

Why and how to use KDE frameworks in non-KDE apps

Javier O. Cordero Pérez





Who is this for?

- The person who wants to make a Linux app
- The professional in the lookout for cool libraries



Linux Dev

*Canonicus Linux programmator
futurum*

Professional

Peritus fanaticus cōdex elit

By the end of this talk

- You'll have learned about frameworks the KDE Community has to offer
- You'll be able to use KDE frameworks in your own projects
- You'll have guidance on how to distribute apps that make use of KDE frameworks



\$ whoami

- Javier O. Cordero Pérez (Cuperino)
- Software Engineer at KDAB
- Degrees in:
 - Computer Science &
 - Mass Media Communications
- Author of QPrompt Teleprompter app

KDE Frameworks

- A collection of add-ons libraries for programming with Qt
- Made primarily to satisfy the needs of the KDE Community



<https://develop.kde.org/products/frameworks>



**Let's see KDE Frameworks
in action!**



Qt Creator

- Needs no introduction
- Uses **KSyntaxHighlighting** for source code highlighting

A screenshot of the Qt Creator IDE. The main window displays a C++ source file named 'documenthandler.cpp'. The code is highlighted with colors: comments are in grey, strings in red, and code in black. The code includes a search function that uses Qt's text editing classes like QTextCursor and QTextDocument. The IDE interface includes a menu bar (File, Edit, View, Build, Debug, Analyze, Tools, Window, Help), a toolbar, and a sidebar with icons for Explorer, Run and Debug, and Help. The status bar at the bottom shows '1. Type to locate (CTRL)', '2. Search R...', '3. Application O...', '4. Compile O...', '5. Te...', '6. Version Co...', '7. Test R...', '8. QML Debugger Co...', '9. General Mes...'.

<https://www.qt.io/download-qt-installer-oss>



Subsurface

- Diverlog app
- Widgets UI for desktop
- **Kirigami** on iOS and Android
- Goal: Share common C++ backend



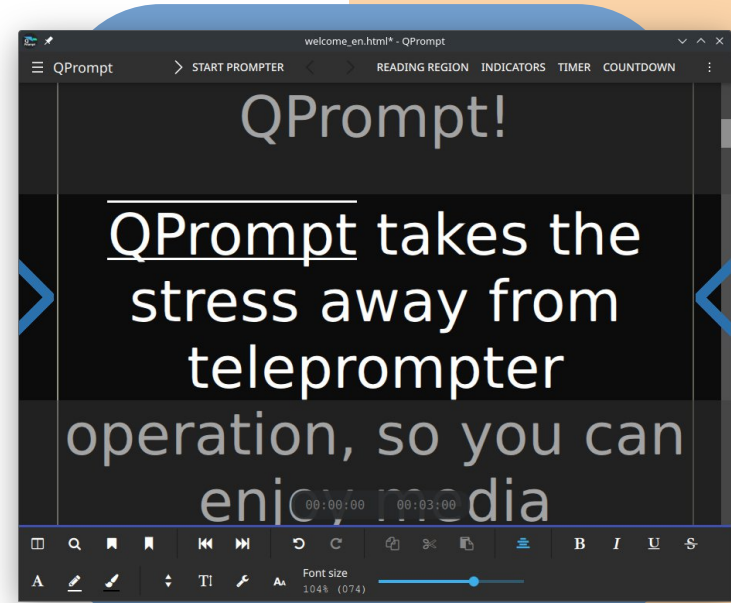
Related talk: *Desktop to Mobile - Developing for Multiple Platforms Without Losing Your Mind*

<https://youtu.be/pa-npvZcm6o?si=OCRUrvit2PZE2NLB>



QPrompt

- Teleprompter app
- Used by professionals and independent video creators alike
- Made using KCoreAddons, Ki18n, **Kirigami, KIconThemes, KCrash, BreezeIcons**
- Goal: Make a native Linux app



<https://qprompt.app>



KomicsReader

- Available on GitHub
- Made with KCoreAddons, Ki18n, **Kirigami**
- It displays comic book formats



<https://github.com/tubbadu/KomicsReader>

Astra

- FFXIV launcher for Linux by Joshua Goins
- Supports profiles, multiple accounts, and plugins
- Made using KCoreAddons, Ki18n, **Kirigami**, **KIconThemes**, **KConfig**, **KArchive**
- Goal: Use and expand your skills



<https://xiv.zone/software/astra>

KDE Frameworks

Divided into tiers:

- Tier 1 – Depend only on Qt, and, sometimes, a small number of third-party libraries)
 - Tier 2 – Additionally depend on tier 1 frameworks
 - Tier 3 – Have more complex dependencies
 - Tier 4 – *“Can be safely ignored by application programmers”*
(I’m quoting the website)

<https://api.kde.org/frameworks/>



KDE Frameworks

“If you ask me...”

- Tier 0
 - + Extra CMake Modules
 - + A few tiny KDE libraries outside the frameworks list
- Tier 1 – Depend on ECM and sometimes 3rdparty libs
- Tier 2 – Additionally depend on tier 1 frameworks
- Tier 3 – Have more complex dependencies

<https://api.kde.org/frameworks/>



KDE Frameworks



<https://api.kde.org/frameworks/>

List of the libraries

Tier 1

Tier 1 frameworks depend only on Qt (and possibly a small number of other third-party libraries), so can easily be used by any Qt-based project.

Table of Content

- [Tier 1](#)
- [Tier 2](#)
- [Tier 3](#)
- [Tier 4](#)
- [Porting Aids](#)

Filters

 Filter by platform

About

Providing everything from simple utility classes to integrated solutions for common requirements of desktop applications


Maintainer

[The KDE Community](#)

Supported platforms

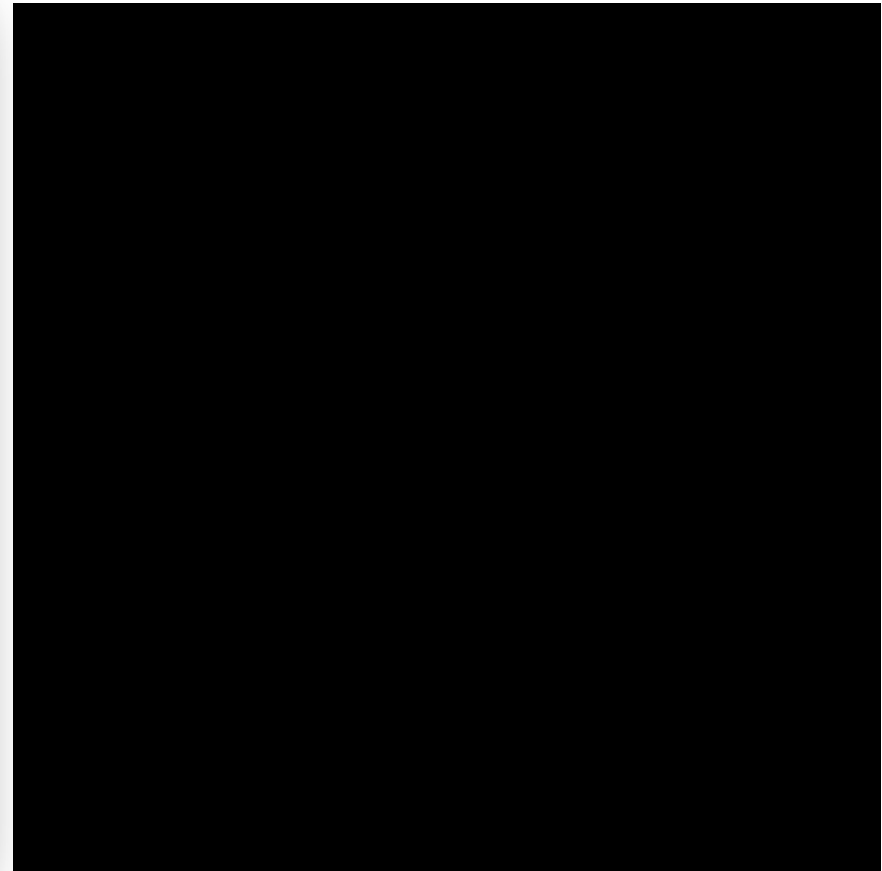
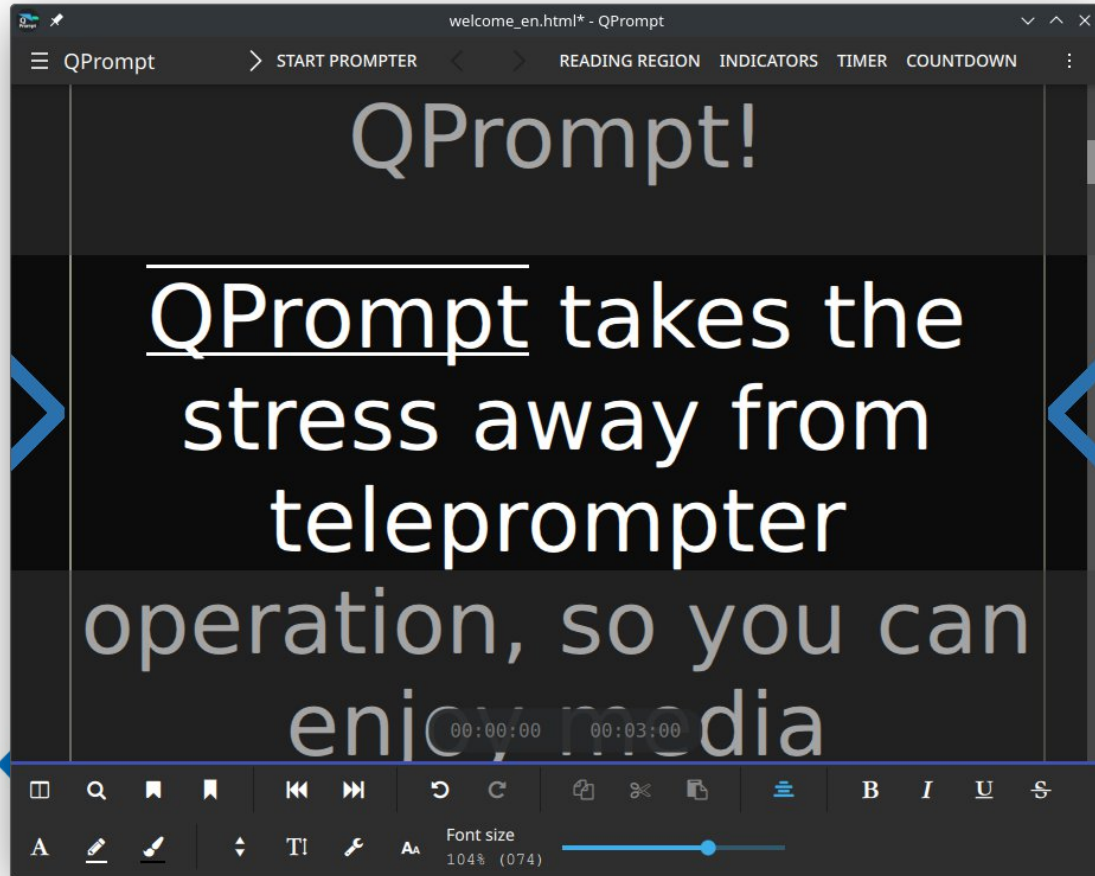
Android (partial), FreeBSD (partial), Linux, macOS (partial), Windows (partial)

Community

	Framework	Maintainer
Attica Open Collaboration Services API	Android FreeBSD Linux Windows iOS macOS	The KDE Community
BluezQt Qt wrapper for BlueZ 5 DBus API	Android FreeBSD Linux  Windows iOS macOS	The KDE Community
BreezeIcons Breeze icon theme	Android FreeBSD Linux Windows iOS macOS	The KDE Community
ECM Extra CMake modules	Android FreeBSD Linux Windows iOS macOS	The KDE Community
KApiDox Scripts and data for building API documentation (dox) in a standard format and style	Android FreeBSD Linux Windows iOS macOS	The KDE Community
KArchive File compression	Android FreeBSD Linux Windows iOS macOS	The KDE Community
KCalendarCore The KDE calendar access library	Android FreeBSD Linux Windows iOS macOS	The KDE Community
KCodecs Text encoding	Android FreeBSD Linux Windows iOS macOS	The KDE Community

KDE Frameworks

Tier 1



<https://x.com/JCuperino/status/1482379644716425222>

The Qt, OpenGL and C++ experts

KDE Frameworks

Tier 1

Sonnet

Multi-language spell checker

Introduction

Sonnet is a plugin-based spell checking library for Qt-based applications. It supports several different plugins, including HSpell, Enchant, ASpell and HUNSPELL.

It also supports automated language detection, based on a combination of different algorithms.

The simplest way to use **Sonnet** in your application is to use the SpellCheckDecorator class on your QTextEdit.

Example

```
#include <QTextEdit>
#include <spellcheckdecorator.h>

MyFoo::MyFoo(QWidget *parent) : QWidget(parent)
{
    QTextEdit *textEdit = new QTextEdit(this);
    Sonnet::SpellCheckDecorator *decorator = new Sonnet::SpellCheckDecorator(textEdit);
}
```

KDE Frameworks

Tier 2

KNotifications

KNotifications is a cross-platform library for creating popup notifications.

It currently supports Linux (and other Unix platforms that implement freedesktop.org notifications), Windows (8 or later), macOS and Android (version 5.0 or later).

Please consult the [KDE Human Interface Guidelines](#) for when using Notifications is appropriate.

[KNotification](#) is the main entry point for using KNotifications.

The global config file

In order to perform a notification, you need to create a description file, which contains default parameters of the notification. It needs to be installed to `knotifications6/appname.notifyrc` in a `QStandardPaths::GenericDataLocation` directory. On Android, this path is `qrc:/knotifications6/`.

The filename must either match `QCoreApplication::applicationName` or be specified as the component name to the [KNotification](#) object.

KDE Frameworks

Tier 1

Solid

Desktop hardware abstraction

Introduction

Solid is a device integration framework. It provides a way of querying and interacting with hardware independently of the underlying operating system.

It provides the following features for application developers:

- Hardware Discovery
- Power Management
- Network Management

Usage

If you are using CMake, you need to have

```
find_package(KF6Solid NO_MODULE)
```

(or similar) in your CMakeLists.txt file, and you need to link to KF6::Solid.

See the documentation for the **Solid** namespace, and the [tutorials on TechBase](#).

KDE Frameworks

Tier 1

KUserFeedback

Framework for collecting feedback from application users via telemetry and targeted surveys.

Telemetry

- Extensible set of data sources for telemetry.
- Full control for the user on what data to contribute.

Surveys

- Distribute surveys and offer users to participate in them.
- Survey targeting based on telemetry data.
- Allow the user to configure how often they want to participate in surveys.

Components

This framework consists of the following components:

- Libraries for use in applications.
- QML bindings for the above.
- A server application.
- A management and analytics application.

KDE Gear

KDE Applications

KDE is a community of friendly people who create over 200 apps which run on any Linux desktop, and often other platforms too. Here is the complete list.



Okular
Document Viewer



Dolphin
File Manager



Kdenlive
Video Editor



Konsole
Terminal



Ark
Archiving Tool



Plasma System
Monitor
System Monitor



Kate
Advanced Text Editor



Gwenview
Image Viewer



Spectacle
Screenshot Capture
Utility



KDE Connect
Device Synchronization



Discover
Software Center



Krita
Digital Painting

KDE Frameworks

- Pros
 - Expand Qt's functionality
 - *Cross platform
 - Well documented
 - Open governance
- Cons
 - Tight integration with KDE's infrastructure can be inconvenient for non-KDE apps

KDE Frameworks

- Pros

- Expand Qt's functionality
- *Cross platform
- Well documented
- Open governance

- Cons

- Tight integration with KDE's infrastructure can be inconvenient for non-KDE apps

*iOS support is very limited

*WASM support is presently non-existent



<https://api.kde.org/frameworks>

KDE Frameworks

- Pros
 - Expand Qt's functionality
 - *Cross platform
 - Well documented
 - Open governance
 - **Free software & open source**
- Cons
 - Tight integration with KDE's infrastructure can be inconvenient for non-KDE apps
 - **Strong copyleft licenses require consideration**

Respecting the licenses

- KDE Frameworks are licensed under **LGPL**, BSD, or MIT licenses
- KDE Apps use GPL licenses



https://community.kde.org/Policies/Licensing_Policy

- 00 The freedom to run the program as you wish, for any purpose
- 01 The freedom to study how the program works, and change it [..]
- 10 The freedom to redistribute copies *so you can help others* [..]
- 11 The freedom to distribute copies of your modified versions to others [..] *give the whole community a chance to benefit from your changes.*



The four essential freedoms

<https://www.gnu.org/philosophy/free-sw.en.html>

GPL family licenses not only give freedoms, they also help defend them

They do this by requiring that you distribute the source code for changes in derivative works

Under the LGPL, non-GPL family programs can be distributed under any terms if they're not derivative works

For a program not to be considered derivative, certain requirements must be met

Also, commercial use is allowed

4. Combined Works (LGPL-3)



You may convey a Combined Work under terms of your choice that, taken together, effectively do not restrict modification of the portions of the Library contained in the Combined Work and reverse engineering for debugging such modifications, if you also do each of the following:

- a) Give prominent notice with each copy of the Combined Work that the Library is used in it and that the Library and its use are covered by this License.
- b) Accompany the Combined Work with a copy of the GNU GPL and this license document.
- c) For a Combined Work that displays copyright notices during execution, include the copyright notice for the Library among these notices, as well as a reference directing the user to the copies of the GNU GPL and this license document.
- d) Do one of the following:
 - 0) Convey the Minimal Corresponding Source under the terms of this License, and the Corresponding Application Code in a form suitable for, and under terms that permit, the user to recombine or relink the Application with a modified version of the Linked Version to produce a modified Combined Work, in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.
 - 1) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (a) uses at run time a copy of the Library already present on the user's computer system, and (b) will operate properly with a modified version of the Library that is interface-compatible with the Linked Version.
- e) Provide Installation Information, but only if you would otherwise be required to provide such information under section 6 of the GNU GPL, and only to the extent that such information is necessary to install and execute a modified version of the Combined Work produced by recombining or relinking the Application with a modified version of the Linked Version. (If you use option 4d0, the Installation Information must accompany the Minimal Corresponding Source and Corresponding Application Code. If you use option 4d1, you must provide the Installation Information in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.)

4. Combined Works (LGPL-3

You may convey a Combined Work under terms of your choice that, taken together, effectively do not restrict modification of the portions of the Library contained in the Combined Work and reverse engineering for debugging such modifications, if you also do each of the following:

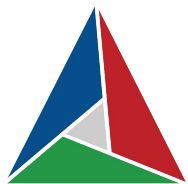
- a) **Give prominent notice** with each copy of the Combined Work **that the Library is used** in it and that the Library and its use are covered by this License.
- b) **Accompany** the Combined Work **with a copy of the GNU GPL and this license** document.
- c) For a Combined Work that **displays copyright notices during execution**, include the copyright notice **for the Library** among these notices, as well as a reference directing the user to the copies of the GNU GPL and this license document.
- d) **Do one of the following:**
 - 0) **Convey the Minimal Corresponding Source** under the terms of this License, **and the Corresponding Application Code** in a form suitable **for**, and under terms that permit, **the user to recombine or relink the Application with a modified version** of the Linked Version to produce a modified Combined Work, in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.
 - 1) **Use a suitable shared library mechanism for linking with the Library.** A suitable mechanism is **one that** (a) uses at run time a copy of the Library already present on the user's computer system, and (b) **will operate properly with a modified version of the Library** that is interface-compatible with the Linked Version.
 - e) **Provide Installation Information**, but only if you would otherwise be required to provide such information under section 6 of the GNU GPL, and only to the extent **that such information is necessary to install and execute a modified version of the Combined Work** produced by recombining or relinking the Application with a modified version of the Linked Version. (If you use option 4d0, the Installation Information must accompany the Minimal Corresponding Source and Corresponding Application Code. If you use option 4d1, you must provide the Installation Information in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.)

How to make use of KDE Frameworks

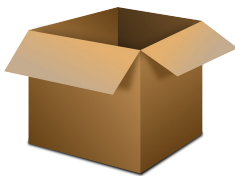
Prepare your development environment

Prepare your ~~developer~~ distribution environment

- How libraries are installed for development is be closely tied to how the app is packaged for distribution



Build it
yourself



Distribution's
packages



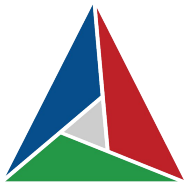
KDE Builder
or
kdesrc-build



KDE's Craft

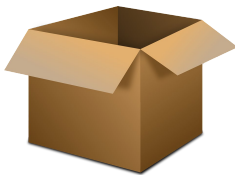
Prepare your distribution environment

Covers the most platforms,
requires most mastery



Build it yourself

Beginner friendly



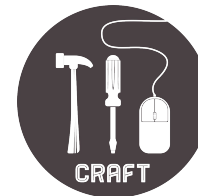
Distribution's packages

Linux only
The KDE way
Unsuitable for 3rd parties



KDE Builder
or
kdesrc-build

Windows
MacOS
ApplImage
*Android



KDE's Craft

KDE Craft



- Pros

- Robust packaging system built with Python
- You can use it to build Windows, Mac, AppImage, FreeBSD
- Easier than packaging apps yourself

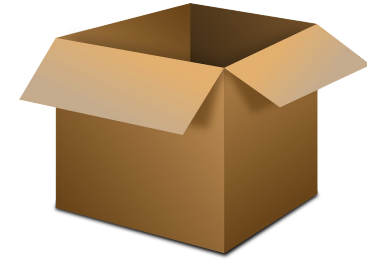
- Cons

- Constantly changing, hard to keep up
- Old versions of KDE frameworks are removed
- Android build scripts are tied to KDE's CI, which is only for KDE Projects



<https://community.kde.org/Craft>

Distribution's packages



- Pros

- Easy to get started...
Install the copy of the library listed with a *-dev* or *-devel* suffix
E.g.
libkf5solid-dev (Ubuntu)
kf6-solid-devel (Fedora)
- Easier to make packages
- Simply list your dependencies

- Cons

- Distributions update these packages sporadically

E.g.
Ubuntu 24.04 ships v5.115.0 of KDE frameworks, which is deprecated (using KDE Neon while targeting Debian is recommended)



Universal Packages



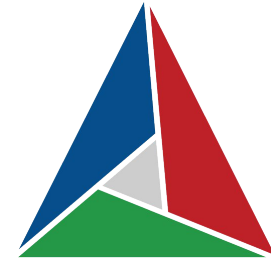
FLATPAK



- Similar to traditional Linux packages
 - Instead of installing additional packages, you link against a Runtime and SDK that brings all KDE Frameworks
- For Flatpak, use:
 - "runtime": "org.kde.Platform",
 - "runtime-version": "6.7", // Replace version number with the most current one
 - "sdk": "org.kde.Sdk",
 - Learn more at: <https://develop.kde.org/docs/packaging/flatpak/packaging/>
- For Snaps, use the latest variants of:
 - <https://snapcraft.io/kf6-core22>
 - <https://snapcraft.io/kf6-core22-sdk>
 - Learn more at: <https://ubuntu.com/tutorials/create-your-first-snap#1-overview>



Build it yourself



- Pros

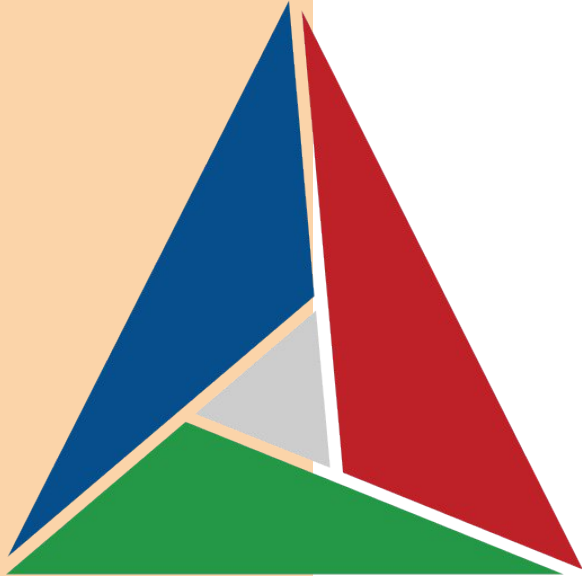
- Develop at your own pace
- Develop for architectures that are unsupported by KDE, like RISC-V and ARM64 Windows
- Distribute on platforms that are limited by KDE's current infrastructure, such as Android and WASM

- Cons

- For each platform you deploy
 - You build all libraries manually
 - You install all libraries manually
 - You create packages or installers manually

CMake

- Build system for C++ code
- It abstracts compilers, packaging tools, and other tools for build and distribution.
- CMake *Generators* will produce code suitable for building projects with vastly different compilers




<https://cmake.org>

Making use of KDE Frameworks

```
set(KF_MIN_VERSION 6.2.0) # Set to match the oldest supported distro
find_package(ECM REQUIRED NO_MODULE)
find_package(KF6 ${KF_MIN_VERSION} REQUIRED COMPONENTS
    CoreAddons
)
target_link_libraries(${PROJECT_NAME} PRIVATE
    KF6::CoreAddons
)
```

Making use of KDE Frameworks




Although other methods for locating libraries in CMake exists, [find_package](#) is recommended because it's what KDE frameworks themselves use

- All the libraries depend on ECM so, at the very least, ECM must be installed
- If you are able to install ECM, you should be able to install the other libraries



Due to hard coded values the KDE frameworks often don't compile if you add via [add_subdirectory](#)



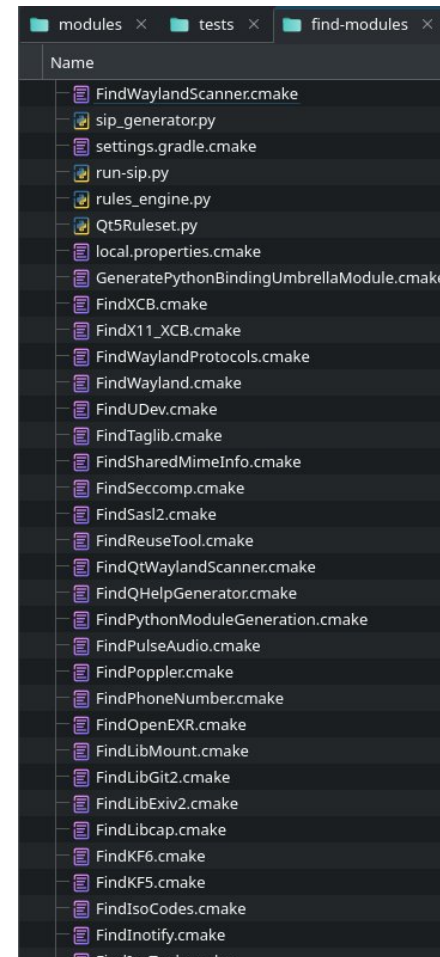
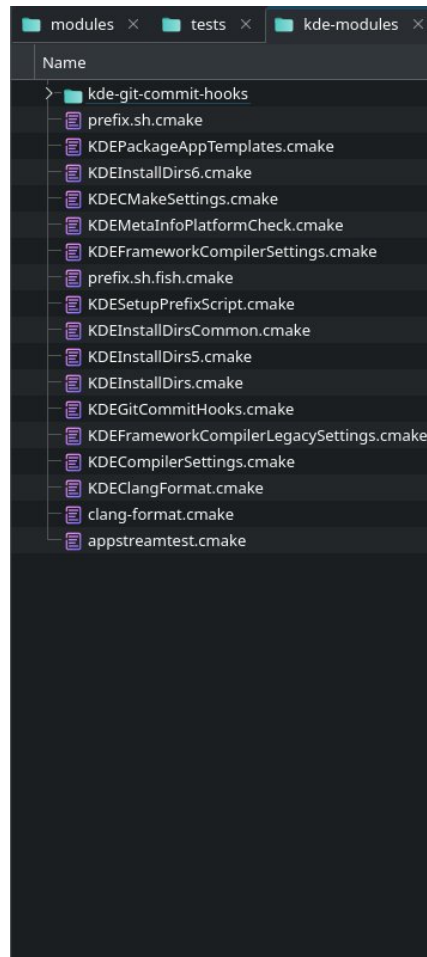
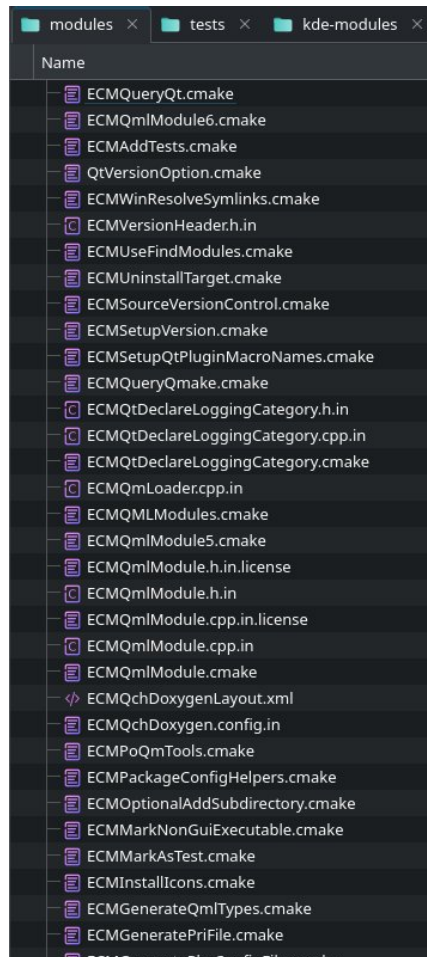
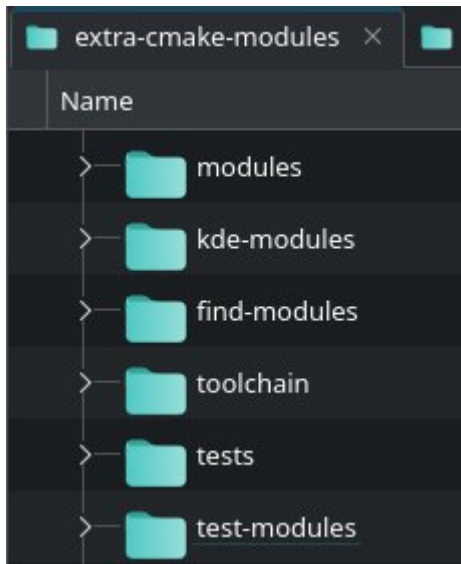
[ExternalProject_Add](#) might not work in CI environments without network access (Flatpak CI)

ECM

Extra CMake Modules

- Set of build scripts for CMake
- Ease building and distributing
- Required for building all KDE Frameworks

Extra CMake Modules



CMakeLists.txt

```
set(KF_MIN_VERSION 6.2.0) # Set to match the oldest supported distro
find_package(ECM REQUIRED NO_MODULE)
find_package(KF6 ${KF_MIN_VERSION} REQUIRED COMPONENTS
    CoreAddons )
target_link_libraries(${PROJECT_NAME} PRIVATE
    KF6::CoreAddons)
```

Install paths for Linux metadata

- # The following `KDE_...` variables are provided by ECM:

```
install(FILES myapp.appdata.xml DESTINATION ${KDE_INSTALL_METAINFODIR})  
install(PROGRAMS myapp.desktop DESTINATION ${KDE_INSTALL_APPDIR})
```

Set icons for app binary & installers

```
set(ICONS_FOLDER ${CMAKE_CURRENT_SOURCE_DIR}/icons/hicolor)
set(RASTER_ICONS
    ${ICONS_FOLDER}/16-apps-com.publisher.myapp.png
    [...] # Add icons from 16px to 512px.
    # ECM supports higher sizes but Flatpak does not.
    ${ICONS_FOLDER}/512-apps-com.publisher.myapp.png
set(VECTOR_ICON ${ICONS_FOLDER}/sc-apps-com.publisher.myapp.svg)
ecm_install_icons(
    ICONS ${RASTER_ICONS} ${VECTOR_ICON}
    DESTINATION ${KDE_INSTALL_ICONDIR})
ecm_add_app_icon(myapp_ICONS ICONS ${RASTER_ICONS})
add_executable(${PROJECT_NAME} [...] ${myapp_ICONS})
```

Matching app version everywhere

In CMake

```
project(myapp VERSION 1.0.0)
ecm_setup_version(${PROJECT_VERSION}
    VARIABLE_PREFIX MYAPP
    VERSION_HEADER "${CMAKE_CURRENT_BINARY_DIR}/myapp_version.h")
```

// In C++

```
#include "myapp_version.h"
MYAPP_VERSION_STRING
```

Build it yourself

- Set of build scripts for CMake
- Ease building and distributing
- Required for building all KDE Frameworks

Build dependencies

- All platforms
 - Git
 - Bash
 - Python 3
- In MacOS, also run:
 - `xcode-select -install`
- In Ubuntu, you can use:

```
sudo apt install python3 python3-pip python3-venv build-essential git wget
```


Manually build and install KDE Frameworks

```
# From a framework's root folder
mkdir build # create make a build folder

# Set the kinds of builds you want to create
CMAKE_CONFIGURATION_TYPES="Debug;Release;RelWithDebInfo;MinSizeRel"

# Set the build type that will be used to make a release
CMAKE_BUILD_TYPE="Release"

# CMAKE_PREFIX_PATH is the folder from which CMake searches for libraries
# Set it to point towards your Qt installation.
CMAKE_PREFIX_PATH="/Qt/6.7.2/gcc/"

# CMAKE_INSTALL_PREFIX is the folder where CMake install() instructions will
copy files to. We'll install KF alongside Qt.
CMAKE_INSTALL_PREFIX=$CMAKE_PREFIX_PATH
```

Manually build and install KDE Frameworks

Run cmake for preparations

```
cmake -DCMAKE_CONFIGURATION_TYPES=$CMAKE_CONFIGURATION_TYPES \  
      -DBUILD_TESTING=OFF \  
      -BUILD_QCH=OFF \  
      -DCMAKE_PREFIX_PATH=$CMAKE_PREFIX_PATH \  
      -DCMAKE_INSTALL_PREFIX=$CMAKE_INSTALL_PREFIX \  
      -B ./build .
```

Build the project with previous configuration

```
cmake --build ./build --config $CMAKE_BUILD_TYPE
```

Install the library to CMAKE_INSTALL_PREFIX

```
cmake --install ./build
```

A script to automatically build and install your libraries 1/8

- Bash can serve as a minimum common denominator across platforms.
 - It's shipped in GNU/Linux, MacOS, and is installed alongside Git on Windows through an MSYS environment
- Acquire your dependencies by downloading the repos as git submodules

```
git submodule add $PATH_TO_GIT_REPO
```

- `git submodule add https://github.com/KDE/kcoreaddons.git`
- `git submodule add https://invent.kde.org/frameworks/kcoreaddons.git`

A script to automatically build and install your libraries 2/8

```
if [[ "$OSTYPE" == "linux-gnu"* ]]; then
    PLATFORM="linux"
    COMPILER="gcc"
elif [[ "$OSTYPE" == "darwin"* ]]; then
    PLATFORM="macos"
    COMPILER="macos"
elif [[ "$OSTYPE" == "win32" || "$OSTYPE" == "msys" ]]; then
    PLATFORM="windows"
    COMPILER="msvc2019_64"
elif [[ "$OSTYPE" == "freebsd"* ]]; then
    PLATFORM="freebsd"
    COMPILER="gcc"
else
    PLATFORM="unix"
    COMPILER="gcc"
fi
```

A script to automatically build and install your libraries 3/8

```
# Update submodules  
echo "Downloading git submodules"  
git submodule update --init --recursive
```

A script to automatically build and install your libraries 4/8

```
# Satisfy KDE's Python dependencies
python3 -m venv venv
if [[ "$PLATFORM" == "windows" ]]; then
    source venv/Scripts/activate
else
    source venv/bin/activate
fi
python -m pip install --upgrade pip
python -m pip install -r requirements.txt
```

The following goes in requirements.txt and is needed to make *Release* builds:

```
sphinx
reuse
```



VCPKG

- Can satisfy most of KDE's 3rdparty dependencies
- Using "Classic mode" libraries are built to a central folder. We then copy them to our prefix
- Why not use VCPKG for installing KDE Frameworks?

<https://vcpkg.io/en/packages>

Add VCPKG as a git submodule

```
mkdir -p 3rdparty
```

```
cd 3rdparty
```

```
git add submodule https://github.com/microsoft/vcpkg.git
```


Initialize VCPKG

- run: `vcpkg new --application`
It will generate a `vcpkg-configuration.json` file like follows:

```
{
  "default-registry":
  {
    "kind": "git",
    "baseline": "509f71e53f45e46c13fa7935d2f6a45803580c07",
    "repository": "https://github.com/microsoft/vcpkg"
  },
  "registries": [
    {
      "kind": "artifact",
      "location": "https://github.com/microsoft/vcpkg-ce-catalog/archive/refs/heads/main.zip",
      "name": "microsoft"
    }
  ]
}
```

A script to automatically build and install your libraries 5/8

```
# Setup VCPKG
./3rdparty/vcpkg/bootstrap-vcpkg.sh -disableMetrics
if [[ "$PLATFORM" == "windows" ]]; then
    VCPKG=./3rdparty/vcpkg/vcpkg.exe
else
    VCPKG=./3rdparty/vcpkg/vcpkg
fi
# Install VCPKG packages
$VCPKG install --x-install-root "$CMAKE_PREFIX_PATH" package names go here
# Copy installed packages into install prefix
for package in ./3rdparty/vcpkg/packages/*; do
    echo $package
    cp -rf $package/* $CMAKE_PREFIX_PATH
done
```

A script to automatically build and install your libraries 6/8

```
# KDE Frameworks
tier_0=""
    ./3rdparty/extra-cmake-modules"
tier_1=""
    ./3rdparty/kcoreaddons
    ./3rdparty/ki18n
    ./3rdparty/kirigami"
tier_2=""
tier_3=""
```

A script to automatically build and install your libraries 7/8

```
CMAKE_CONFIGURATION_TYPES="Debug;Release;RelWithDebInfo;MinSizeRel"  
CMAKE_BUILD_TYPE="Release"
```

```
for dependency in $tier_0 $tier_1 $tier_2 $tier_3; do  
  echo -e "\n\n~~~" $dependency "~~~\n"  
  cmake -DCMAKE_CONFIGURATION_TYPES=$CMAKE_CONFIGURATION_TYPES \  
    -DBUILD_TESTING=OFF \  
    -BUILD_QCH=OFF \  
    -DCMAKE_PREFIX_PATH=$CMAKE_PREFIX_PATH \  
    -DCMAKE_INSTALL_PREFIX=$CMAKE_INSTALL_PREFIX \  
    -B ./$dependency/build ./$dependency/  
  cmake --build ./$dependency/build --config $CMAKE_BUILD_TYPE  
  cmake --install ./$dependency/build
```

done

A script to automatically build and install your libraries 8/8

```
echo "MyApp"
```

```
cmake -DCMAKE_CONFIGURATION_TYPES=$CMAKE_CONFIGURATION_TYPES  
      -DCMAKE_PREFIX_PATH=$CMAKE_PREFIX_PATH  
      -DCMAKE_INSTALL_PREFIX=$CMAKE_INSTALL_PREFIX  
      -B ./build .
```

```
cmake --build build --config $CMAKE_BUILD_TYPE
```

CPack

- Part of CMake
- Abstracts various packaging tools
- Can be used to make:
 - Deb & RPM packages
 - NSIS installers for Windows
 - Drag-and-Drop and PackageMaker installers for macOS



Using CPack to make builds

```
cd build  
cpack
```

Example CPack code in CMakeLists.txt

```
# CPACK: General Settings
set(CPACK_GENERATOR "DEB")
set(CPACK_PACKAGE_VENDOR "Javier O. Cordero Pérez")
set(CPACK_PACKAGE_CONTACT "redacted@email.com")
set(CPACK_PACKAGE_DESCRIPTION_FILE "${CMAKE_SOURCE_DIR}/${DESCRIPTION_FILE}")
set(CPACK_RESOURCE_FILE_README "${CMAKE_SOURCE_DIR}/${README_FILE}")
set(CPACK_RESOURCE_FILE_LICENSE "${CMAKE_SOURCE_DIR}/${LICENSE_FILE}")
set(CPACK_MONOLITHIC_INSTALL On)

set(CPACK_RPM_PACKAGE_LICENSE "GPLv3") # Also used by FreeBSD generator

set(ICONS_DIR "${CMAKE_SOURCE_DIR}/src/icons")
set(IMAGES_DIR "${CMAKE_SOURCE_DIR}/src/images")
set(COMPRESION_TYPE "xz")

# CPACK: Archive generator settings
set(CPACK_THREADS 0)
set(CPACK_ARCHIVE_THREADS 0)
```


Example CPack code in CMakeLists.txt

```
if(UNIX AND NOT ANDROID)
  # CPACK: DEB specific settings
  set(CPACK_DEBIAN_PACKAGE_SECTION "Multimedia")
  set(CPACK_DEBIAN_COMPRESSION_TYPE ${COMPRESION_TYPE})
  set(CPACK_DEBIAN_PACKAGE_DEPENDS "libqt6svg5 (>= 6.6.2), qml-module-qt-labs-platform (>=6.6.2), qml-module-qt-declarative (>=6.6.2)")

  # CPACK: RPM specific settings
  set(CPACK_RPM_PACKAGE_GROUP "Multimedia/Video")
  set(CPACK_RPM_PACKAGE_REQUIRES "qt6-qtbase >= 6.6.2, qt6-qtbase-gui >= 6.6.2, qt6-qtdeclarative >= 6.6.2, qt6-qtdatamodel >= 6.6.2")
  set(CPACK_RPM_COMPRESSION_TYPE ${COMPRESION_TYPE})
```

Example CPack code in CMakeLists.txt

```
elseif(WIN32)
    set(CPACK_GENERATOR "NSIS")
    set(CPACK_PACKAGE_EXECUTABLES "QPrompt" "QPrompt")
    set(CPACK_RESOURCE_FILE_LICENSE "${CMAKE_SOURCE_DIR}/${LICENSE_FILE}")
    set(CPACK_NSIS_EXECUTABLES_DIRECTORY "${BIN_INSTALL_DIR}")
    set(CPACK_NSIS_MUI_ICON "${ICONS_DIR}/qprompt.ico")
    set(CPACK_PACKAGE_ICON "${IMAGES_DIR}/installer.bmp")
    set(CPACK_NSIS_MUI_WELCOMEFINISHPAGE_BITMAP "${IMAGES_DIR}/welcome.bmp")
    set(CPACK_NSIS_MUI_UNWELCOMEFINISHPAGE_BITMAP "${IMAGES_DIR}/welcome.bmp")
    set(CPACK_NSIS_MUI_HEADERIMAGE "${IMAGES_DIR}/header.bmp")
    set(CPACK_NSIS_CONTACT ${CPACK_PACKAGE_CONTACT})
    set(CPACK_NSIS_INSTALLED_ICON_NAME "qprompt${CMAKE_EXECUTABLE_SUFFIX}")
    set(CPACK_NSIS_MENU_LINKS
        "${CMAKE_SOURCE_DIR}/${LICENSE_FILE}" "License"
        "${CMAKE_SOURCE_DIR}/${README_FILE}" "Readme"
    )
    set(CPACK_NSIS_MUI_FINISHPAGE_RUN "${CPACK_NSIS_INSTALLED_ICON_NAME}")
    set(InstallRequiredSystemLibraries On)
```

Example CPack code in CMakeLists.txt

```
elseif(APPLE AND NOT IOS)
    set(CPACK_GENERATOR "DragNDrop")
    set(CPACK_DMG_FORMAT "UDBZ")
    set(CPACK_DMG_VOLUME_NAME "QPrompt")
    set(CPACK_SYSTEM_NAME "OSX")
    set(CPACK_PACKAGE_FILE_NAME "QPrompt-${PROJECT_VERSION}")
    set(CPACK_PACKAGE_ICON "${ICONS_DIR}/qprompt.icns")
    set(CPACK_DMG_SLA_USE_RESOURCE_FILE_LICENSE ${LICENSE_FILE})
    set(CPACK_DMG_BACKGROUND_IMAGE "${IMAGES_DIR}/DMGBackground.png")
```

Example CPack code in CMakeLists.txt

```
elseif(BSD AND NOT APPLE)
    set(CPACK_FREEBSD_PACKAGE_MAINTAINER ${CPACK_PACKAGE_CONTACT})
    set(CPACK_FREEBSD_PACKAGE_ORIGIN "multimedia")
    set(CPACK_FREEBSD_PACKAGE_CATEGORIES "devel/qt6-base, x11-toolkits/qt6-declarative, graphics/qt6-svg, x11-toolkit")
endif()

include(CPack)
```



Include

To include (libraries) or not to include?

Depending on the platform you'll target is whether you will need to include the libraries packaged alongside your program



Not include

Include

- For Windows, MacOS and other OS without a traditional package manager
- For AppImages

Not include

- For traditional Linux package managers
- For Snap and Flatpak universal package formats

Tools to aggregate libraries

- windeployqt
- macdeployqt

```
$CMAKE_PREFIX_PATH/bin/windeployqt.exe \  
  ./build/bin/$CMAKE_BUILD_TYPE/MyApp.exe
```

```
$CMAKE_PREFIX_PATH/bin/macdeployqt.exe \  
  ./build/bin/MyApp
```

Copy the remaining libraries or re-run CMake commands with a different CMAKE_INSTALL_PREFIX

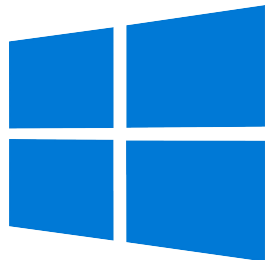


<https://www.youtube.com/watch?v=gnpPosTbttM>

Signatures



- Apple requires packages and binaries to be signed
- Your binaries, Qt and KDE Frameworks libraries will all need to be signed as well
- Signing is a requirement even if you won't distribute on the AppStore
- You will also need to sign your packages to distribute on Microsoft's Store



Additional resources

- CMake Tutorial. Code Tech Tutorials. (Feb 15, 2023)
<https://youtube.com/playlist?list=PLaIVdRk2RC6o5GHu618ARWh0VO0bFlif4&si=YllwK0VUZWDFZyk9>
- “Deploying Qt Applications (Windows|Mac|Linux)”. LearnQtGuide. (Sep 17, 2019).
<https://youtube.com/playlist?list=PLQMs5svASiXNx0UX7tVTncos4j0j9rRa4&si=RdgjxzE5kQRF4HUL>
- CMake and Qt. KDAB. (Sep 20, 2021)
https://youtube.com/playlist?list=PL6CJYn40gN6g1_yY2YkqSym7FWUId926M&si=u16RE2LgjUQyxFra
- Getting started with Kirigami. KDE.
<https://develop.kde.org/docs/getting-started/kirigami/>
- KDE’s Human Interface Guidelines. KDE.
<https://develop.kde.org/hig/>
- The KDE Frameworks.
<https://api.kde.org/frameworks/index.html>
- Repos in KDE Frameworks. KDE.
<https://invent.kde.org/frameworks>



Thank you for your time