Search

Building projects on the command line

This page explains how to configure and build existing projects. If you want to know how to create a Qt-based CMake project, see the documentation on how to get started with CMake.

To build a Qt project, CMake needs to know where the Qt installation is located. Usually this is done by setting the CMake variable CMAKE_PREFIX_PATH to Qt's installation prefix. If you are cross-compiling, see Cross-compiling for details on additional variables you will need to set.

If Qt is installed using the online installer, choose a Qt version within the top-level installation directory. For example, the following command shows how this is done on Windows:

```
cmake -DCMAKE_PREFIX_PATH=C:\Qt\6.4.0\msvc2019_64 -S <source-dir> -B <build-dir>
```

The `<source-dir>` and `<build-dir>` placeholders represent the source and build directories of your project.

CMake generators

CMake generates the necessary build system files that enable build tools such as GNU Make or Ninja to build your project.

CMake's default generator depends on the platform and build environment. For example on Windows, CMake generates Visual Studio project files if a Visual Studio environment is detected.

For a consistent developer experience on all platforms, use the `Ninja` or `Ninja Multi-Config` generator.

You can select the CMake generator either by setting the CMAKE_GENERATOR environment variable or using the `-G` argument:

```
cmake -G Ninja ...
```

qt-cmake

The `qt-cmake` script is a convenient alternative to configure your project. It eliminates the need for you to specify the CMAKE_PREFIX_PATH. You can find it located in the `bin` directory of your Qt installation prefix. The script passes all parameters to CMake, so you can use it just like you would `cmake\`:

```
C:\Qt\6.4.0\msvc2019_64\bin\qt-cmake -G Ninja -S <source-dir> -B <build-dir>
```

After the build system files are generated, your project is ready to be built:

```
cd <build-dir>
ninja
```

```
cmake --build <build-dir>
```

Cross-compiling

Building your project for a platform that is different from your development machine is called cross-compiling. An example is building for Android (the target platform) on a Windows machine (the host platform).

Cross-compiling with CMake requires a toolchain file for most platforms. It also requires a Qt version for the development host, in addition to a Qt version for the target platform. For example, you need Qt for Windows and Qt for Android installed to cross-compile for Android on Windows.

Use `qt-cmake` from the Qt installation for the target platform, to cross-compile your project for that platform:

This will configure your project for the target platform. The toolchain file is automatically passed, and possibly other platform-specific variables are set up.

Specifying a custom toolchain file

The `qt-cmake` script passes a Qt-internal toolchain file to CMake. This toolchain file sets several variables that are specific to Qt's target platform.

If you are using a Qt installation that has not been built on your machine, `qt-cmake` needs to know the location of the CMake toolchain file for the target platform.

In such a case, you can instruct `qt-cmake` to chainload a custom toolchain file by setting the `QT_CHAINLOAD_TOOLCHAIN_FILE` variable:

```
~/Qt/6.4.0/android_armv7/bin/qt-cmake -DQT_CHAINLOAD_TOOLCHAIN_FILE=<file-path> -S <source-dir> -B <build-dir>
```

This instructs Qt's internal toolchain file to load your custom toolchain file as well.

‹ Getting started with CMake

Imported targets ›