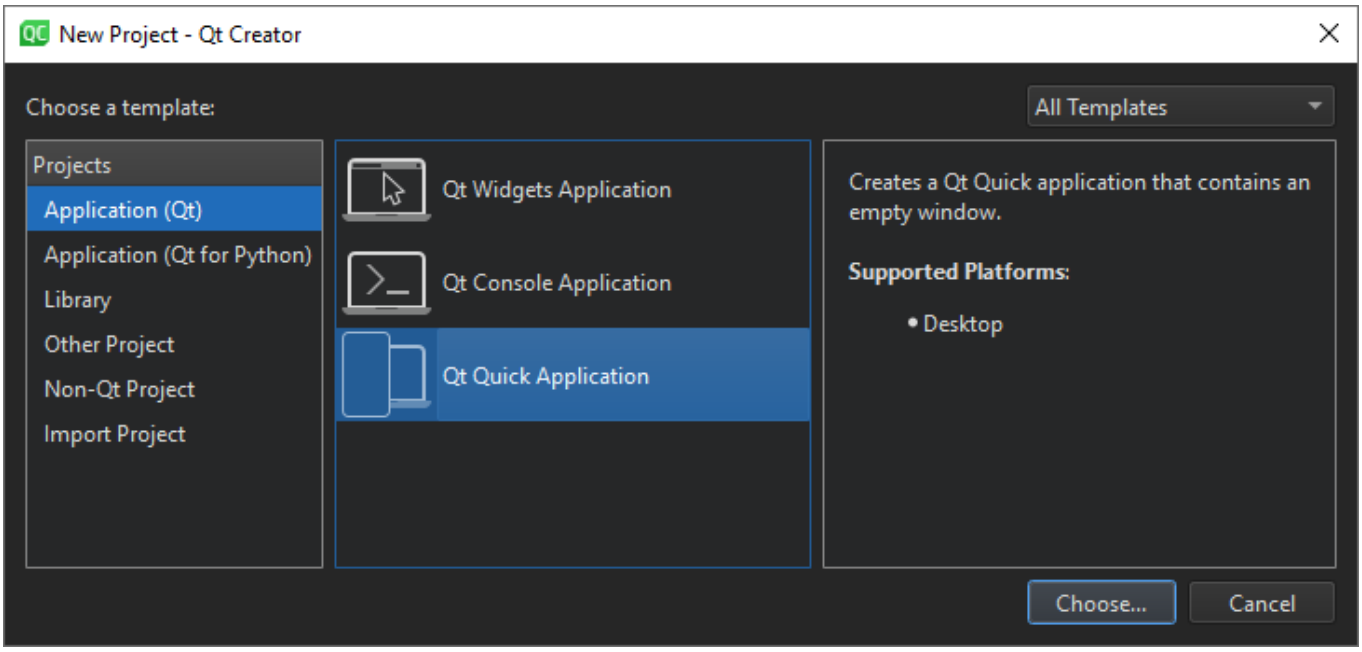
Creating Qt Quick Projects



The following table lists the wizard templates for creating a new Qt Quick project from scratch.

Category	Wizard Template	Purpose
Application (Qt)	Qt Quick Application	Creates a Qt Quick 2 application project that can contain both QML and C++ code. You can build the application and deploy it to desktop, embedded, and mobile target platforms.
Application (Qt for Python)	Qt for Python - Qt Quick Application	Creates a Python project that contains an empty Qt Quick Application.
Other Project	Qt Quick UI Prototype	Creates a Qt Quick UI project with a single QML file that contains the main view. You can preview Qt Quick 2 UI projects in the QML Scene preview tool. You do not need to build them because they do not contain any C++ code. This project type is compatible with Qt Design Studio. However, use this template only if you are prototyping. You cannot create a full application by using this template. Qt Quick UI projects cannot be deployed to embedded or mobile target platforms. For those platforms, create a Qt Quick application instead.
Library	Qt Quick 2 Extension	Creates C++ plugins that make it possible to offer extensions that can be loaded dynamically into Qt Quick 2 applications.

Category Wizard Template Purpose

Note: The SDK for a particular target platform might install additional templates for that platform. For example, the QNX templates are installed as part of the QNX SDK.

Qt Creator creates the necessary boilerplate files. Some of the files are specific to a particular target platform.

Creating Qt Quick Applications

1. Select **File** > **New Project** > **Application (Qt)** > **Qt Quick Application** > **Choose**.
2. In the **Project Location** dialog, **Name** field, enter a name for the project. Keep in mind that you cannot easily change the project name later.
3. In the **Create in** field, enter the path for the project files. Select the **Use as default project location** check box to create new projects in this folder by default. You can move project folders later without problems.
4. Select **Next** (or **Continue** on macOS) to open the **Define Build System** dialog.
5. In the **Build system** field, select the build system to use for building and running the project: qmake, CMake, or Qbs.
6. Select **Next** to open the **Define Project Details** dialog.
7. Select the Qt version to develop with in the **Minimum required Qt version** field. The Qt version determines the Qt Quick imports that are used in the QML files.
8. Select the **Use Qt Virtual Keyboard** check box to add support for Qt Virtual Keyboard to the application.

   Note: If you have not installed the Qt Virtual Keyboard module when you installed Qt, an error message will appear when you try to open *main.qml* for editing. You can use the Qt Maintenance Tool to install Qt Virtual Keyboard.

9. Select **Next** to open the **Translation File** dialog.
10. In the **Language** field, select a language that you plan to translate the application to. You can add other languages later by editing the project file.
11. In the **Translation file** field, you can edit the name for the translation source file that will be generated for the selected language.
12. Select **Next** to open the **Kit Selection** dialog.
13. Select kits for the platforms that you want to build the application for.

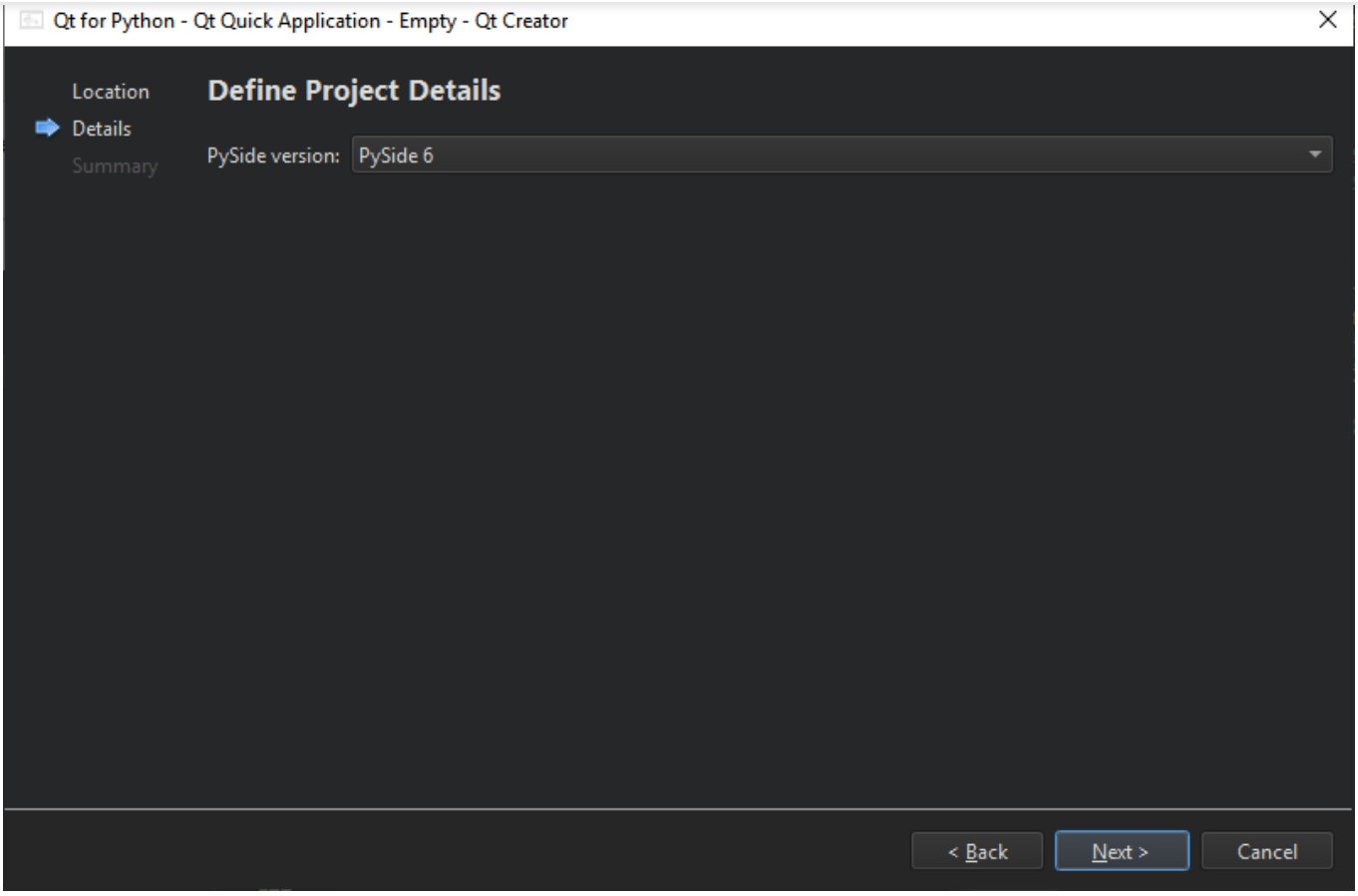
    Note: Kits are listed if they have been specified in **Edit** > **Preferences** > **Kits** (on Windows and Linux) or in **Qt Creator** > **Preferences** > **Kits** (on macOS). For more information, see Adding Kits.

14. Select **Next** to open the **Project Management** dialog.
15. Review the project settings, and select **Finish** (on Windows and Linux) or **Done** (on macOS) to create the project.

Qt Creator creates a QML file, *main.qml*, that you can modify in the **Edit** mode.

Creating Qt Quick Based Python Applications

The **Qt for Python - Qt Quick Application - Empty** wizard enables you to create a Python project that contains a



The wizard adds the following imports to the source file to provide access to QGuiApplication and QQmlApplicationEngine:

```
import sys
from pathlib import Path

from PySide6.QtGui import QGuiApplication
from PySide6.QtQml import QQmlApplicationEngine
```

The wizard also adds a main function, where it creates a QGuiApplication instance and passes system arguments to the QGuiApplication object:

```
if __name__ == "__main__":
    app = QGuiApplication(sys.argv)
...
```

The following lines in the main class create a QQmlApplicationEngine instance and load the generated QML file to the engine object:

```
engine = QQmlApplicationEngine()
qml_file = Path(__file__).resolve().parent / "main.qml"
engine.load(qml_file)
```

application exits with an error code. If loading succeeds, the wizard calls the `app.exec()` method to enter the Qt main loop and start executing the Qt code:

```
if not engine.rootObjects():
    sys.exit(-1)
sys.exit(app.exec())
```

Open the .qml file in the **Edit** mode to design a Qt Quick UI, or use Qt Design Studio.

Creating Qt Quick UI Projects

Qt Quick UI Prototype projects are useful for testing or prototyping user interfaces, or for setting up a separate project just for QML editing, for example. You cannot use them for application development because they do not contain:

- C++ code
- Resource files (.qrc)
- Code needed for deploying applications to devices

For more information about how to turn Qt Quick UI Prototype projects into Qt Quick Application projects, see Converting UI Projects to Applications.

To create a Qt Quick UI Prototype project:

1. Select **File** > **New Project** > **Other Project** > **Qt Quick UI Prototype**.
2. Select **Choose** to open the **Project Location** dialog.
3. In the **Name** field, enter a name for the application.
4. In the **Create in** field, enter the path for the project files. Select the **Use as default project location** check box to create new projects in this folder by default.
5. Select **Next** (or **Continue** on macOS) to open the **Define Project Details** dialog.
6. In the **Minimum required Qt version** field, select the Qt version to develop with. The Qt version determines the Qt Quick imports that are used in the QML files.

   You can add imports later to combine Qt Quick basic types with Qt Quick Controls, Qt Quick Dialogs, and Qt Quick Layouts (available since Qt 5.1).

7. Select the **Use Qt Virtual Keyboard** check box to add support for Qt Virtual Keyboard to the application.

   > **Note:** If you have not installed the Qt Virtual Keyboard module when you installed Qt, an error message will appear when you try to open *main.qml*.

8. Select **Next** to open the **Kit Selection** dialog.
9. Select kits for the platforms that you want to build the application for.

   > **Note:** Kits are listed if they have been specified in **Edit** > **Preferences** > **Kits** (on Windows and Linux) or in **Qt Creator** > **Preferences** > **Kits** (on macOS). For more information, see Adding Kits.

11. Review the project settings, and select **Finish** (on Windows and Linux) or **Done** (on macOS) to create the project.

Qt Creator creates the following files:

- .qmlproject project file defines that all QML, JavaScript, and image files in the project folder belong to the project. Therefore, you do not need to individually list all the files in the project.
- .qml file defines a UI item, such as a component or the whole application UI.
- ui.qml file defines a form for the application UI. This file is created if you selected the **With .ui.qml file** check box.

To use JavaScript and image files in the application, copy them to the project folder.

‹ Developing Qt Quick Applications

Using Qt Quick Designer ›