Google Summer of Code

Port digiKam to Qt6 on Linux

MENTORS

Gilles Caulier <caulier.gilles@gmail.com>
Maik Qualmann <metzpinguin@gmail.com>

INTRODUCTION

Name	Anjani Kumar
Email	anjanik012@gmail.com
KDE Invent	https://invent.kde.org/anjani
Github	https://github.com/anjanik012
LinkedIn	https://www.linkedin.com/in/anjanik012
Telegram	https://t.me/anjanik012
Matrix	@anjanik012:kde.org
Website	https://anjani.live
Country of Residence	India (+5:30 UTC)
College	B.I.T. Sindri, Dhanbad

ABSTRACT

digiKam 7.2 as of now, is fully compatible with Qt 5.15.2 on all major platforms. Qt6 has been released recently. Given that Qt5 and Qt6 are 8 years apart, the way of doing things has changed and complies with the C++17 standard. This project is aimed at preparing the digiKam project to be ported and released with Qt6 on Linux. Implementations that rely on deprecated, removed, improved APIs will be ported with a lot of regression testing.

The AppImage is known to be broken. Internationalization, or proper ICU support doesn't work with AppImage. ICU works fine on other sources for digiKam like flatpak and native binaries. This issue will be fixed by porting the AppImage builder scripts to use Qt 5.15.2 with KDE Patches collection (Not for the moment) which was recently announced (https://dot.kde.org/2021/04/06/announcing-kdes-qt-5-patch-collection). New scripts will be implemented to build the AppImage with Qt6 in future.

This project focuses on the Linux platform. For macOS, macports (https://www.macports.org/) don't support Qt6, nor the MXE project (https://mxe.cc/) that is used to build digiKam for Windows.

GOALS

These are the goals of this project:

- **Port to Qt6**
  - Fix all the Qt5Core Compatibility uses in the project. (https://doc-snapshots.qt.io/qt6-dev/qtcore5compat-module.html) . By fixing, it means to remove their usages completely and replace them with the new Qt6 API wherever possible.
  - Do regression testing with the current test suite, port them to Qt6 and write new unit tests if required.
  - Replace Boost Random code with *QRandomGenerator* with Qt6 in DImg Framework.

- **New AppImage Builder**
  - Fix ICU issue support with AppImage by porting it to use Qt 5.15.2.
  - Create new build scripts to create AppImage with Qt6.

IMPLEMENTATION DETAILS

Fix all Qt5 deprecated compiler warnings

The first step would be to fix all the deprecated compiler warnings in the current Qt 5.15 code before using any Qt6 API.
This is the full build log containing all warnings.
https://gist.github.com/anjanik012/66f80b246c38d65c029cca8030cd5772

I've compiled them class-wise to get a more clear view:

Class	Complete List
QStringList (A total of 30 files)	https://gist.github.com/anjanik012/a701333c390ab13f8a447f8bca8f946d
QTextStream (A lot of files)	https://gist.github.com/anjanik012/c3285eb6e40546fbbeb5a3db5c511ace
QButtonGroup (2 files)	https://gist.github.com/anjanik012/45060fc25cc81992826dbdcdb302a3f0
QDateTime (3 files)	https://gist.github.com/anjanik012/84352bd8d9f8ed84323d3b7ce6b75518
QFlags (3 files)	https://gist.github.com/anjanik012/794e65a0d7c4cefadfd38c131796ebaf
QHash (2 files)	https://gist.github.com/anjanik012/8af8c1d0f2aa392f37371ae02e1ca226
QImage (6 files)	https://gist.github.com/anjanik012/8e6987c3efd5d7c11586f86d4c5e6c04
QMap (3 files)	https://gist.github.com/anjanik012/78be83e172ee99c92e597aa0f856e217
QPixmap (2 files)	https://gist.github.com/anjanik012/4304bac86a3911698ab725f707c31197
QWheelEvent (5 files)	https://gist.github.com/anjanik012/f033a6cdd52a9ddc379472322cce7d40
QPrinter (2 files)	https://gist.github.com/anjanik012/fb4149c2a51115897b4b99e8a52e4cbe
QProcess (1 file)	https://gist.github.com/anjanik012/3bbf02d0519f91204c6df100a9712e37
qrand() (18 files)	https://gist.github.com/anjanik012/13d5978d4733ea2b627f414c68a1e2af
qsrand() (10 files)	https://gist.github.com/anjanik012/59513471c928a3efdf5d782ff0665b38
QSet (7 files)	https://gist.github.com/anjanik012/af2f63b403bd9c2fc581731b5b9186f0
QTabletEvent (1 file)	https://gist.github.com/anjanik012/e8

	24c7eab9c30ac42fb5aa8ad96204ca
QTimeLine (1 file)	https://gist.github.com/anjanik012/9a00d471fdd77a664c6a3ded73060d03

This is a crucial step. After this, the main porting work will begin by introducing Qt6 to digiKam
https://doc-snapshots.qt.io/qt6-dev/qtcore5compat-module.html.
This module is given for easing the porting process and moving away from it is recommended as quickly as possible.

Qt5CoreCompat module usages

These are the usages of deprecated classes that contain direct references to Qt5 CoreCompat module of Qt6. I have included Github gists containing the list of all usages in the files that need to be changed.

Class	**Solution**	**Complete list**
QRegExp (A Total of 81 files)	Use QRegularExpressions from Qt6::Core	https://gist.github.com/anjanik012/668d26b4fef310329747f6503bced1ee
QStringRef(A total of 12 files)	Use QStringView from Qt6::Core	https://gist.github.com/anjanik012/2f9b5f4b0201a284d86c228405b20302
QTextCodec(A total of 8 files)	Use QTextCodec from Qt6::Core5Compat. Unfortunately, there is no alternative to it other than using the compat module. https://doc-snapshots.qt.io/qt6-dev/internationalization.html	https://gist.github.com/anjanik012/4bc44a6958532940253ea3e9b0083667
QXmlAttributes, QXmlInputSource, QXmlDefaultHandler, QXmlSimpleReader	This implementation to read and parse track files is not used and also require some changes and	

In class *TrackReader* from core/*utilities/geolocation/geoiface/tracks/trackreader.h*	improvements apart from being ported to Qt6. The solution is to use the QXmlStreamReader class from Qt6::Core to ease up the parsing.	

Components that rely on removed Qt5 classes

DImg framework uses Qt X11 Extras

The ICC settings class (*IccSettings*) declared in *dimg/filters/icc/iccsettings.h* uses *QX11Info* class which is removed from Qt6. It's purpose is to retrieve the system X11 ICC monitor profile. The function *IccSettings::monitorProfile()* returns the default sRGB profile if it can't find a system profile.

One possible way to port this is to extract the function definitions out from the *QX11Info* class and use them directly in our codebase.
https://code.qt.io/cgit/qt/qtx11extras.git/tree/src/x11extras/qx11info_x11.cpp
Upon seeing this source, it seems that *QX11Info* is just a wrapper/helper class and its functionality can be used directly using Qt6::Core GUI classes.
For example, *QX11Info::isPlatformX11()* has this body

QGuiApplication::platformName() == QLatin1String("xcb");

Other used functions from this class are also easily accessible directly from GUI classes.

On wayland, unfortunately, the color management protocol at the time of writing this document, is unstable and is work-in-progress https://gitlab.freedesktop.org/wayland/wayland-protocols/-/merge_requests/14 . This can be implemented when the protocol becomes stable in future.

Since this project only focuses on the Linux port, windows and macOS work is left out for now. One possible way to make this work on Windows is to try this https://stackoverflow.com/a/64427505/5859944. I'll try this on Windows but it is not a part of the project.

Rajce webservice plugin uses QXmlPattterns

Well, this plugin is already optional in CMake and currently broken. The plugin uses *QXmlPatterns* class which is removed from Qt6. If we were to change the

implementation compatible to Qt6 we would need to use *QXmlStreamReader* to parse any XML data. However, the rajce API will be changed in 2021. They have announced that they will be releasing a new REST API which is good news. The new implementation can get rid of XML parsing altogether in favor of JSON. The announcement can be read at https://www.rajce.idnes.cz/api.
I'll put this plugin in quarantine for now as, reimplementing the plugin for the same API for some time, only to become redundant when a new API is published will be not ideal. Porting this will also require a good amount of time which can be difficult to do in GSoC along with other more important tasks.

Removing libO2 dependency

The current libO2 implementation is based on the *Qt Network* module and the OAuth implementations use *QNetworkManager* class. The *QNetworkManager* class has gone under some changes in Qt6 but hasn't been deprecated. However, there is a better alternative, the *QNetworkAuthentication* module from Qt6 that has OAuth 1 & 2 implementations.

While the current implementation of web services plugin will continue to function in Qt6 (except rajce), libO2 becomes a dependency which can be dropped with a simpler implementation based on *QNetworkAuthentication* module. This task also requires a good time to complete. Since it will work with Qt6, it has a low priority.

Replace Boost Random Code with Qt6 in DImg Framework

DImg framework uses boost::random module to generate high-quality random numbers. However, the *QRandomGenerator* is also suitable. Dropping boost dependency from this use case makes sense. We'll make use of the Qt6 API here. The class implementing the generators is defined in *core/libs/dimg/filters/randomnumbergenerator.cpp.* All the boost random usages will be replaced by *QRandomGenerator* class. The implementation will become simpler.

There is another boost dependency in DImg, Boost Graph which is used in version history. Getting rid of Boost completely will be desirable but there is no alternative to Boost Graph in STL or Qt. If we wanted to drop Boost completely, we would have to implement our own graph data structures and tests which is too much for this project.

Using clazy for code analysis

clazy tool is great for the last phase of porting. Most importantly, it'll help to remove subtle Qt6 changes such as warnings of deprecated operators, partially changed classes, usage of deprecated members of some classes, etc. This last step is crucial for ensuring code quality and will help in fixing regression test failures. The idea is to fix all of these warnings module-by-module and keep running regressions tests until we find no more. The following are the checks that will be used with clazy:

- qt6-deprecated-api-fixes
- qt6-header-fixes
- qt6-qhash-signature
- qt6-qlatin1stringchar-to-u
- qt6-fwd-fixes
- missing-qobject-macro

All the "fixes" checks introduce changes in the code directly. This is not a good thing to do. So, I'll use these "fixes" checkers manually on specific parts of digiKam one-by-one and apply the suggestions manually.

General regression testing strategy

With so many changes being made to components, regression testing is quite important to ensure nothing breaks and if something does, we can fix it. DigiKam has a great testing suite. The plan is to partially select components that have undergone changes one-by-one and run the corresponding tests. This process will be repeated for a few number of iterations till a satisfactory phase is achieved. The tests can simultaneously be ported to Qt6 if required.

Porting AppImage builder to use Qt 5.15.2 and later Qt6 with ICU support

The current appimage contains Qt 5.14.2 and doesn't work with ICU. These are the related bugs:

- https://bugs.kde.org/show_bug.cgi?id=410980
- https://bugs.kde.org/show_bug.cgi?id=418670
- https://bugs.kde.org/show_bug.cgi?id=425168

The Qt used in the appimage is not compiled with *icu* because of the locale problems with the appimage. Also, the Mageia 6 is not capable of building Qt 5.15.2.
To enable ICU for the appimage, upgrading the Qt version compiled with *icu* support seems to be the solution. However, to maintain binary compatibility with most Linux distributions, the libc dependency has to be sufficiently old. The current appimage is built on Mageia 6 https://www.mageia.org/en/. I tried upgrading to Mageia 7.1 to build the appimage, but the OpenSSL packaged with Mageia 7.1 is 1.1.0 whereas Qt 5.15 requires 1.1.1.
Also due to a bug with the configure in Qt 5.15, we cannot pass our custom compiled OpenSSL 1.1.1 to the Qt configure, which makes Mageia 7.1 unusable for building the appimage.
Thus, the only solution is to use *Mageia 8* which has both Qt 5.15.2 and OpenSSL 1.1.1 in the official repositories. This simplifies the build process of the appimage as we no longer would have to build Qt ourselves. This Qt is also built with ICU support which should be enough to fix all the listed bugs above.

Also, Qt6 uses cmake for the build system which makes Mageia 8 a good environment to build the appimage.

The implementation flow would be something like:

- Simplifying build hosts scripts to remove external compilation of tools and dependencies like cmake, openssl, libicu, Qt 5.15.2 as the newest versions are provided by the distribution.
- Write a new build host script targeted to build Qt6 and its dependencies, if required.
- Port appimage builder script to work with new Qt6 builder script.

Finally, the appimage would need to be tested properly in different environments to ensure that the broken ICU is fixed.

TIMELINE

This year's GSoC is 10 weeks long and the student is required to work an average of 18 hours a week. I'm prepared to work for 20 hours a week on average for the whole period. I'll be in constant contact with the mentors and update them regularly with my progress. Also, there are only 2 evaluations this time. This is how I plan to work for the program.

I have an internship that is expected to start in June and will go on for 8 weeks. I don't think this would affect my work in GSoC. Due to the pandemic there is

an overwhelming chance that the internship will be work-from-home. In that case, I'll have flexible working hours and can easily meet the deadlines of GSoC according to the timeline I've described below. If it happens to be offline, this is how my workflow will be for the whole period.

I'll work 3hrs per day from Monday to Friday. That sums to 15 hrs. Since, I've time to spare on weekends, I expect to work for 5 hours in the period. That gets me 20 hours a week on average. If I fall short of deadlines, I'm prepared to give more time on working days. I can assure you of no miscommunications with the assigned mentors due to my internship.

The main goals of the project are porting to Qt6 and building well functioning AppImages. I'm dividing the work in small doable chunks and assigning them with a priority. This is needed as tasks labelled with higher priority will need to be done first to port digiKam correctly. These are the tasks with priority high to low with approximate times required for completion:

Sl. No	**Task**	**Reason**
1.	Fixing all Qt5 deprecated compiler warnings (2 weeks)	This is the top priority task as these are deprecated in Qt5 and must be fixed before using Qt6 in the project. This involves working with a lot of files and thus, will require some time to be completed.
2.	Removing Qt5 modules that are removed in Qt6 (2-3 days)	Again, this needs to be done before introducing Qt6 as these modules are not present in Qt6. This is a short task that would not need much time.
3.	Introducing Qt6 to digiKam by fixing Qt5CoreCompat module usages (2 weeks)	This is the first point of time where Qt6 will be introduced. This task is a long task as it involves a lot of files and patches.
4.	Replace boost random uses with Qt6 in DImg framework (2-3 days)	Boost random can be replaced with Qt6. Though not as crucial as it can be left and it will work but since an option is available in Qt6, hence this is desirable. This is a short task.

5.	Regression testing and conducting clazy checks (1-2 weeks)	This is the point where we test the changed code for regressions and fixing them. Clazy checks would need to be done to fix other Qt6 compatibility issues. This is important and a medium length task.
6.	AppImage Builder (at least 2 weeks)	This is an important task as AppImage is a widely used bundle. It also has bugs that are explained in the implementation section. This is a long task that would need a lot of testing in different environments.
Pending tasks for later		
7.	Replacing libO2 with new OAuth implementation (at least 2 weeks)	Our Qt6 port will work with current implementation and can be ensured with some regression tests. The Qt modules used in this are also not deprecated, so it is expected to work with some housekeeping. That's why, this task receives a low priority. In addition, this task is time-taking and would require a lot of improvements. Completing this task along with other high priority tasks is a challenge given the time constraints of GSoC.
8.	Fixing Rajce plugin (2 weeks)	This plugin is broken and depends on the web service component of digiKam. The plugin needs a rewrite as we wait for the new API to arrive. This is a long task. However, isolating the plugin from the build procedure is a small task.

Community Bonding Period (17 May - 7 June)

This period is important for introducing my project goals to the community. The preparation for the project can be done in this period. I'll:

- Write at least two introductory blog posts on planet KDE.
- Discuss the implementation in detail with the mentors.
- Configure the coding environment like setting up Qt6, tools, Mageia 8, etc. before the coding period starts.

One important task in this period is to plan a regression testing strategy with the mentors before the coding period starts. I've put a general strategy above but more planning might be needed.

Coding Period
Deliverables for 1st evaluation (7 June - 12 July) - 5 weeks

Task: Fix Compiler warnings of deprecated Qt5 usages (2 weeks)

- Fix all warnings as described in implementation section
- Test for regressions

Task: Drop Qt X11 Extras dependency and Replace Boost Random in DImg framework (1 week)

- Get the required functions and definitions from the Qt5 source tree
- Port *iccsettings.cpp*
- Port random number generator in DImg framework to QRandomGenerator from Qt6
- Isolate Rajce plugin
- Write inline documentation where needed
- Refactor code

Task: Fix Qt5CoreCompat usages(2 weeks)

- Port to Qt6
- Write new XML parser for TrackReader
- Write documentation
- Regression testing / clazy checks
- Prepare for evaluation
- Write Blog post

First Evaluation (July 12 - July 16)

- Fix bugs
- Apply suggestions made by the mentors
- Submit code

Deliverables for final evaluation (July 16 - August 16) - 5 weeks

Task: More regression testing and Clazy checks (1-2 weeks)

- Fix regressions

Task: Port AppImage Builder (2-3 weeks)

- Port old scripts
- Write new scripts for Qt6
- Fix any errors/bugs happening
- Test the appimage on different platforms
- Test Qt ICU support

Task: Work on pending later tasks in final weeks if all higher priority tasks are complete

- Drop libO2 and implement new OAuth framework
- Test web services
- Port rajce plugin if the new REST API arrives

Final Evaluation (August 16 - August 23)

- Final refactors/formatting if required
- Write a complete blog post.
- Submit Final Work

Final Evaluations (August 16 - August 23)

Post GSoC

I certainly wish to stay in the community after GSoC. I'll try to complete the pending tasks if they are left and will keep contributing to the project,keep polishing my work and see digiKam be released with Qt6.

INVOLVEMENT IN KDE

I've been using KDE products for a long time and have always desired to contribute and finally, I did it in the GSoC 2020 season. I applied to KDE Connect. Though my application wasn't selected (because I wasn't ready), I kept contributing since then to improve, and have been given the developer status. These are some of the projects in which I have contributed:

- KDEConnect Android(Fixed broken clipboard plugin and worked on UI)
- KWeather (Small patch in UI)
- KWeatherCore (Add CAP protocol support)
- Neochat (Small patch in UX)

Here is my KDE invent account https://invent.kde.org/anjani.

The KDE community is very welcoming and helpful. The products made by KDE are some of the best in their respective domains. Working with the best people in industry has already taught me a lot. Though I've applied to one more organization, CERN-HSF this year, my first preference is this project. I've already worked for an year and I wish to do more and someday mentor a KDE project myself. I hope that I am selected for this project and help make KDE better than ever. Looking forward to a great summer!

MORE ABOUT ME

I'm a junior in college, currently pursuing B.Tech in Information Technology from India. I've always had a keen interest in Linux and FOSS since I was in high school. I'm eager to be a part of the FOSS community and through this, I'm happy to contribute in fixing small issues or reporting bugs that I find on Github.
I first learnt C++ in high school which was about 4 years ago. From the last 2 year in college, I made a couple of projects with C++ and CMake to get proficient in it. One is a small 8puzzle solver https://github.com/anjanik012/8puzzleSolver. Another tool that I created was, in my opinion, rather cool and intriguing. It authenticates super-user requests on the Linux desktop with an Android phone (Both devices must be in a local network). I used Boost ASIO library to implement networking code https://github.com/anjanik012/suto. This was inspired by the KDE Connect project on which I've also worked on. I'm working on the adding CAP (Common alerting protocol) support to the KWeatherCore library. This was my Season of KDE 2021 project. I'm quite comfortable using Qt.
In addition to these, I like to teach and guide freshers in the FOSS community in my university campus. Currently, I'm serving as the Secretary of the Hackathon and Coding Club at my university and I've spent the past couple years teaching as well as enhancing my own skills in the field. I'm also a member of LEO Club that works on the social fronts, and have participated enthusiastically for Blood Donation Camps and Book Donation Drives for underprivileged children living in the societies near my university.