Using QML Modules with Plugins

QML modules may use C++ plugins to expose components defined in C++ to QML applications.

To create a QML module and make it appear in the Components view:

1. Create custom components and place all the `.qml` files in a directory dedicated to your module. For example: `imports\asset_imports`.
2. For Qt Quick UI projects (.qmlproject), specify the path to the directory that contains the module in the .qmlproject file of the application where you want to use the module as a value of the `importPaths` variable. For example `importPaths: [ "imports", "asset_imports" ]`.
3. Create a `qmldir` file for your module and place it in the module directory. For more information, see Module Definition qmldir Files.
4. Create a `qmltypes` file, as instructed in Generating Type Description Files.
5. Create a directory named `designer` in your module directory.
6. Create a `.metainfo` file for your module and place it in the `designer` directory. Meta information is needed to display the components in **Components**. Use a metainfo file delivered with Qt, such as `qtquickcontrols2.metainfo`, as an example.
7. Build your module using the same Qt version and compiler as Qt Design Studio. For more information, see Running QML Modules in Design Mode.

Your module should now appear in **Components**. Your components should appear in a subsection of **Components** if a valid `.metainfo` file is in place.

Generating Type Description Files

When registering QML types, make sure that the QML module has a `plugins.qmltypes` file. Ideally, it should be located in the same directory as the `qmldir` file. The `qmltypes` file contains a description of the components exported by the module's plugins and is loaded by Qt Design Studio when the module is imported.

For more information, see Type Description Files.

Dumping Plugins Automatically

If a module with plugins lacks the `qmltypes` file, Qt Design Studio tries to generate a temporary file itself by running the `qmldump` program in the background. However, this automatic dumping is a fallback mechanism with many points of failure and you cannot rely upon it.

Importing QML Modules

By default, Qt Design Studio will look in the QML import path of Qt for QML modules.

If you use CMake, add the following command to the CMakeLists.txt file to set the QML import path:

```
set(QML_IMPORT_PATH ${CMAKE_SOURCE_DIR}/qml ${CMAKE_BINARY_DIR}/imports CACHE STRING "" FORCE)
```

The import path affects all the targets built by the CMake project.

Running QML Modules in Design Mode

A QML emulation layer (also called QML Puppet) is used in the **Design** mode to render and preview images and to collect data. To be able to render custom components correctly from QML modules, the emulation layer must be built with the same Qt version and compiler as the QML modules.

On Windows, select **Help** > **About Qt Design Studio** to check the Qt version and compiler that you need to use to build your plugin. For example: `Based on Qt 5.15.2 (MSVC 2019, 64 bit)`.

On macOS, select **Qt Design Studio** > **About Qt Design Studio** to see something like: `Based on Qt 5.15.2 (Clang 10.0 (Apple), 64 bit)`.

A plugin should behave differently depending on whether it is run by the emulation layer or an application. For example, animations should not be run in the **Design** mode. You can use the value of the `QML_PUPPET_MODE` environment variable to check whether the plugin is currently being run by an application or edited in the **Design** mode.

If you want to use a different module in the **Design** mode than in your actual application for example to mockup C++ items, you can use `QML_DESIGNER_IMPORT_PATH` in the `.pro` file (for qmake projects), or declare and set the property `qmlDesignerImportPaths` in your product (for Qbs projects). Modules in the import paths defined in `QML_DESIGNER_IMPORT_PATH` will be used only in the **Design** mode. For an example, see Qt Quick Controls - Contact List.

‹ Simulating Dynamic Systems

Dynamic Behaviors ›