates, but some of them would better be fixed through static analysis — e.g. pairing of `json_reader_begin_element()` and `json_reader_end_element()`, or detection of calls to `sigaction()` and `sleep()` which are fundamentally incompatible with GLib main loops.

One example is building SQL and SPARQL queries: both SQLite and Tracker provide safe APIs for building queries, but also provide unsafe ones. Universally, the unsafe ones are easier to use (incorrectly), so they are what got used. We must not design APIs like this in future.

Client 2: Threading problems

- Threads were used instead of async calls
- Reinvented thread pools
- Implications on main contexts ignored

As mentioned above, sync calls were used instead of async ones and, when those caused problems, they were moved out to worker threads with some quite dubious `GThread` calls. How can we steer people towards using async calls?

A good example of the complexity people are missing is in some code which queued jobs for a worker thread pool. Some of the code run in the threads called `g_timeout_add()` and ended up shunting the sync work (the computation; not just the results) straight back to the main thread.

Client 2: File system access problems

- Repeated usage of sync APIs
- Problems building file paths
- No validation of file sandboxing

There weren't many examples of file system access in client 2's code, but that which existed tended to build and examine file paths using string manipulation functions (rather than `g_build_filename()` and friends). There was no validation performed on file paths to constrain them to sandboxes — we should think about incorporating sandboxing ideas into all new APIs.

3.6 Summary

Client 2: Summary

- Various bootstrapping and build system integration problems
- Project layout, namespacing, versioning and version control conventions
- Confusion between sync, async and threading
- Various specific problems or misunderstandings of the API
- No tooling warnings enabled — should be on by default
- APIs which could be abused, were abused (SPARQL queries)
- Problems with incompatible POSIX APIs
- Need to think about file sandboxing as part of the API

In summary, client 2 had various bootstrapping and build system problems which could be fixed with bootstrap tools, macros and build system examples. They had repeated confusion over sync and async methods. There were various API-specific problems and misunderstandings which need documenting, and we can assume in API design that if an API can be abused, it will be. Many of their problems could have been eliminated by developing with a stricter set of warnings and errors enabled; but some problems can only really be detected by static analysis, such as interactions between POSIX and GLib.

Miscellany