Building a QML application

In Building a C++ console application we showed the CMakeLists.txt file for a simple console application. We will now extend it to create a QML application that uses the Qt Quick module.

This is the full project file:

```
cmake_minimum_required(VERSION 3.16)

project(hello VERSION 1.0 LANGUAGES CXX)

set(CMAKE_AUTOMOC ON)
set(CMAKE_CXX_STANDARD 17)
set(CMAKE_CXX_STANDARD_REQUIRED ON)

find_package(Qt6 6.2 COMPONENTS Quick Gui REQUIRED)

qt_add_executable(myapp
    main.cpp
)

qt_add_qml_module(myapp
    URI hello
    VERSION 1.0
    QML_FILES
        main.qml
        FramedImage.qml
    RESOURCES
        img/world.png
)

target_link_libraries(myapp PRIVATE Qt6::Gui Qt6::Quick)
```

Let's walk through the changes we have made. We specify CMAKE_AUTOMOC, CMAKE_CXX_STANDARD, and CMAKE_CXX_STANDARD_REQUIRED.

```
set(CMAKE_AUTOMOC ON)
set(CMAKE_CXX_STANDARD 17)
set(CMAKE_CXX_STANDARD_REQUIRED ON)
```

Qt6::Quick targets we later link against.

```
find_package(Qt6 6.2 COMPONENTS Quick Gui REQUIRED)
```

Note that the application will still link against Qt6::Core, because Qt6::Quick depends on it.

qt_add_executable creates and finalizes an application target:

```
qt_add_executable(myapp
    main.cpp
)
```

qt_add_qml_module passes the target of the executable, a URI, module version, and a list of QML files to ensure that myapp becomes a QML module. Among other things, this places the QML files into `qrc:/${URI}` in the resource file system.

```
qt_add_qml_module(myapp
    URI hello
    VERSION 1.0
    QML_FILES
        main.qml
        FramedImage.qml
    RESOURCES
        img/world.png
)
```

First, `qt_add_qml_module` ensures that `qmlcachegen` runs. Second, it creates a `myapp_qmllint` target, which runs `qmllint` on the files in QML_FILES.

By adding the referenced resources, they get automatically added to the application under the same root path as the QML files – also in the resource file system. By keeping the path in the resource system consistent with the one in the source and build directory, we ensure that the image is always found, as it is resolved relative to FramedImage.qml. It refers to the image in the resource file system if we load main.qml from there, or to the one in the actual file system if we review it with the `qml` tool.

In the `target_link_libraries` command, we link against `Qt6::Quick` instead of `Qt6::Core`.

```
target_link_libraries(myapp PRIVATE Qt6::Gui Qt6::Quick)
```

‹ Getting started with CMake Building a reusable QML module ›

DOCUMENTATION

The Qt Company

Contact Us

Company

About Us
Investors
Newsroom
Careers
Office Locations

Licensing

Terms & Conditions
Open Source
FAQ

Support

Support Services
Professional Services
Partners
Training

For Customers

Support Center
Downloads
Qt Login
Contact Us
Customer Success

Community

Contribute to Qt
Forum
Wiki
Downloads
Marketplace

Feedback Sign In