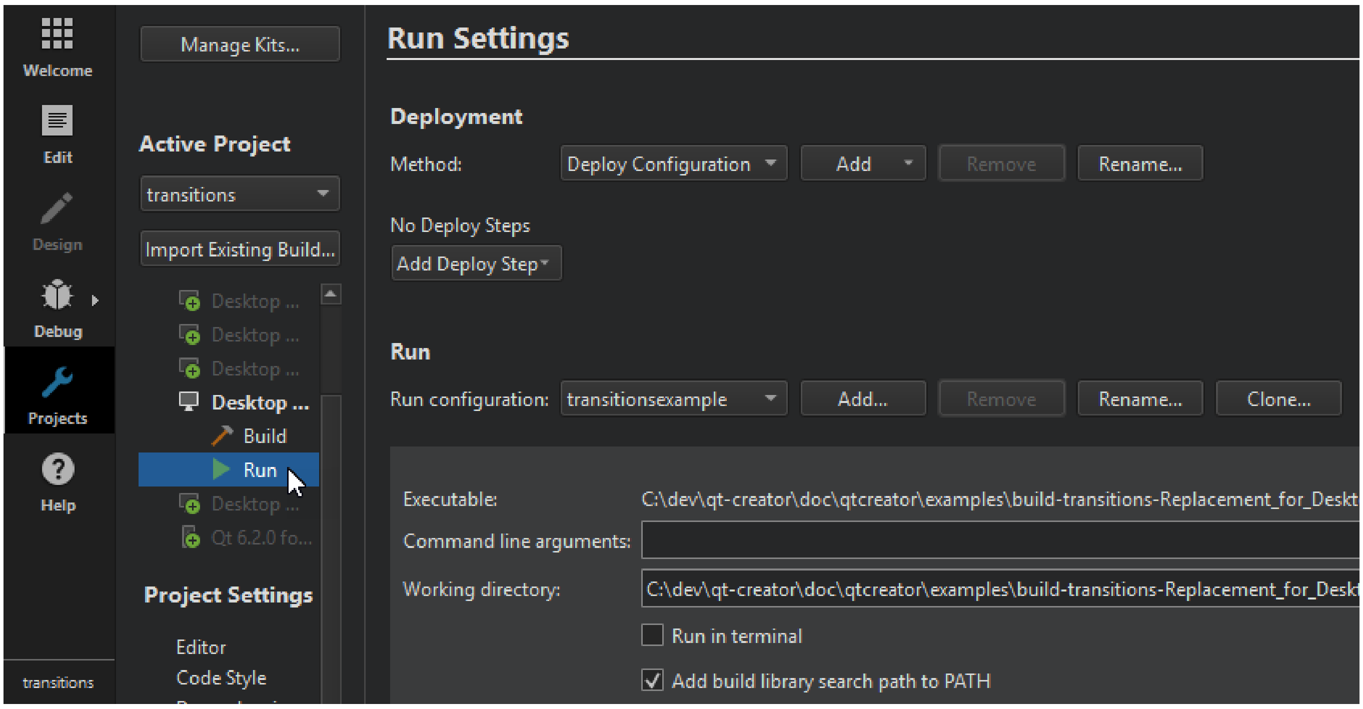
# Specifying Run Settings

The run settings to specify depend on the type of the project and on the kit that you build and run the project with.

Qt Creator automatically creates run configurations for your project. To view and modify them, select **Projects** > **Build & Run** > **Run**.



Method: Deploy Configuration

No Deploy Steps

Run configuration: transitionsexample

Executable: C:\dev\qt-creator\doc\qtcreator\examples\build-transitions-Replacement_for_Deskt

Command line arguments:

Working directory: C:\dev\qt-creator\doc\qtcreator\examples\build-transitions-Replacement_for_Deskt

☐ Run in terminal

☑ Add build library search path to PATH

To prevent Qt Creator from automatically creating run configurations, select **Edit** > **Preferences** > **Build & Run**, and then deselect the **Create suitable run configurations automatically** check box.

## Managing Run Configurations

The available run configurations are listed in the **Run configuration** field. To add run configurations for a project, select **Add**. To add a run configuration that is based on the current one, select **Clone**.

To rename the current run configuration, select **Rename**.

To remove the current run configuration, select **Remove**.

The run configurations for qmake projects derive their executable from the parsed .pro files. For more information on how the commands are constructed, see Starting External Processes.

If a project has multiple executables, you need to tell Qt Creator which one it should run.

### CMake Run Targets

When using CMake, you can filter the run target list by setting `qtc_runnable` as the value of the `FOLDER` property in the `CMakeLists.txt` file for the project. For example:

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

If you do not specify `qtc_runnable` for any project, Qt Creator automatically adds run configurations for all targets specified in `CMakeLists.txt`.

### qmake Run Targets

When using qmake, you can prevent Qt Creator from automatically creating run configurations for subprojects by specifying the `qtc_runnable` variable in the .pro files of the application projects (`TEMPLATE=app`) that you want to run. For example

```
CONFIG += qtc_runnable
```

If none of your application projects specifies `qtc_runnable`, Qt Creator creates run configurations for all application projects.

If any of your application projects specifies `qtc_runnable`, Qt Creator creates run configurations only for subprojects that also have `CONFIG += qtc_runnable` set in their .pro files.

For more information about qmake project templates, see TEMPLATE.

### Meson Run Targets

Qt Creator automatically adds run configurations for all targets declared with `executable()` function in Meson build descriptions.

## Specifying Run Settings for Desktop Device Types

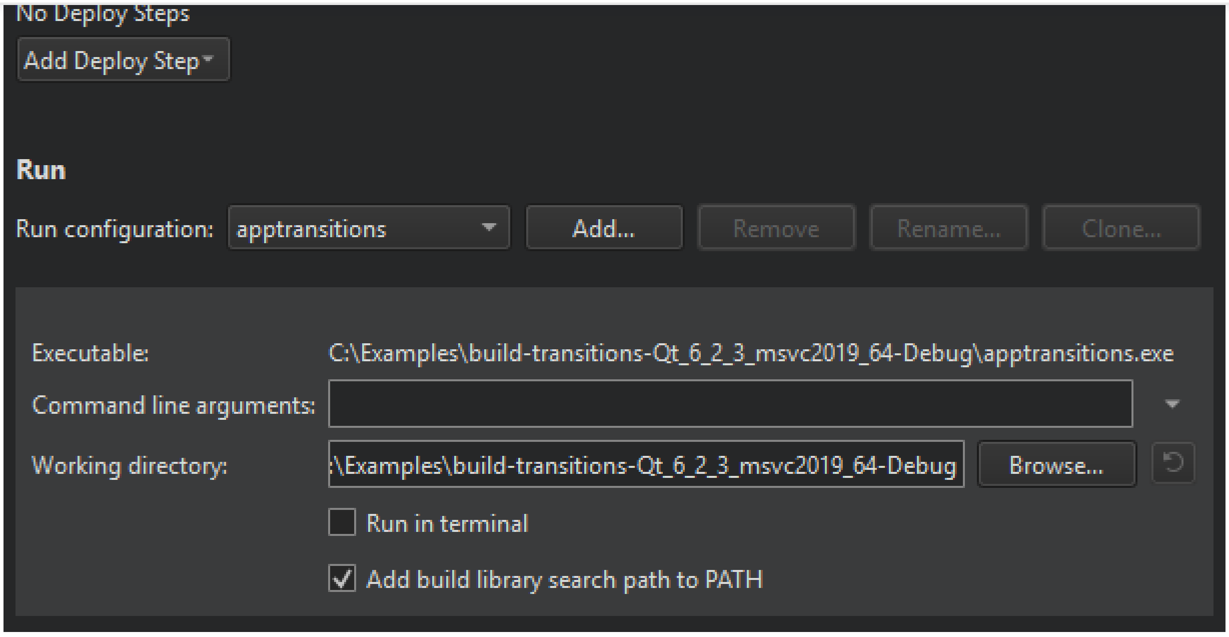
You can specify command line arguments to be passed to the executable and the working directory to use. The working directory defaults to the directory of the build result.

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.

Run Settings

Deployment


No Deploy Steps

Add Deploy Step

**Run**

Run configuration: apptransitions

Executable: C:\Examples\build-transitions-Qt_6_2_3_msvc2019_64-Debug\apptransitions.exe

Command line arguments:

Working directory: :\Examples\build-transitions-Qt_6_2_3_msvc2019_64-Debug

☐ Run in terminal

☑ Add build library search path to PATH

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 `make install`, and want to make sure that the deployed application will find the libraries also when it is run without Qt Creator.

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.

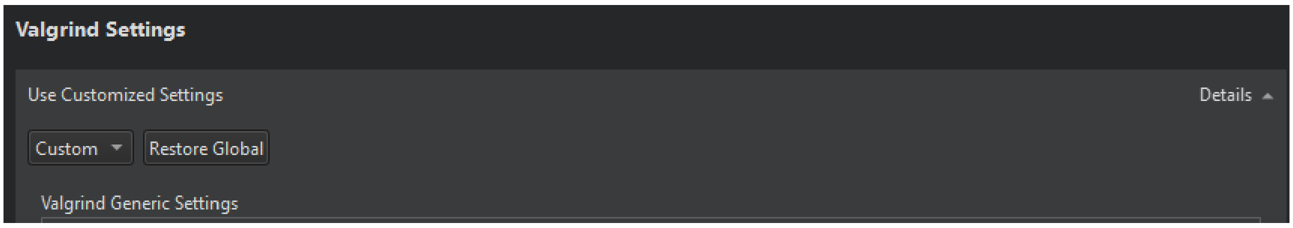
## Specifying Valgrind Settings

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.


**Valgrind Settings**

Use Customized Settings

Custom

Valgrind Generic Settings

Detect self-modifying code: Only on Stack

3. In **Valgrind executable**, specify the path to the Valgrind executable.
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.

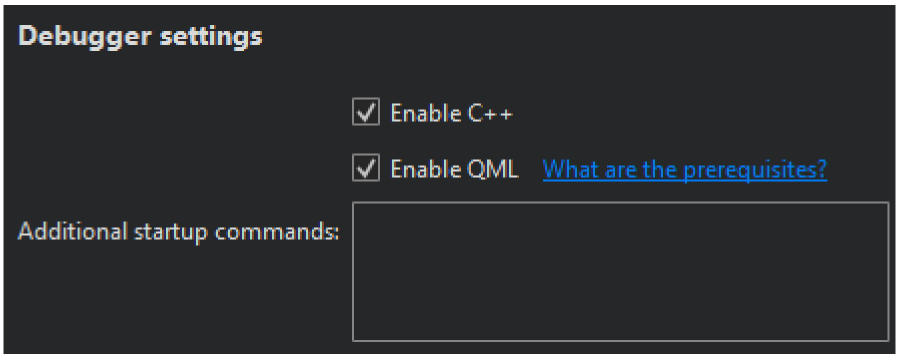
For more information about the CallGrind and MemCheck settings, see:

- 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


Debugger settings

[✓] Enable C++

[✓] Enable QML What are the prerequisites?

Additional startup commands:

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.

the added options, the application might not start.

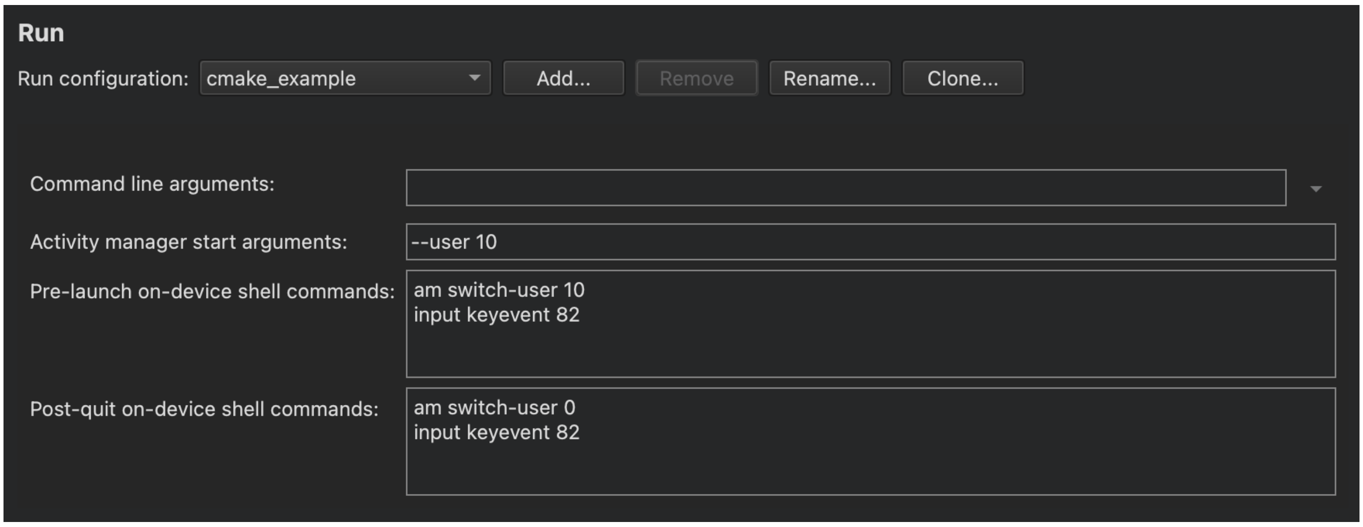
The default arguments for the Activity manager for a normal run:

```
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 `--user 10`, where 10 is the user ID of the user account.



Run configuration: cmake_example
Command line arguments:
Activity manager start arguments: --user 10
Pre-launch on-device shell commands:
```
am switch-user 10
input keyevent 82
```
Post-quit on-device shell commands:
```
am switch-user 0
input keyevent 82
```

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 10 on the device before running the application:

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

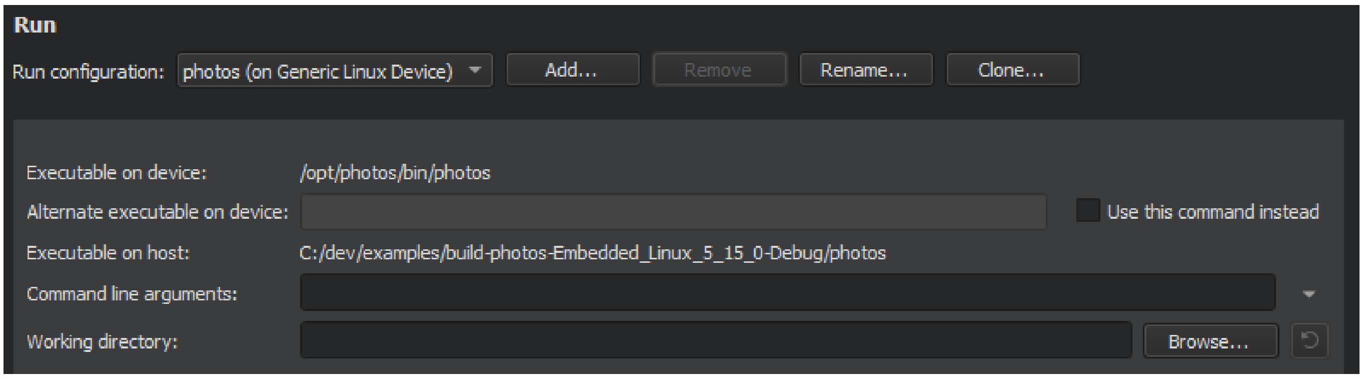
Enter the following commands into **Post-quit on-device shell commands** to switch back to the default user, 0, and to unlock the screen after the application is quit:

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

## Specifying Run Settings for Linux-Based Devices

When you run the application, Qt Creator copies the files to the connected device.

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


Run
Run configuration: photos (on Generic Linux Device)
Executable on device: /opt/photos/bin/photos
Alternate executable on device:
Use this command instead
Executable on host: C:/dev/examples/build-photos-Embedded_Linux_5_15_0-Debug/photos
Command line arguments:
Working directory:

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.

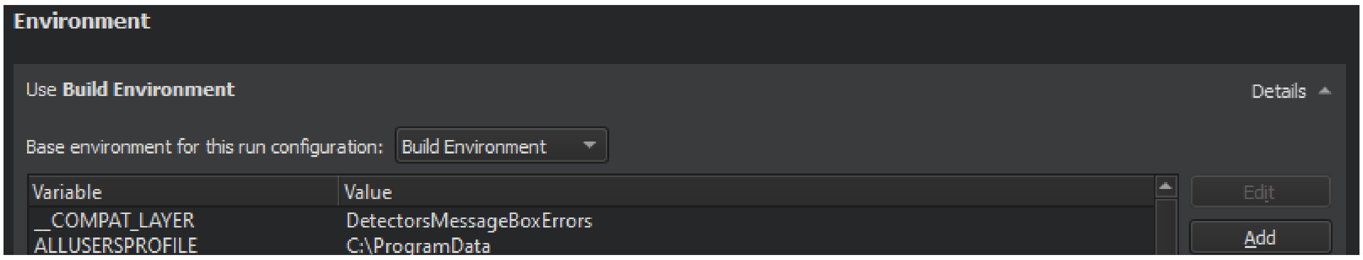
The run settings display the path to the executable file on the development host and on the device.

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

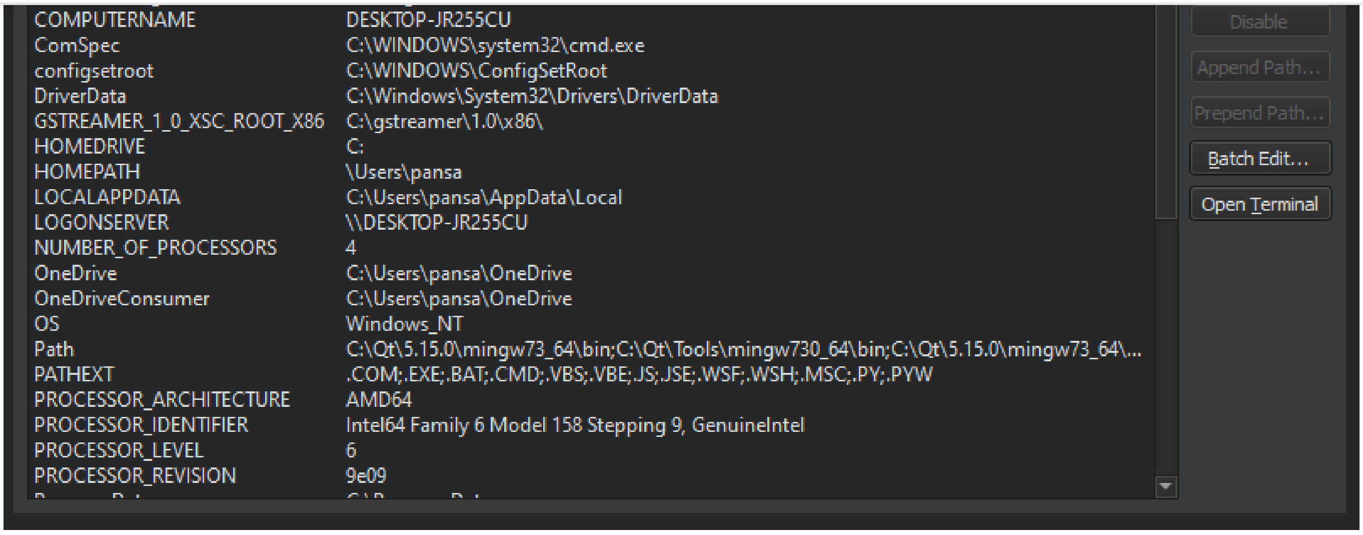
## Selecting the Run Environment

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.


Environment
Use **Build Environment**
Base environment for this run configuration: Build Environment

| Variable | Value |
|---|---|
| __COMPAT_LAYER | DetectorsMessageBoxErrors |
| ALLUSERSPROFILE | C:\ProgramData |



| | |
|---|---|
| COMPUTERNAME | DESKTOP-JR255CU |
| ComSpec | C:\WINDOWS\system32\cmd.exe |
| configsetroot | C:\WINDOWS\ConfigSetRoot |
| DriverData | C:\Windows\System32\Drivers\DriverData |
| GSTREAMER_1_0_XSC_ROOT_X86 | C:\gstreamer\1.0\x86\ |
| HOMEDRIVE | C: |
| HOMEPATH | \Users\pansa |
| LOCALAPPDATA | C:\Users\pansa\AppData\Local |
| LOGONSERVER | \\DESKTOP-JR255CU |
| NUMBER_OF_PROCESSORS | 4 |
| OneDrive | C:\Users\pansa\OneDrive |
| OneDriveConsumer | C:\Users\pansa\OneDrive |
| OS | Windows_NT |
| Path | C:\Qt\5.15.0\mingw73_64\bin;C:\Qt\Tools\mingw730_64\bin;C:\Qt\5.15.0\mingw73_64\... |
| PATHEXT | .COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC;.PY;.PYW |
| PROCESSOR_ARCHITECTURE | AMD64 |
| PROCESSOR_IDENTIFIER | Intel64 Family 6 Model 158 Stepping 9, GenuineIntel |
| PROCESSOR_LEVEL | 6 |
| PROCESSOR_REVISION | 9e09 |

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**.

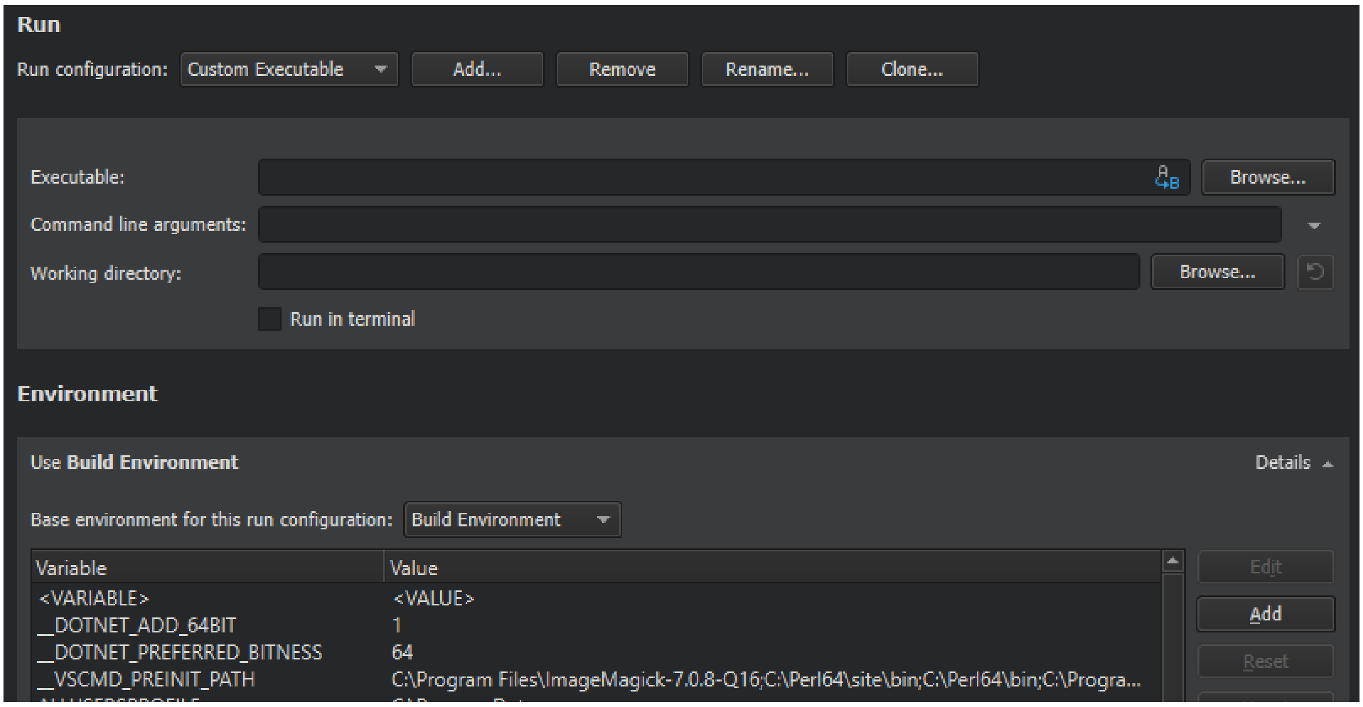
When running on a mobile device connected to the development host, Qt Creator fetches information about the **Device Environment** from the device. Usually, it does not make sense to edit the device environment.

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.



**Run**

Run configuration: Custom Executable

Executable:

Command line arguments:

Working directory:

Run in terminal

**Environment**

Use **Build Environment**

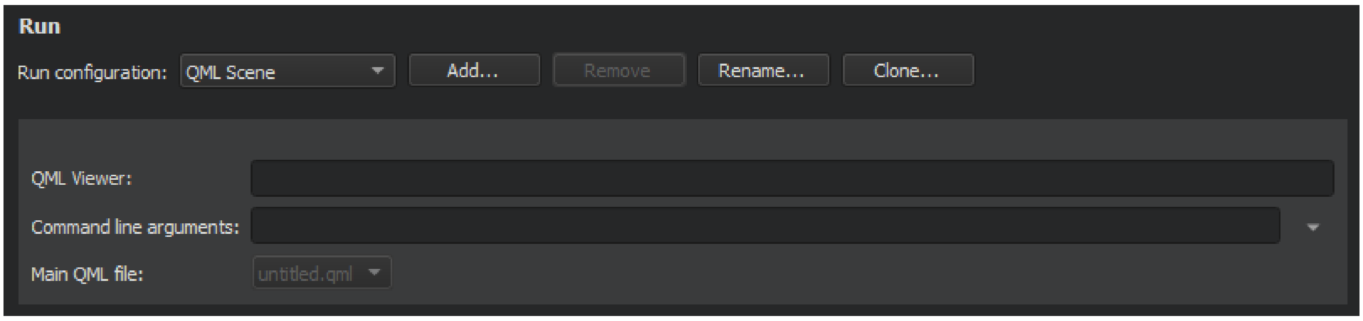
Base environment for this run configuration: Build Environment

| Variable | Value |
|---|---|
| <VARIABLE> | <VALUE> |
| __DOTNET_ADD_64BIT | 1 |
| __DOTNET_PREFERRED_BITNESS | 64 |
| __VSCMD_PREINIT_PATH | C:\Program Files\ImageMagick-7.0.8-Q16;C:\Perl64\site\bin;C:\Perl64\bin;C:\Progra... |

## Specifying Run Settings for Qt Quick UI Projects

You can specify run settings for kits with **Desktop** device type:

- 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.



Run configuration: QML Scene

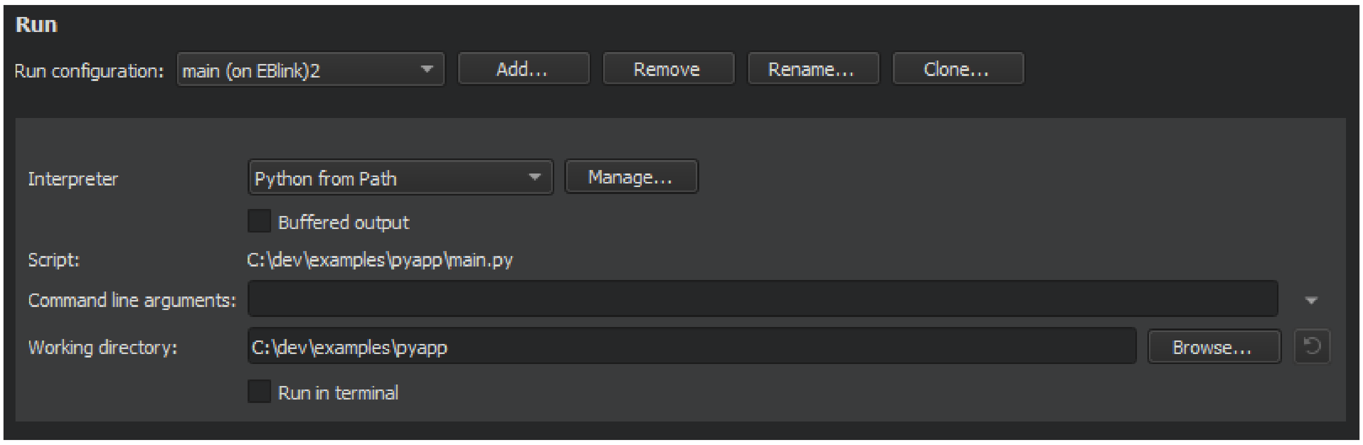
QML Viewer:

Command line arguments:

Main QML file: untitled.qml

## Specifying Run Settings for Python Projects

You can specify settings for running Qt for Python applications:



Run configuration: main (on EBlink)2

Interpreter: Python from Path

Buffered output

Script: C:\dev\examples\pyapp\main.py

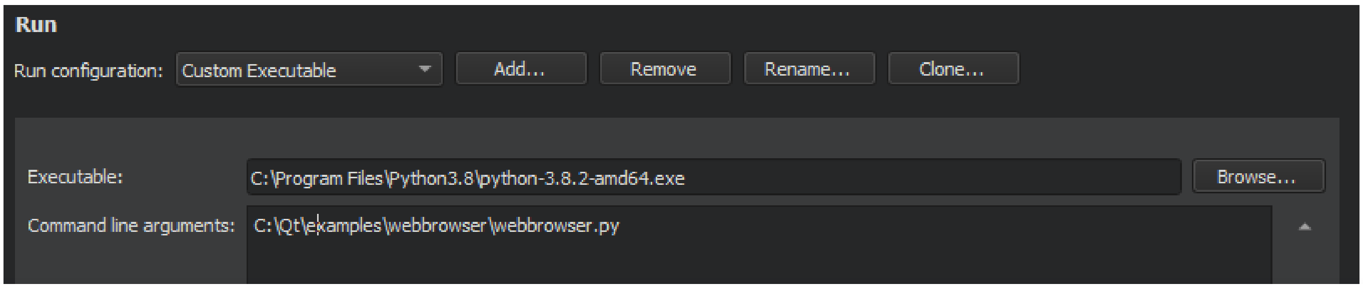
Command line arguments:

Working directory: C:\dev\examples\pyapp

Run in terminal

- 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 `main.py`, create a custom executable run configuration:



Run configuration: Custom Executable

Executable: C:\Program Files\Python3.8\python-3.8.2-amd64.exe

Command line arguments: C:\Qt\examples\webbrowser\webbrowser.py

1. Select **Add** > **Custom Executable**.
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 ›

Qt DOCUMENTATION

Feedback Sign In