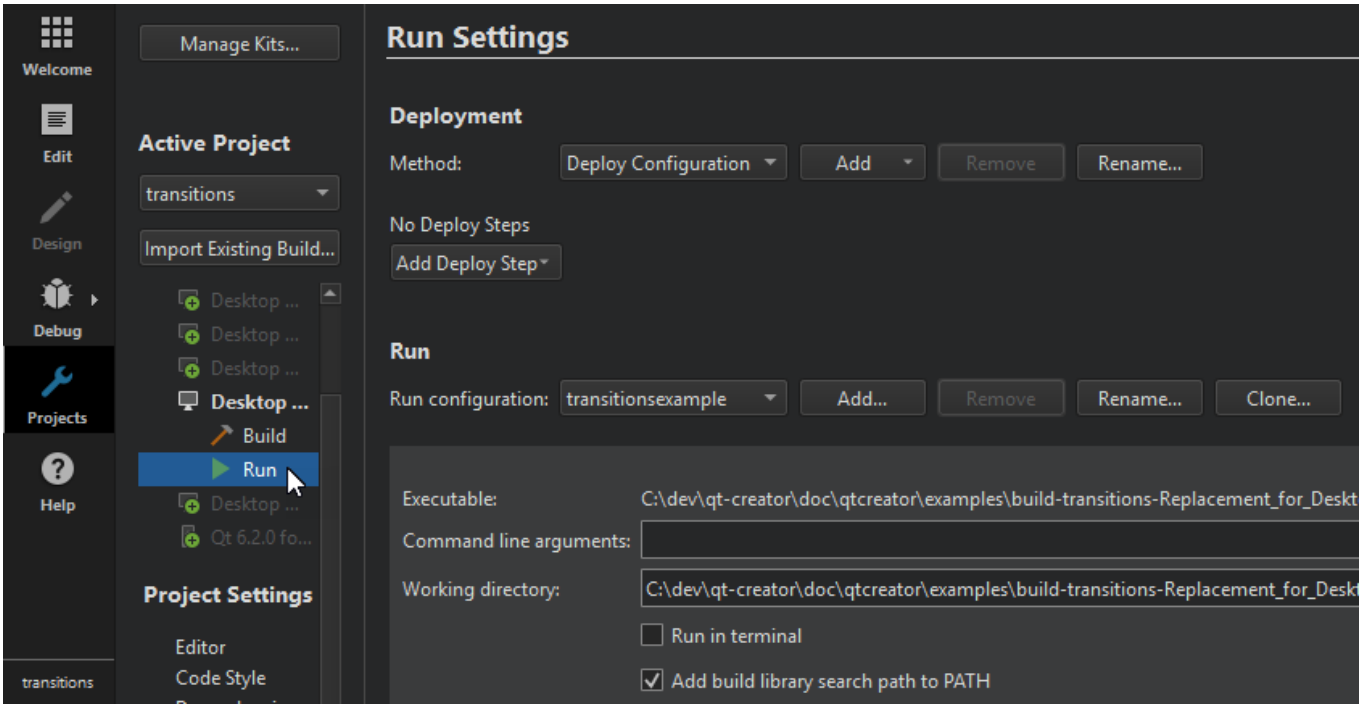
指定运行设置

要指定的运行设置取决于项目的类型以及用于生成和运行项目的工具包。

Qt 创建器会自动为您的项目创建运行配置。若要查看和修改它们，请选择"**项目**>**生成**" > "**运行**>**运行**"。



要防止 Qt Creator 自动创建运行配置，请选择**编辑**>**首选项**>**生成 & 运行**，然后取消选中 **自动创建合适的运行配置** 复选框。

管理运行配置

可用的运行配置在"**运行配置**"字段中列出。若要为项目添加运行配置，请选择"**添加**"。要添加基于当前运行配置的运行配置，请选择**克隆**。

若要重命名当前运行配置，请选择"**重命名**"。

若要删除当前运行配置，请选择"**删除**"。

qmake 项目的运行配置从已分析的 .pro 文件派生其可执行文件。有关如何构造命令的详细信息，请参阅启动外部进程。

选择默认运行目标

运行目标

使用 CMake 时，可以通过在项目文件中将属性的值设置为值来筛选运行目标列表。例如：
`qtc_runnable`FOLDER`CMakeLists.txt`

```
set_target_properties(main_executable PROPERTIES FOLDER "qtc_runnable")
```

如果未指定任何项目，Qt Creator 会自动为 中指定的所有目标添加运行配置。
`qtc_runnable`CMakeLists.txt

qmake 运行目标

使用 qmake 时，您可以通过在要运行的应用程序项目的 .pro 文件中指定变量 （） 来防止 Qt Creator 自动为子项目创建运行配置。例如`qtc_runnable`TEMPLATE=app

```
CONFIG += qtc_runnable
```

如果没有一个应用程序项目指定， Qt Creator 将为所有应用程序项目创建运行配置。`qtc_runnable`

如果任何应用程序项目指定， Qt Creator 仅为在其 .pro 文件中也设置的子项目创建运行配置。
`qtc_runnable`CONFIG += `qtc_runnable`

有关 qmake 项目模板的更多信息，请参见模板。

介子运行目标

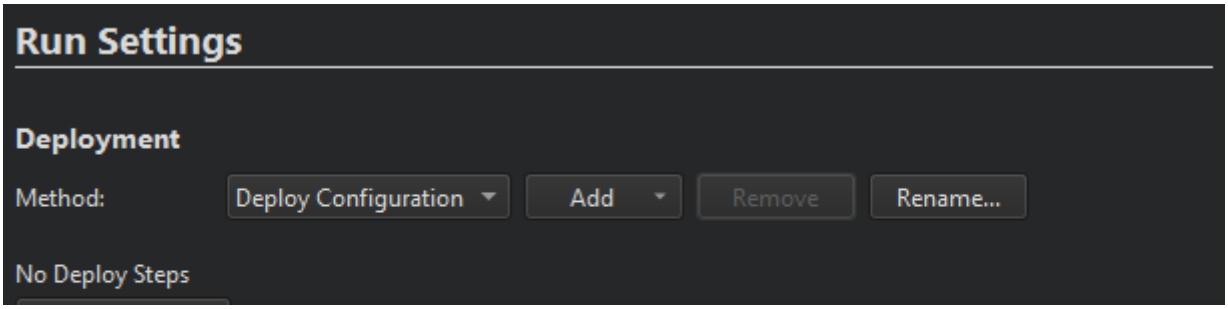
Qt Creator 会自动为 Meson 构建描述中用函数声明的所有目标添加运行配置。`executable()`

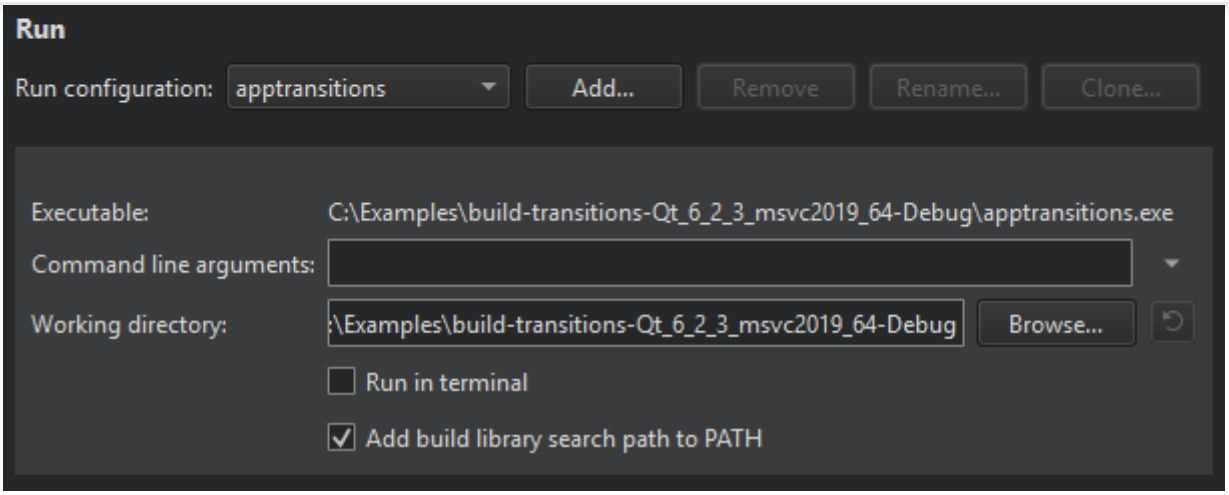
Specifying Run Settings for Desktop Device Types

You can specify command line arguments to be passed to the executable and the working directory to use. The working directory defaults to the directory of the build result.

For console applications, check the **Run in terminal** check box. To specify the terminal to use on Linux and macOS, select **Edit** > **Preferences** > **Environment** > **System**.

To run with special environment variables set up, select them in the **Run Environment** section. For more information, see Selecting the Run Environment.





When building an application, Qt Creator creates a list of directories where the linker will look for libraries that the application links to. By default, the linked libraries are made visible to the executable that Qt Creator is attempting to run. Usually, you should disable this option only if it causes unwanted side-effects or if you use deployment steps, such as , and want to make sure that the deployed application will find the libraries also when it is run without Qt Creator.`make install`

To disable library linking for the current project, deselect the **Add build library search path to PATH** check box. To disable library linking for all projects, select **Edit** > **Preferences** > **Build & Run**, and then deselect the **Add linker library search paths to run environment** check box.

The **Use debug version of frameworks (DYLD_IMAGE_SUFFIX=_debug)** option (only available on macOS) enables you to debug (for example, step into) linked frameworks, such as the Qt framework itself. You do not need this option for debugging your application code.

On Linux, select the **Run as root user** check box to run the application with root user permissions.

You can also create custom executable run configurations where you can set the executable to be run. For more information, see Specifying a Custom Executable to Run.

Specifying Valgrind Settings

Qt Creator integrates Valgrind code analysis tools for detecting memory leaks and profiling function execution. You can configure the tools according to your needs.

You can specify Valgrind settings either globally for all projects or separately for each project.

To specify Valgrind settings for the current project:

1. In the **Valgrind Settings** section, select **Custom**.
2. Specify Valgrind settings for the project.

4. In **Valgrind arguments**, specify additional arguments for Valgrind.
5. In **Detect self-modifying code**, select whether to detect self-modifying code and where to detect it: only on stack, everywhere, or everywhere except in file-backend mappings.

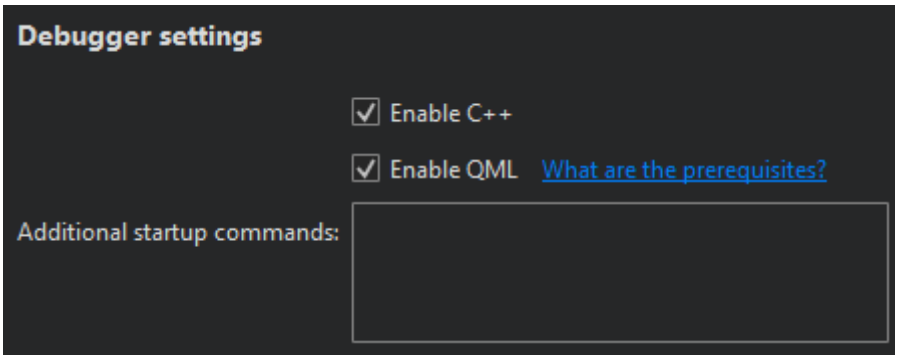
For more information about the CallGrind and MemCheck settings, see:

- Selecting Profiling Options
- Selecting Options for Memory Analysis

Click **Restore Global** to revert to the global settings.

To specify global Valgrind settings, select **Edit** > **Preferences** > **Analyzer**.

Enabling Debugging



To select the languages to debug, select the **Enable C++** and **Enable QML** check boxes.

> **Note:** Opening a socket at a well-known port presents a security risk. Anyone on the Internet could connect to the application that you are debugging and execute any JavaScript functions. Therefore, you must make sure that the port is properly protected by a firewall.

Optionally, in **Additional startup commands**, you can enter additional settings for debugging C++:

- Custom debugging helpers
- GDB commands to execute after GDB has started, but before the debugged program is started or attached, and before the debugging helpers are initialized

However, you can usually leave this field empty.

For more information about debugging, see Debugging.

Specifying Run Settings for Android Devices

To run and debug an application on an Android device, you must create connections from the development host to the device, as instructed in Connecting Android Devices.

A default set of Android Activity manager (am) start options is applied when starting applications. You can specify additional start options in the **Activity manager start arguments** field. However, if the default options conflict with the added options, the application might not start.

```
am start -n <package_name>/<QtActivity_name>
```

The default arguments for the Activity manager for the debugger mode:

```
am start -n <package_name>/<QtActivity_name> -D
```

For example, to run the application as a particular user, enter the start option , where is the user ID of the user account.`--user 1010`

You can specify shell commands to run before the application is started and after it is quit. For example, enter the following commands into **Pre-launch on-device shell commands** to unlock the screen and to switch to the user account on the device before running the application:10

```
input keyevent 82
am switch-user 10
```

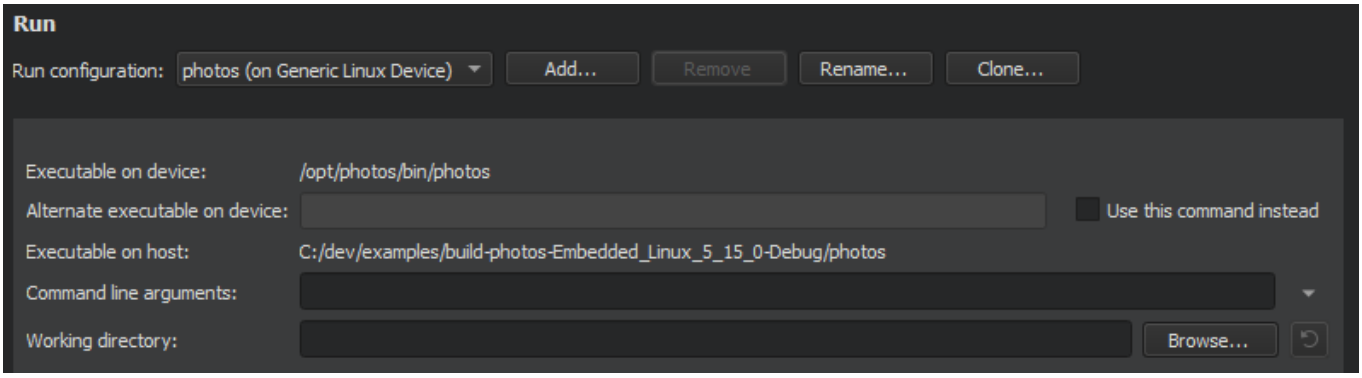
Enter the following commands into **Post-quit on-device shell commands** to switch back to the default user, , and to unlock the screen after the application is quit:0

```
am switch-user 0
input keyevent 82
```

Specifying Run Settings for Linux-Based Devices

To run and debug an application on a Linux-based device, you must create connections from the development host to the device and add the device configurations to kits. Click **Manage Kits** to add devices to kits. For more information, see Connecting Generic Remote Linux Devices.

another application launches your application, for example, enter the command in the **Alternate executable on device** field and select the **Use this command instead** check box.



You can specify arguments to pass to your application in the **Command line arguments** field.

Specifying Run Settings for QNX Devices

To run and debug an application on a QNX device, you must create connections from the development PC to the device. Click **Manage device configurations** to create a connection. For more information, see Connecting QNX Devices.

Specifying run settings for QNX Neutrino devices is very similar to Specifying Run Settings for Linux-Based Devices.

Specifying Run Settings for Boot2Qt Devices

To run and debug an application on a Boot2Qt device (commercial only), you must create connections from the development host to the device and add the device configurations to kits. Select **Manage Kits** to add devices to kits. For more information, see Boot2Qt: Installation Guide.

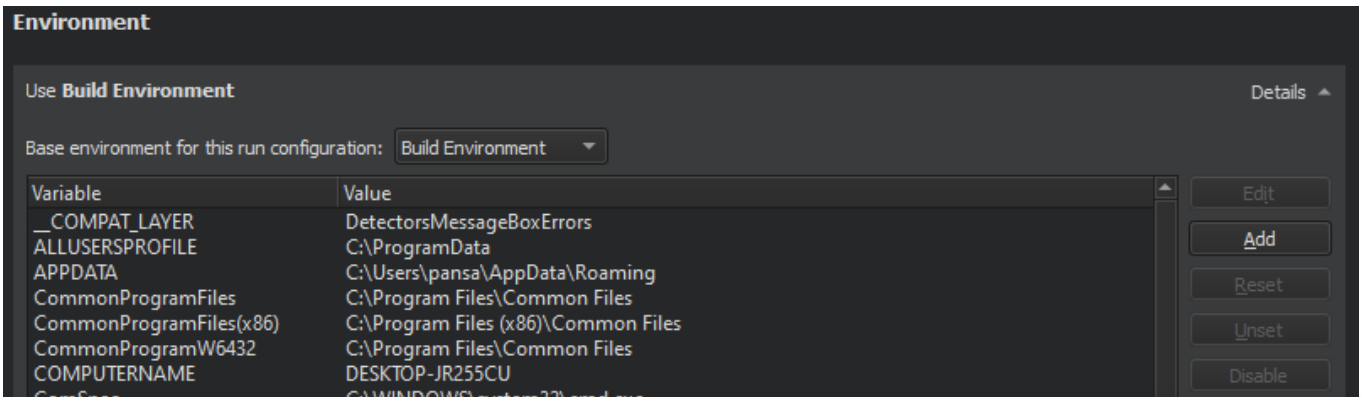
The run settings display the path to the executable file on the development host and on the device.

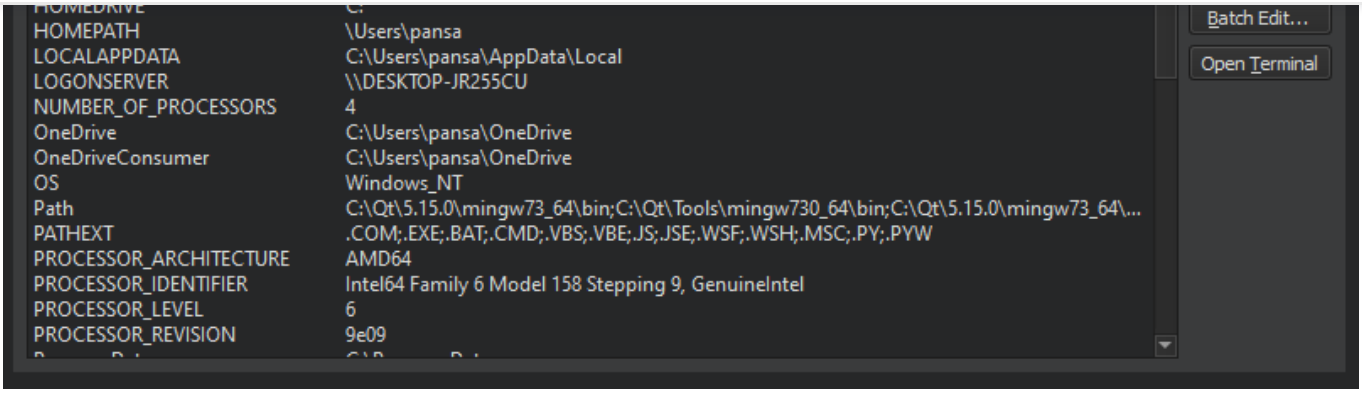
For more information on the deployment steps, see Deploying Applications to Boot2Qt Devices.

Selecting the Run Environment

Qt Creator automatically selects the environment used for running the application based on the device type. You can edit the environment or select another environment in the **Run Environment** section.

You can edit existing environment variables or add, reset and unset new variables.





When running on the desktop, the **Build Environment** is used by default, but you can also use the **System Environment** without the additions made to the build environment. For more information, see Build Environment and Specifying Environment Settings.

To run in a clean system environment, select **Clean Environment**.

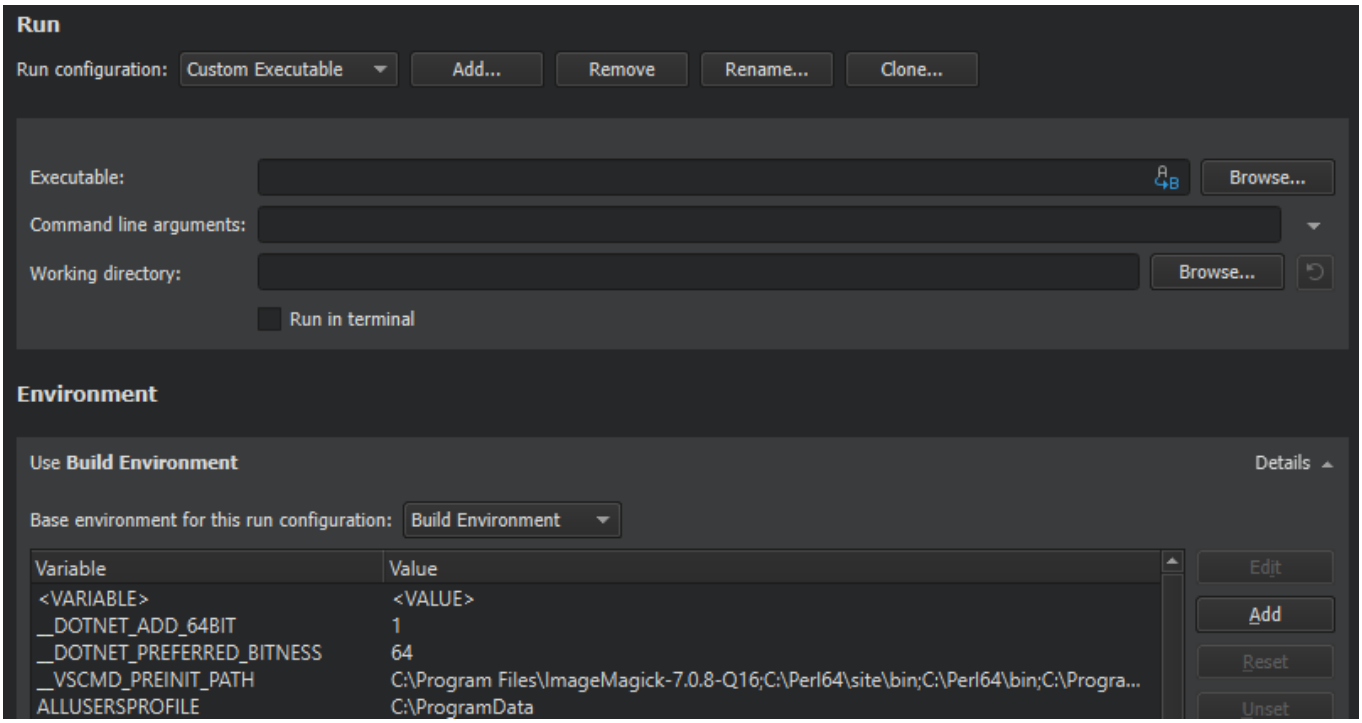
When running on a mobile device connected to the development host, Qt Creator fetches information about the **Device Environment** from the device. Usually, it does not make sense to edit the device environment.

To modify the environment variable values for the run environment, select **Batch Edit**. For more information, see Batch Editing.

Specifying a Custom Executable to Run

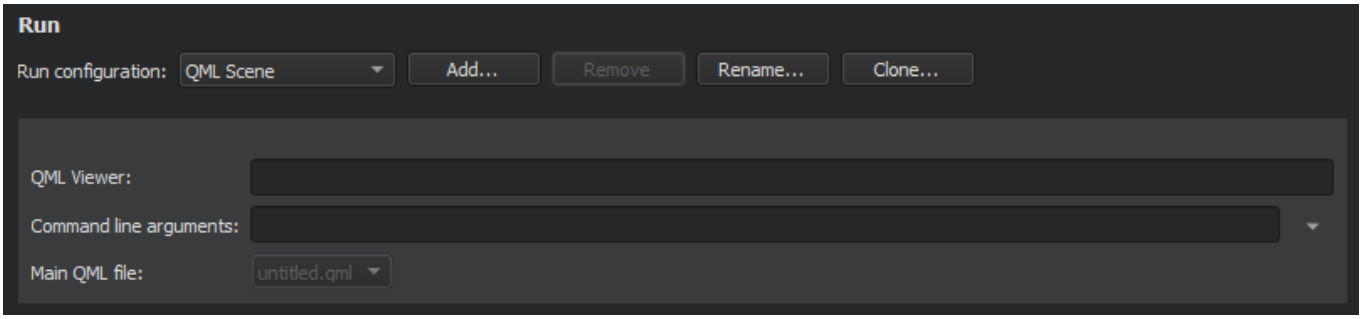
If you use CMake, Meson or the generic project type in Qt Creator, or want to run a custom desktop executable, create a **Custom Executable** run configuration for your project. For example, when working on a library, you can run a test application that links against the library.

Specify the executable to run, command line arguments, working directory, and environment variables to use.



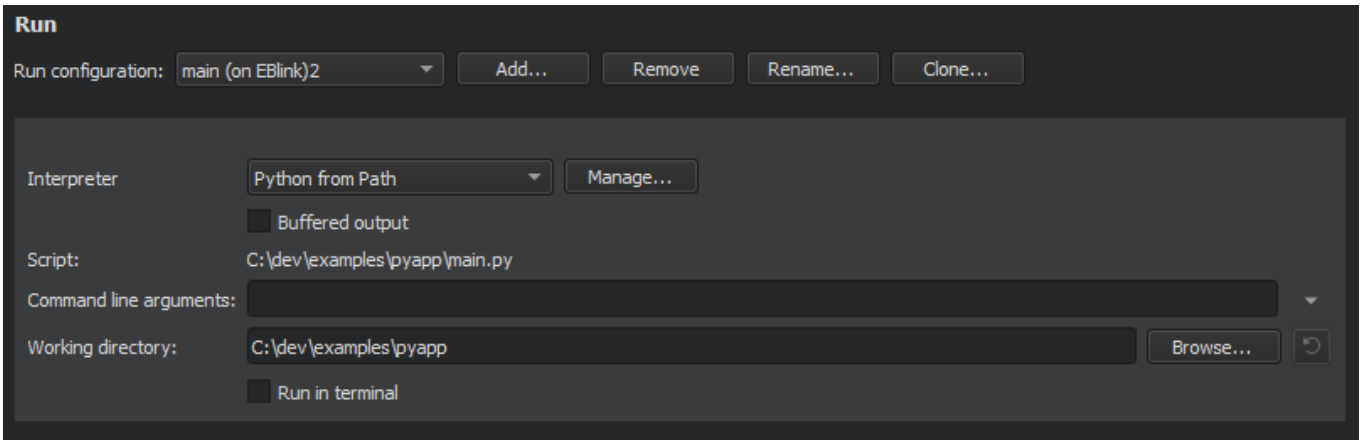
Specifying Run Settings for Qt Quick UI Projects

- In the **QML Viewer** field, specify the Qt QML Viewer to use.
- In the **Command line arguments** field, specify arguments to be passed to the executable.
- In the **Main QML file**, select the file that Qt QML Viewer will be started with.



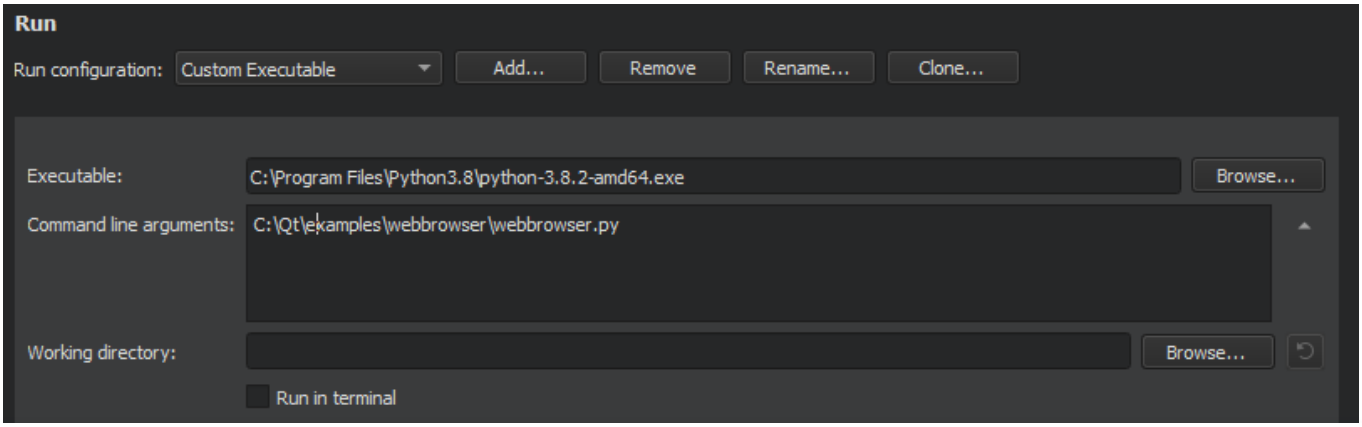
Specifying Run Settings for Python Projects

You can specify settings for running Qt for Python applications:



- In the **Interpreter** field, specify the path to the Python executable.
- Select the **Buffered output** check box to buffer the output. This improves output performance, but causes delays in output.
- In the **Script** field, you can see the path to the main file of the project that will be run.
- In the **Command line arguments** field, specify command line arguments to be passed to the executable.

If you want to run some other Python file than , create a custom executable run configuration:`main.py`



2. In the **Executable** field, specify the path to the Python executable.
3. In the **Command line arguments** field, select the Python file to run.

‹ Specifying Build Settings

Specifying Editor Settings ›

© 2022 The Qt Company Ltd. Documentation contributions included herein are the copyrights of their respective owners. The documentation provided herein is licensed under the terms of the GNU Free Documentation License version 1.3 as published by the Free Software Foundation. Qt and respective logos are trademarks of The Qt Company Ltd in Finland and/or other countries worldwide. All other trademarks are property of their respective owners.

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