Project description: Qt Native

Overview

The goal of the project is to add a new module to Qt that offers a way to write true native user interfaces (UI) for both mobile and desktop. This will be done by implementing a thin layer on each platform that wraps the native controls into a QML / C++ API.

Background

Historically we have tried to avoid wrapping native controls in Qt. Instead we write our own controls from scratch, and where needed, style them to resemble native look-and-feel. This has many advantages, one of them being that we define the APIs ourselves, and as such, can offer true cross-platform ways of writing UIs.

This approach was easier before, mainly because of two reasons. The first being that most platforms offered a styling API we could use for drawing native controls. This has changed, and today many of the same platforms (e.g macOS, iOS, Android, Windows) don't have that. On macOS, for example, we still use the old carbon styling APIs that was left unmaintained by Apple after they switched to 64-bit architectures in 2007. On Android and iOS, such APIs have never even existed.

The other important change is that application UIs have become more dynamic, especially on mobile, with extensive use of transitions and animations. Without a proper styling API, those are hard to fake. And the platforms varies quite much, not only when it comes to styling, but also layout structure. Android apps will e.g typically make use of a side navigation controller (drawer) as the main navigation system, while iOS favors a tab bar.

So while we could (and to a certain degree do) mimic platform styles, the result never ends up with a quality that is acceptable. And trying to fake a style can also have legal issues, and potentially cause apps written with Qt to be rejected on app stores. And at the moment, we have no good existing solution on how to solve all this.

Solution

We believe that Qt needs to offer a way to write native looking applications. If we want Qt to stay ahead of the competition, and be the SDK developers choose for cross-platform development also in the future, we need a way to solve this. And the solution we propose is to research and implement how we can offer a Qt API backed by native controls.

What value will Qt Native add to Qt

The most imperative benefit of having this product will be that we end up with an offering we don't have, the ability to write 100% native looking applications with Qt, especially on mobile. This will also include controls that we are currently prohibited from using, like the ribbon UI on Windows. We understand that we currently don't make much revenue on mobile, but we'll probably make even less if we don't have a complete product to sell. More long term we see the possibility of Qt falling behind competitors like Xamarin and React Native. They already provide cross-platform mobile UIs by wrapping native controls, and the former have also similar offerings for macOS and Windows. Many predict that mobile and desktop will merge at one point, and we think it's important that we try to get a market share on mobile going forward, even if it doesn't currently generate substantial revenue. Otherwise we'll take the risk of becoming more and more irrelevant on those platforms as our competitors keeps getting stronger, and perhaps eventually lose out on desktop (and even embedded) in the future, once they start investing into those areas too. We believe that having a product like Qt Native will make Qt more complete, gain more mindshare, and as such, easier to sell, also also for markets not directly involved in mobile or native UIs.

Where does Qt Native belong in Qt

If you want to create applications with a true native look and feel, and are willing to write code that is likely to contain platform specific sections, then Qt Native will be your best option. Otherwise, if you plan to brand / style your application, or prefer to write one UI for all platforms, or target platforms like embedded, then Qt Quick Controls is more flexible, and will be your best option. They will also be possible to combine, so that you can have a Qt Quick Controls view inside a Qt Native application.

Initial cost

A rough estimate is that each platform we choose to support will take one man-year to implement for a tech-preview to be reached. A lot of work has been done already for Android and iOS, so they might complete faster. iOS and macOS is also very similar API-wise, so a lot of code/pattern reuse should be possible between them.

Maintenance cost

A distinction between Qt Native and e.g Qt Quick Controls is that in the latter case, we control the API. We decide how it should be, and which features should be implemented next. With Qt Native, this will shift towards the maintainers being more concerned about what new APIs are available on a platform, and if it makes sense to wrap those as well. As such, the module will never be "done". Instead, each platform will need to be maintained regularly, and especially when new minor/major releases of underlying SDKs are released. Practically, we expect at least

one developer per platform to spend around 20% of his working time per year. The exact percentage and cost will of course depend on our ambitions for the module.

Scope

As a first version, we could consider only doing an implementation for Android and iOS. Windows phone has little market share, and can be skipped for now. Desktop platforms can also be done in the next round, to not steal too much manpower from other projects.

Milestone plan

Target for this module will be Qt-6.

Current status

Currently we have a git repository with a module that contains source code for generating both QML and c++ libraries for iOS, Android, and macOS, together with examples and a few autotests. The examples shows how you can use the libraries to create windows, table views, tab bars, buttons and other native controls using either QML or c++. While the quality level of each platform varies, we are at a point where one can write some simple examples and get a feel of how things will look like in the end.

We have spent some time experimenting on how to add a common API on top of the different wrapper libraries. But we're still unsure about how it can best be realized. Should the common API be QML only, or should it include c++? Should it be separate libraries with independent controls, or should we instead align the platform APIs? Does it make sense to have a common API at all? So the current implementation, and some of the examples, are still in a state that reveals this process.

Further implementation

We will focus on implementing a different API per platform that resemble the API of the underlying native SDK. We want Qt Native to offer all the bells and whistles available on a platform, and not be restricted to a common cross-platform API. This means that you will need to write different variations of your UI for each platform that you target. That said, it makes sense to try to settle on some common vocabulary for the most standard controls. E.g a button should be called Button, and have a property 'text' on all platforms, and not 'label' or 'title' etc. And we will use Qt "primitive" types whenever possible, like QString, QRectF, QColor etc, rather than trying to wrap the native equivalent for those as well. This should make it quite easy for a developer to move from one platform to next during development, and some code sharing should be possible.

A key feature that needs to be available, is the possibility to add a QQuickView as a child into the scene. That way, you can use Qt Native to write the frame and structure of the app (toolbars, tabbars, title bars, drawers, dialogs, etc), but use QtQuick in the center of the screen with your cross-platform “business logic”.

While we focus on a QML API, the implementation will be done in c++. We plan to make the libs public so that you can write apps using c++ if you want to, but more important, be able to extend the controls, or wrap your own native controls, in an easy way.

Challenges

A lot of the implementation time will likely be spent on translating imperative functional APIs into declarative property-based QML APIs. From prototyping with iOS and Android, this has so far turned out to be tricky. But now that we got experience on how to do this, the same patterns should be applicable on other platforms as well.

While it might sound appealing to write generators that can auto-generate wrappers based on e.g native interface / header files, I would at this point not recommend going that route. It has been hard enough to implement the prototypes manually. Trying to capture that into generators that understands how to e.g convert native multi-argument functions into good QML APIs based on properties and bindings will be really hard, and take much more time to implement.