Using Qt 6 under LGPLv3

by Burkhard Stubert / 2023/01/06 / 21 Comments

Feature	LGPL-3.0	GPL	Professional	Enterprise
Essential modules	Yes	Yes	Yes	Yes
Free add-on modules	Yes	Yes	Yes	Yes
Paid add-on modules	No	Yes	Yes	Yes
Run on automotive-grade SoCs	Yes	Yes	No	Yes
Qt for MCU	No	No	Yes	Yes
Qt Safe Renderer	No	No	No	Yes
Qt Application Manager	No	Yes	No	Yes
Ready-made Linux images	No	No	Yes	Yes

The Qt Company changed Qt licensing in February 2022. All the separate commercial packages – including Qt for MCU, Qt Safe Renderer, Qt Automotive Suite and Qt Automation – were folded into two Qt for Device Creation licenses: Professional and Enterprise. The Qt Marketplace license for modules like CoAP, MQTT, Charts and for the design tool bridges was discontinued. My post helps you answer the crucial question: Shall you use Qt Commercial or Qt LGPL-3.0?

The Three Qt License Offerings

The Qt Company offers Qt for embedded devices under three licenses: Qt for Device Creation, LGPL-3.0 and GPL. Qt for Device Creation is the commercial offering, whereas LGPL-3.0 and

GPL are the free and open-source offerings. I often use Qt Commercial as a short hand for Qt for Device Creation. GPL means GPL-2.0 or GPL-3.0.

The Qt Company updated their commercial offering the last time in February 2022. They introduced it in the post The new simplified Qt commercial licensing: what's in it for you? on their company blog. My analysis of the new licensing from my February 2022 newsletter holds up very well. It has been confirmed in many conversations with executives, managers and developers from companies using Qt or planning to use Qt.

The Qt for Device Creation license comes in two editions: the Professional and the Enterprise edition. Here is the feature comparison table for LGPL-3.0, GPL, Professional and Enterprise.

Feature	LGPL-3.0	GPL	Professional	Enterprise
Essential modules	Yes	Yes	Yes	Yes
Free add-on modules	Yes	Yes	Yes	Yes
Paid add-on modules	No	Yes	Yes	Yes
Run on automotive-grade SoCs	Yes	Yes	No	Yes
Qt for MCU	No	No	Yes	Yes
Qt Safe Renderer	No	No	No	Yes
Qt Application Manager	No	Yes	No	Yes
Ready-made Linux images	No	No	Yes	Yes
Qt Creator	No	Yes	Yes	Yes
Qt Design Studio Professional	No	No	Yes	Yes
Technical Standard Support	No	No	Yes	Yes

Table 1: Feature comparison between LGPL-3.0, GPL, Professional and Enterprise editions

Qt for MCU is the offering for running Qt on microcontrollers (MCUs). It provides the code to bring-up Qt on bare-metal systems or on RTOSs. It also provides the Ultralite version of Qt Quick. The firmware with Qt can fit into 6MB of RAM and flash. Before the license change in February 2022, Qt for MCU was a separately licensed product with costs in the upper 5-digit USD range per customer project. Now, it is included in both Qt Professional and Qt Enterprise.

Before the license change in February 2022, Qt Safe Renderer was a separately licensed product with costs in the upper 5-digit USD range per customer project. Now, it is included in Qt Enterprise but not in Qt Professional. Qt Safe Renderer allows us to show some indicators (icons or text) inside an otherwise unsafe GUI application. Even if the GUI application crashes, the indicators are guaranteed to be shown on the screen. This guarantee is possible, because the safe renderer is certified to SIL 3 and ASIL D for road vehicles, SIL 4 for railway vehicles and "fit-for-use" in medical devices.

The Qt Application Manager is a window and application manager licensed under Qt Enterprise or GPL. The window manager is based on the Qt Wayland Compositor. It integrates Qt Virtual Keyboard as a window running in its own process. Every application can use the keyboard for input. The Qt Application Manager can run any applications like Qt, HTML and OpenGL applications and display them, as long as they support the client-side Wayland protocol.

The application manager controls the lifetime of the applications. If the system is low on memory, it can terminate some applications. It can quick-launch QML applications by loading the QML into a running QML engine and install signed third-party applications. The applications can communicate with each other over D-BUS – not over Qt Remote Objects.

Qt for Device Creation provides ready-made Linux images for many widely used SoCs, system-on-modules (SoMs) and SBCs made by NXP, Nvidia, Intel, Texas Instruments, Raspberry Pi Foundation, ST, Toradex, Boundary Devices, Kontron and many others. The installation of a commercial Boot2Qt image sets up QtCreator for cross-compilation, for deploying your Qt application to the device, for running it on the device and for remote debugging. So, we can immediately start developing our Qt application.

Qt Creator is an excellent IDE for developing embedded software using QML, Qt, C++, C, CMake and Python. When we press `Ctrl+R` (Run), Qt Creator cross-builds our software for the embedded device, installs it on the device and runs it. This gives us early and frequent feedback how our software behaves on the device. If an issue arises, we can do remote debugging or run static analysis tools on the device.

I have been using Qt Creator since 2006 for my daily development. In my TDD legacy workshops, I sometimes use Visual Studio Code or CLion IDE. They are better than Qt Creator in some respects (e.g., automated refactorings) and worse in others (e.g., navigation of QML code, unit testing, CMake support). I never felt the need to say good-bye to Qt Creator, my trusted coding companion. Choosing an IDE is a very personal decision. Developers should choose the IDE with which they feel most comfortable and productive.

Although Qt Creator is only available under GPL besides Qt Commercial, we can use it for developing commercial software with Qt LGPL-3.0. The code we write in Qt Creator is not under GPL but under any license we choose. It's similar to the g++ compiler. The compiler is under GPL, but its output is not.

Qt Design Studio Professional is a tool for designing 2D and 3D user interfaces. It is only available with the commercial licenses. Importing designs from Adobe XD, Figma, Sketch, Maya, Modo, Blender and other 3rd party tools is not available in Qt Design Studio Professional. These bridges are only available in Qt Design Studio Enterprise (see this comparison chart), which is neither included in Qt for Device Creation Professional nor Enterprise.

The new Qt licensing from February 2022 introduced some more changes:

- Qt for Automation, which included the modules Qt MQTT and Qt CoAP, was folded into Qt Professional and hence into Qt Enterprise.
- The additional features of Qt Automotive Suite are now part of Qt Enterprise.
- The bridges from design tools like Adobe XD, Figma and Sketch to Qt Design Studio were available through the now discontinued Qt Marketplace License. They are not included in any Qt for Device Creation offering.

If you need help with answering the question "Shall you use Qt Commercial or Qt LGPL-3.0?" and with checking the license compliance of an embedded Linux system, have a look at my product page

Qt for Device Creation

The Qt for Device Creation license comes in two editions: the Professional and the Enterprise edition. The main differentiator is, whether our embedded software runs on automotive-grade SoCs (systems on chip) or not. A SoC is automotive-grade, if it works fine in the temperature range from -40°C to +125°C. All machines and devices used outdoors like cars, trucks, agricultural machines, construction machines, ships, e-bikes, sports watches and GPS handhelds fall in this category. Their manufacturers must buy the license Qt for Device Creation Enterprise.

Industrial-grade SoCs work fine in temperature range from 0°C to +70°C and consumer-grade SoCs in the range from 0°C to +40°C. Embedded software running on industrial-grade or consumer-grade SoCs is covered by the license Qt for Device Creation Professional. Example devices are medical devices, TVs, STBs, home and professional appliances, phones and all industrial machinery located indoors.

Guessing the Qt Licensing Costs

For both licenses, we must pay both per-developer fees and per-device fees. The license fees depend on the number of developers and the number of devices. There are discounts, if we pay the license fees for 3 or 5 years in advance and if we buy licenses for more devices. I would assume that the per-developer and per-unit fees for Qt Enterprise are higher than the fees of Qt Professional.

Warning: I don't know the prices. The prices are my best guesses based on what I hear from the Qt community. You should talk to Qt Sales to learn the exact prices. The prices from Qt Sales should be the basis for the cost comparison between Qt Commercial and Qt LGPL-3.0. However, I believe that a price indication – if not the actual price – is an important part of treating customers fairly. That's why I am trying to give you an idea of the Qt license fees.

The Qt Company doesn't publish any pricing information for the Qt for Device Creation licenses. However, they give the per-developer fees for the Qt for Application Development license (Qt for Desktop). The Professional edition clocks in at 3,620 USD per year and per developer, and the Enterprise edition at 4,090 USD.

Qt for Device Creation Professional includes everything from Qt for Application Development Enterprise. It's fair to assume that the per-developer fee for both Qt for Device Creation licenses is higher than 4,090 USD, when paid on a yearly basis. 4,500 USD seems like a good guess.

The Qt Company gives big discounts, if we pay for the licenses 3 or 5 years in advance and if buy for a bigger number of developers. If we got a 20% discount for 3 years and a 35% discount for 5 years, we would end up with yearly per-developers fees of 3,600 USD and 2,900 USD, respectively.

In addition to the per-developer fee, the Qt for Device Creation licenses also come with a per-device fee (a.k.a. royalties). We must buy distribution licenses in minimum batches of 2,000 units. We pay for the distribution licenses in advance for the complete licensing period (3-5 years). What I hear from the Qt community, the per-unit fee could be roughly 8 USD. Again, The Qt Company gives generous discounts, if we order distribution licenses in higher volumes.

	Year 1-5	Year 6-10	Year 11-15	Year 1-15
#Developers	3	4	5	
Developer fees (USD)	45,000	60,000	75,000	180,000
#Devices	4,000	6,000	8,000	
Device fees (USD)	32,000	48,000	64,000	144,000
Total (USD)	77,000	108,000	139,000	**324,000**

Table 2: Typical Qt license costs over 15 years

The table shows the typical costs for a company building driver terminals for construction machines. The lifetime of the terminals is 15 years. Per year, the company pays 3,000 USD per developer and 8 USD per device.

- For the first 5 years, 3 developers build 800 devices per year.
- For the second 5 years, 4 developers build 1,200 devices per year.
- For the third 5 years, 5 developers build 1,600 devices per year.

The license fees are always due at the beginning of a 5-year licensing period. Upfront costs around 100,000 USD for each 5-year period and total costs of more than 300,000 USD over 15 years is a considerable investment for such a company. These companies want good value for their investment. They want shorter development times and earlier time to market. Providing more than anecdotal evidence for that is hard. And it's even harder, because Qt Commercial competes with an excellent free product: Qt LGPL-3.0.

Combining Commercial and Free Qt Licenses

The Qt License Agreement forbids combining commercial and free open-source Qt licenses during development and on devices.

> "Prohibited Combination" shall mean any effort to use, combine, incorporate, link or integrate Licensed Software **with any software created with or incorporating Open Source Qt**, or use Licensed Software for creation of any such software.
>
> Qt License Agreement (version 4.4.1), "1. Definitions" (emphasis mine)

The meaning of “use”, “combine”, “incorporate”, “link” and “integrate” is not defined any further. It’s vague. So, we must tread very carefully. Let met spell out **what we must not do, if we hold a commercial Qt license**.

- Software using Qt Commercial must not link against libraries using Qt LGPL-3.0 or being written with non-commercial Qt tools (e.g., Qt Creator). This includes the following cases:
  - We must not use any libraries that a completely independent third party developed using Qt LGPL-3.0 or Qt GPL. Any embedded Linux system contains a couple of these libraries.
  - We must not use any software written with Qt Creator under GPL – even if this software doesn’t use Qt at all. It is impossible to check whether any third-party developer uses Qt Creator to develop software for embedded Linux systems.
- An application using Qt Commercial must not communicate with any another application using Qt LGPL-3.0 or Qt GPL, if both applications run on the same device. This is more restrictive than GPL.
- An application using Qt Commercial must not start another application using Qt LGPL-3.0 or Qt GPL, if both applications run on the same device. This is more restrictive than GPL.
- We must not use software developed by a contractor or affiliate with Qt tools or libraries under GPL or LGPL-3.0 (see clause 3.4.ix).
- We must not use any Qt tools like Qt Creator, Qt Design Studio, QML compiler extensions and QML Live to build software using Qt LGPL-3.0 or Qt GPL. It doesn’t matter whether these tools are also available under a non-commercial Qt license.

> “Permitted Software” shall mean (i) **third party open source software products** that are generally available for public in source code form and free of any charge under any of the licenses approved by Open Source Initiative as listed on https://opensource.org/licenses, **which may include parts of Open Source Qt or be developed using Open Source Qt**; and (ii) software The Qt Company has made available via its Qt Marketplace online distribution channel.
>
> **Qt License Agreement** (version 4.4.1), “1. Definitions” (emphasis mine)

The definition of “Permitted Software” above contradicts the definition of “Prohibited Combination”. It would, indeed, rule out the fairly absurd consequences of the first bullet point above. But – which interpretation should we apply?

If we take the license obligations seriously and we better do, we should at least renegotiate the License Agreement or even walk away from it. **Some obligations are impossible to satisfy, some are contradictory and the sensible ones are harder to satisfy than those of LGPL-3.0.**

I'd urge The Qt Company to rewrite their License Agreement along the following objectives:

- The license holder must pay the license fee for every developer, who writes Qt code used in software running on an embedded device.
- The license holder must pay the license fee for every produced embedded device with Qt code written by its developers.

If the license holder cheats with the numbers of developers or devices, The Qt Company has the right to suspend or cancel the License Agreement. This implies that the customers of the license holder must not use the embedded device any more. This is a temporary or permanent sales stop – and definitely not a situation companies want to be in.

Qt LGPL-3.0

Clause *4. Combined Works* of the LGPL-3.0 lays down our obligations. Here is my translation from legalese (see my talk Using Qt under LGPLv3 for more details).

We may **release our combined work** – our software combined with the Qt libraries under LGPL-3.0 – **under a license of our choice**, if we satisfy the following obligations.

- *(4a) We give prominent notice (e.g., in the user manual) that our software uses Qt under LGPL-3.0.*
- *(4b) We provide copies of the LGPL-3.0 and GPL-3.0 to the buyers of the software, where buyers are businesses or consumers (private persons).*
- *(4c) We display the copyright notices of Qt and the license texts of LGPL-3.0 and GPL-3.0 prominently in the user interface.*
- *(4d) We provide the mechanism of either (4d0) or (4d1) to link our software with a potentially modified but interface-compatible version of the Qt libraries.*
    - *(4d0) We use any mechanism for linking – including static linking.*
    - *(4d1) We use dynamic linking to combine our software with the shared Qt libraries.*

- *(4e) We provide installation information (according to clause "6. Conveying Non-Source Forms" of the GPL-3.0) how to build a modified version of Qt, how to install Qt on the device, and how to run the software with Qt on the device. This clause **only applies to Consumer products**.*

Let me spell out the ramifications of the above clause in plain English.

- We can keep the source code of our software secret (see the first sentence).
- We can link our software with shared or static Qt libraries (see (4d0) and (4d1)). If we use static linking, we must provide the object files of our software. We don't have to do that for dynamic linking.
- Clause (4e) – a.k.a. the anti-tivoisation clause – only applies to Consumer products like phones, TVs, STBs, game consoles, home appliances and cars. When consumers use their right and install modified Qt libraries on their devices, we can void the warranty. We could even forbid consumers to drive their cars.
  Conversely, we do not have to provide any way to install modified Qt libraries on B2B products like most medical devices, all professional appliances and all industrial, construction and agricultural machines. B2B products are those that we sell to businesses only.

Using Qt under LGPL-3.0 is a perfectly viable option for businesses. In the last three years alone, more than 60 companies inquired about using Qt LGPL-3.0, 16 bought a license assessment and 8 bought a full-blown license compliance check of their embedded Linux systems from me. Through my consulting work, I know of many more companies using Qt LGPL-3.0 including very well-known manufacturers of agricultural, construction and industrial machines, cars and home appliances.

Qt GPL-2.0 and GPL-3.0

In constrast to LGPL-3.0, GPL requires us to **release our combined work** – our software combined with the Qt libraries or any source code under GPL – **under the GPL**. This implies that we must make the source of our software publicly available. Businesses usually don't want that, as they would lose their competitive advantage. Publishing the source code is typically OK or even required from universities and government agencies.

Even businesses frequently use software under GPL: as tools or applications running in their own process. For example, Git and BusyBox are GPL-2.0-only, GCC is GPL-3.0-or-later with GCC Runtime Library Exception, Qt Creator is GPL-2.0-or-later. Proprietary software may call tools

under GPL or communicate with other applications or services under GPL – and we can keep the source code of the proprietary software secret.

This gives us a way to use Qt modules like Qt Wayland Compositor, Qt Application Manager and Qt Virtual Keyboard that are under Qt Commercial and GPL but not under LGPL-3.0. We run these modules as separate applications in their own processes. Our proprietary software communicate with them over Qt Remote Objects, D-Bus or gRPC. The Wayland compositor shows client applications under GPL (e.g., the virtual keyboard) or any other license with the right size at the right position.

Of course, we must publish the source code of all the applications under GPL (e.g., Wayland compositor, virtual keyboard, application manager) including our modifications. This should not be a problem, as these modifications are hardly ever a competitive advantage. We can keep the the source code of our proprietary applications and libraries secret, as they do not link against any libraries under GPL and do not integrate any source code under GPL.

Some of the essential and add-on Qt modules are under GPL-2.0-only and some under GPL-3.0-only. As GPL-2.0 and GPL-3.0 are not compatible, software must not mix these modules. For example, software using both Spatial Audio (GPL-3.0-only) and TextToSpeech (GPL-2.0-only) violates GPL-3.0. We must not release such software.

Licensing of Qt Modules

Essential Qt Modules

Qt Module	LGPL	GPL	Commercial
Core	3	2, 3	Pro, Enterprise
D-Bus	3	2, 3	Pro, Enterprise
GUI	3	2, 3	Pro, Enterprise
Network	3	2, 3	Pro, Enterprise
QML	3	2, 3	Pro, Enterprise
Quick	3	2, 3	Pro, Enterprise

Qt Module	LGPL	GPL	Commercial
Quick Controls	3	2, 3	Pro, Enterprise
Quick Dialogs	3	2, 3	Pro, Enterprise
Quick Layouts	3	2, 3	Pro, Enterprise
Quick Test	3	2, 3	Pro, Enterprise
Test	3	2, 3	Pro, Enterprise
Widgets	3	2, 3	Pro, Enterprise

Table 3: Licensing of modules in Qt Essentials

All essential Qt modules are available under LGPLv3 and additionally under GPLv2, GPLv3, Qt for Device Creation Professional (Pro) and Qt for Device Creation Enterprise (Enterprise). The essential Qt modules are guaranteed to be binary and source compliant throughout Qt 6 and to run on all platforms supported by Qt.

Add-On Qt Modules under LGPLv3

Qt Module	LGPL	GPL	Commercial
3D	3	2, 3	Pro, Enterprise
Qt5 Core Compatiblity	3	2, 3	Pro, Enterprise
Bluetooth	3	2, 3	Pro, Enterprise
Concurrent	3	2, 3	Pro, Enterprise
Help	3	2, 3	Pro, Enterprise
Image Formats	3	2, 3	Pro, Enterprise
Multimedia	3	2, 3	Pro, Enterprise

Qt Module	LGPL	GPL	Commercial
NFC	3	2, 3	Pro, Enterprise
OPC UA	3	2, 3	Pro, Enterprise
OpenGL	3	2, 3	Pro, Enterprise
PDF	3	2, 3	Pro, Enterprise
Positioning	3	2, 3	Pro, Enterprise
Print Support	3	2, 3	Pro, Enterprise
Quick Widgets	3	2, 3	Pro, Enterprise
Remote Objects	3	2, 3	Pro, Enterprise
SCXML	3	2, 3	Pro, Enterprise
Serial Bus	3	2, 3	Pro, Enterprise
Serial Port	3	2, 3	Pro, Enterprise
Shader Tools	3	2, 3	Pro, Enterprise
Spatial Audio	3	3	Pro, Enterprise
SQL	3	2, 3	Pro, Enterprise
State Machine	3	2, 3	Pro, Enterprise
SVG	3	2, 3	Pro, Enterprise
TextToSpeech	3	2	Pro, Enterprise
UI Tools	3	2, 3	Pro, Enterprise
Wayland Client	3	2, 3	Pro, Enterprise
WebChannel	3	2, 3	Pro, Enterprise

Qt Module	LGPL	GPL	Commercial
WebEngine	3	2, 3	Pro, Enterprise
WebView	3	2, 3	Pro, Enterprise
XML	3	2, 3	Pro, Enterprise

Table 4: LGPLv3 Modules in Qt Add-Ons

All the above add-on modules are available under LGPLv3 and additionally under GPLv2 (except Spatial Audio), GPLv3 (except TextToSpeech), Qt for Device Creation Professional (Pro) and Qt for Device Creation Enterprise (Enterprise). Add-on modules are not guaranteed to run on all platforms. Binary and source compatibility differs from module to module.

The licensing of the following modules needs special attention.

- Since Qt 6.4, **Qt Multimedia** comes with two backends: gstreamer and ffmpeg. The section Licenses and Attributions states correctly: *"Some optional components of FFmpeg are only available under GPL. The FFmpeg backend shipped with the Qt binary packages is configured to not contain any of the components that are available under GPLv2 only."*
  I have come across several Yocto BSPs that build the GPL components of ffmpeg by default. I don't see how Qt Multimedia could ensure that ffmpeg doesn't load any GPL plugins at runtime. As we want to be on the safe side with licenses, I'd make sure that the BSP build doesn't use the option `--enable-gpl`.
  If the ffmpeg backend used a plugin, a library or just a single source file under GPL, our proprietary application linking against Qt Multimedia would be under GPL. Hence, we would have to publish our proprietary source code – no matter whether we hold a commercial Qt license or not.
- **Qt OPC UA** provides two implementations in the form of plugins. The first plugin uses the free and open-source Open62541 library under the Mozilla Public License v2.0. The second plugin uses the commercial UA CPP library. Hence, we can use the Qt OPC UA with the Open62541 plugin under LGPLv3 but not Qt OPC UA with the UA CPP plugin. The latter combination with UA CPP is only possible under a commercial Qt license.
- The module **Qt Shader Tools** is classified as a commercial Qt add-on on the page All Modules. A look at the source code and at the licensing page reveals that the runtime library is under LGPLv3 and that the command-line tool `qsb` is under GPL. Hence, our proprietary software can link against the runtime library under the LGPLv3. Calling a tool under GPL like

`qsb` during the build is not a problem. It's like calling `g++` during the build. Therefore, I re-classified the module as an LGPLv3 add-on module.

- **Qt Wayland Client** comprises the libraries that Qt applications need to communicate over the Wayland protocol with a Wayland compositor (e.g., Weston, Qt Wayland Compositor). Qt Wayland Client is available under LGPLv3, whereas Qt Wayland Compositor is not. Qt Wayland compositor is only available under a commercial Qt license or GPLv3.
- **Qt WebEngine** uses more than 200 other components with many different licenses (see QtWebEngine Licensing). In isolation, these licenses – including BSD, MIT, MPL-2.0, Apache-2.0 and LGPL-2.1 – are unproblematic. The combination of some licenses may be prohibitive. We must not combine a component under Apache 2.0 with a component under LGPL-2.1-only. Apache-2.0 components may only be combined with components under LGPL-2.1-or-later or LGPL-3.0.
  Qt WebEngine uses 60 components under Apache-2.0 and 13 components under LGPL-2.1. 2 of the 13 components under LGPL-2.1 – `hunspell` and `hyphenation-patterns` – contain source files that are under LGPL-2.1-only (`phonet`.cxx and `phonet.hxx` in `hunspell` and `hyph-hy.hyb` in `hyphenation-patterns`). Qt WebEngine violates LGPL-2.1-only and must not be deployed with the violating components.

To be very clear: Releasing Qt WebEngine with both the Apache 2.0 components and the two components under LGPL-2.1-only (hunspell and hyphenation-patterns) is a **license violation.** We must not release Qt WebEngine in this configuration to customers. It does not matter whether we use Qt WebEngine under Qt Commercial, Qt LGPLv3 or Qt GPL.

I see two options under our control to remedy this situation.

- If we don't need Qt WebEngine, we don't install it on the embedded device.
- If we need Qt WebEngine, we build it without the components hunspell and hyphenation-patterns.

My request to The Qt Company is to ask the developers of the `hunspell` and `hyphenation-patterns` components to change LGPL-2.1 into LGPL-2.1-or-later everywhere. I think that this is the intention of the component developers anyway. This change should be possible for the Qt 6.5 release at the end of March 2023.

We only looked for LGPL-2.1-only components in Qt WebEngine. We must also look for such components in other Qt modules. If, for example, Qt Multimedia depends on an LGPLv2.1-only library and our application uses Qt Multimedia and Qt WebEngine, we have a license violation, too.

The discussions of Qt Multimedia, Qt Wayland Client and Qt WebEngine should show one thing. **Even if we buy a commercial Qt license, we are not safe from license violations.**

The source code of the following modules may be a bit hard to find.

- **Qt PDF** is part of Qt WebEngine. Its source code is in the directory `qtwebengine/src/pdf` of the Qt repository.
- The source code of **Qt TextToSpeech** can be found in the directory `qtspeech` of the Qt repository.

Commercial Add-On Qt Modules

Qt Module	LGPL	GPL	Commercial
Charts	No	3	Pro, Enterprise
CoAP	No	3	Pro, Enterprise
Data Visualization 3D	No	3	Pro, Enterprise
Http Server	No	3	Pro, Enterprise
Lottie Animation	No	3	Pro, Enterprise
MQTT	No	3	Pro, Enterprise
Network Authorization	No	3	Pro, Enterprise
Quick 3D	No	3	Pro, Enterprise
Quick 3D Physics	No	3	Pro, Enterprise
Quick Timeline	No	3	Pro, Enterprise
Virtual Keyboard	No	3	Pro, Enterprise
Wayland Compositor	No	2, 3	Pro, Enterprise

Table 5: Commercial Modules in Qt Add-Ons

From Qt 5.15 to Qt 6.2, The Qt Company offered the modules Charts, CoAP, MQTT, OPC UA with UA CPP plugin, KNX and PDF under the Qt Marketplace License for yearly prices between 50 and 200 USD. This offering was discontinued silently some time after the release of Qt 6.2. I couldn’t find any announcement.

The modules Charts, CoAP, MQTT and OPC UA with UA CPP plugin are now only available under the commercial Qt for Device Creation licenses. The PDF module is now available under LGPLv3. The KNX module was discontinued.

The Crucial Question: Qt LGPLv3 or Qt Commercial?

Early in the project, we must ask the crucial question: *“Shall we use Qt LGPL-3.0 or Qt Commercial?”* The answer depends on the specific project.

1. We ask Qt Sales for a quote and create a table for the license costs over the lifetime of our product similar to Table 2.
2. We go through the feature comparison table and the table of the commercial add-on modules, possibly add some features, and identify all features that are available under Qt Commercial but not under LGPL-3.0.
3. We go through the list of Step 2, enumerate some alternatives for each item in the list, estimate the costs for the different alternatives, and choose the best alternative.
4. We sum up the costs for the alternatives needed for using Qt LGPL-3.0.
5. If the costs for Qt LGPL-3.0 (Step 4) are lower than the costs for Qt Commercial (Step 1), we use Qt LGPL-3.0. Otherwise, we use Qt Commercial.

This is an objective way to reach a decision. The initial question *“Shall we use Qt LGPL-3.0 or Qt Commercial?”* boils down to *“How can we save development efforts worth the cost of Qt Commercial (e.g., 324,000 USD)?”* or *“How can we earn the costs for Qt Commercial (e.g., 324,000 USD) by launching our product earlier?”*

Business analysts, product managers, system architects and experienced developers discuss the alternatives and their costs in Step 3. Here are some typical considerations from my experience.

Qt for MCU being part of Qt Professional and Qt Enterprise seems like a big advantage over Qt LGPL-3.0. I have my doubts.

- MCUs powerful enough to run Qt GUIs smoothly are more expensive than, say, an i.MX6ULL with a Cortex-A7 application processor and Linux. It’s a lot easier to find developers for an embedded Linux system running on a microprocessor than for a bare-metal or RTOS system running on a microcontroller.

- As soon as the firmware of a microcontroller controls a machine, the control code is most likely safety-critical. Then, it's advisable to separate the safety-critical code from the non-safety-critical code, that is, the GUI code. The separation reduces the certification efforts significantly. You get this separation naturally when you use hybrid SoCs like the NXP Vybrid, i.MX7 or i.MX8. The GUI runs on the microprocessor and the control code on the microcontroller.
- Many GUI frameworks have a much smaller memory footprint than Qt and are better suited for microcontrollers.
- Many companies developing embedded systems with Qt have no idea how to use the Cortex-M microcontroller on their iMX8 SoCs. In other words, they don't need the microcontroller.

Currently, *Qt Safe Renderer* does not support any interaction. If we need, say, a safety-critical button, we realise it in hardware. Similarly, we could realise indicator lamps in hardware. Manufacturers of agricultural and construction machines are used to realise safety-critical buttons and indicators in hardware and can buy them off the shelf. So can manufacturers of medical devices. As with Qt for MCU, I am skeptical whether Qt Safe Renderer compels purchasers to buy Qt Enterprise.

Qt Application Manager is a pretty good window and application manager. Although it is licensed under GPL and Qt Enterprise only, we can still use it with client-side applications using Qt under LGPL-3.0. As applications running in different processes and communicating over inter-process communication never constitute a combination in the sense of the GPL or LGPL, we are on the safe side (see Qt GPL-2.0 and Qt GPL-3.0 for more details).

The window and application managers of most embedded systems are fairly simple. The Qt Application Manager would often be too much of a good thing. We could write a simple window manager in QML based on Qt Wayland Compositor, add some basic application life-cycle management and some inter-application communication with Qt Remote Objects. Nothing fancy. Again, the window manager is licensed under GPL-3.0, whereas the applications are under a proprietary license of our choice and use Qt LGPL-3.0. We need to figure out which approach suits our needs better.

Qt for Device Creation provides ready-made Linux images for many widely used SoCs, SoMs and SBCs. The Linux images contain all the Qt modules, some Qt example applications (auto-starting at boot time) and a Wayland-based window manager. If we work with one of the hardware platforms supported by Qt Commercial, we can save quite a bit of time – especially if we have never built a Linux system with Yocto. We can start from a working Linux image. Qt Creator is set up for building, deploying, running and debugging for the target device right out of the box.

Pretty soon, however, we must tailor the Linux images to our needs. We must remove unwanted packages, change and create Yocto recipes, and build the images ourselves. That's when we must go through the steep learning curve of Yocto – no matter which Qt license we use. Fortunately, the image and SDK builds run smoothly most of the time – especially when we follow descriptions like Setting Up Yocto Projects with kas, Building a Linux Image with Yocto and Building a Qt SDK with Yocto.

Need Help with License Compliance?

If you need help with license compliance, have a look at my product:

Product Description: License Compliance for Embedded Linux Systems

In the Q&A section of the product description, you find a list of my free resources about Qt Licensing.

Tags: LGPL LGPLV3 QT QT6