Designer-Developer Workflow

Note: In this section, you are using advanced menu items. These are not visible by default. To toggle the visibility of advanced menu items, see Customizing the Menu.

Qt Design Studio enables designers and developers to work together on common projects to develop applications. Designers use the views in the **Design** mode to modify UI files (*.ui.qml*), whereas developers use Qt Creator to work on the Qt Quick (*.qml*) and other files that are needed to implement the application logic and to prepare the application for production.

Use the Git version control system to ensure that changes are not lost when files are passed back and forth between designers and developers.

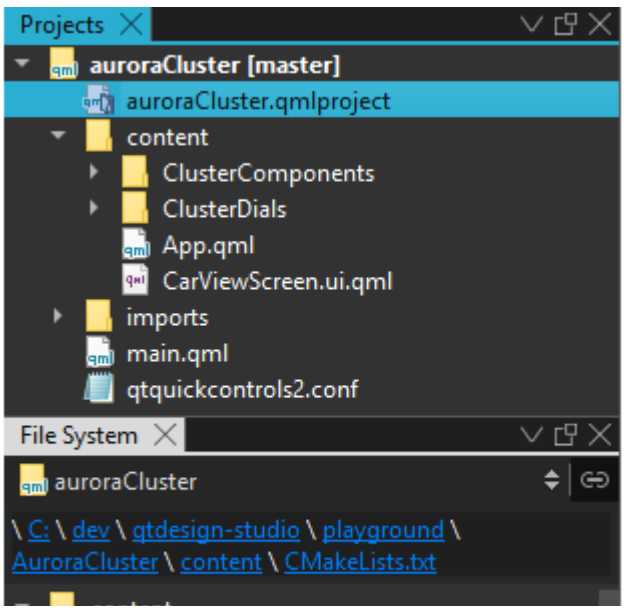
Qt Design Studio projects come with boilerplate code for a working Qt 6 application that you can build and run in Qt Creator using CMake. Therefore, you can open, build, and run the projects with Qt Creator.

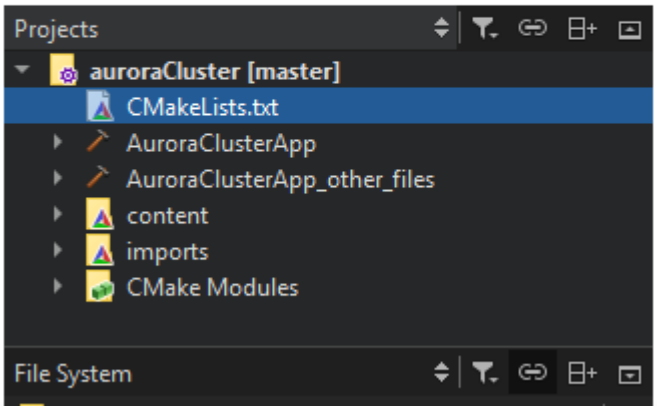
Qt Design Studio continues to use the *.qmlproject* file format, while Qt Creator uses a *CMakeLists.txt* file as the project file. This enables you to share your project as a fully working C++ application with developers.

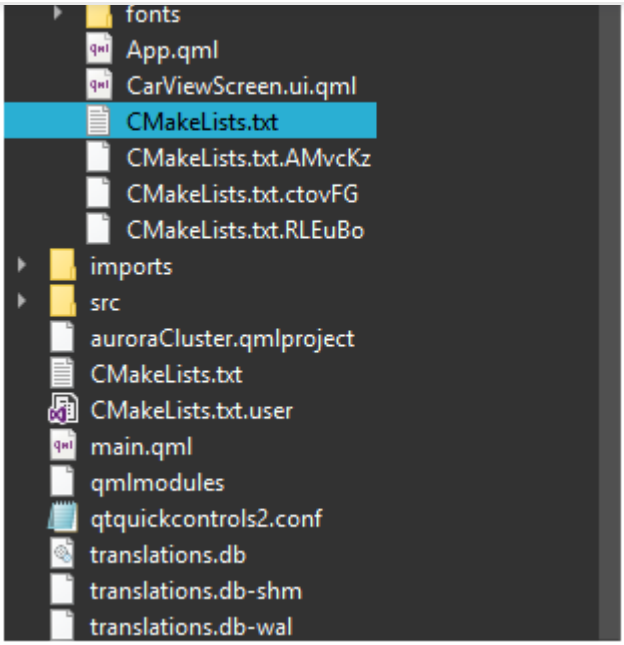
If you add or remove QML files in Qt Design Studio, you have to regenerate the *CMakeLists.txt* project configuration file by selecting **Build** > **Run** > **Generate CMakeLists.txt Files**.

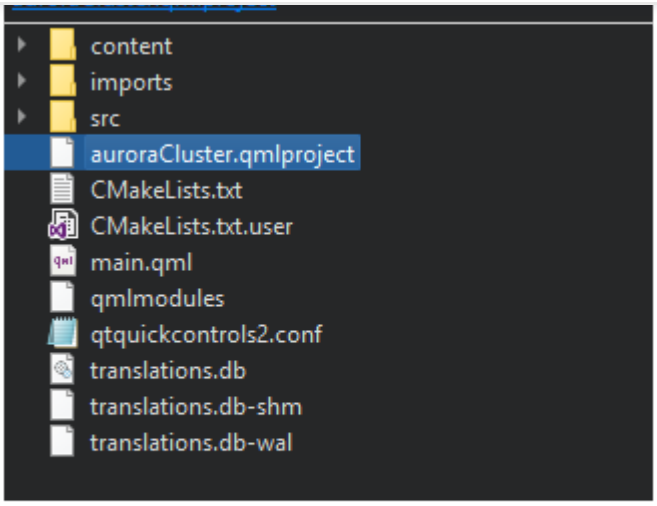
If you use Git, you can clone an example project here.

The following image shows the example project structure and contents in the Projects and File System views in Qt Design Studio and Qt Creator:









Converting Project Structure for CMake

Qt Design Studio can generate *CMakeLists.txt* and other related files to use with Qt Creator and to compile into an executable application but only if the project has a certain folder structure. If you have a Qt Design Studio QML project that doesn't have the CMake configuration, follow these steps to convert it's file structure to correct format.

1. Create a folder named *content* in the project's folder. This folder contains the application's main module.
2. Move all QML files of the project's main module to the *content* folder. If your project has multiple modules, place the other modules in the *imports* or *asset_imports* folder.
3. If your project's main module has resource folders such as *fonts* or *images*, move them to the *content* folder.
4. Create a folder named *src* in the project's folder. This folder contains C++ code for compiling the project.
5. If your project doesn't have an *imports* folder for other QML modules, create it now even if you do not have other modules. The CMake file generator expects it.
6. In the project's *.qmlproject* file:
   - Add *'.'* in importPaths. For example:

     ```
     importPaths: [ "imports", "asset_imports", "." ]
     ```

   - Change mainFile to *'content/App.qml'*:

     ```
     mainFile: "content/App.qml"
     ```

7. In the *content* folder, create a file named *App.qml* and add the following content:

   ```
   import QtQuick
   import QtQuick.Window
   import YourImportModuleHere
   ```

```
        visible: true
        title: "YourWindowTitleHere"
        <YourMainQmlClassHere> {
        }
    }
```

8. In *App.qml*, modify imported modules, window dimensions, window title, and main QML class appropriately.

> **Note:** This template assumes that your project has a module named *YourImportModuleHere* in the *imports* folder containing a singleton class named *Constants*. This isn't mandatory.

9. Generate CMake files and C++ source files that are used to compile the application into an executable file by selecting **Build** > **Generate CMakeLists.txt files**.

‹ Implementing Applications

Coding ›