

Entwicklungsumgebung



Diese Anleitung gilt für die Entwicklung ab BPS 2.24.1.0

Um eine Entwicklungsumgebung einzurichten sollten sie vorgängig eine virtuelle Maschine vorbereitet haben wie es in [VM für Entwicklung](#) beschrieben.

Beachten sie, dass nicht alle Anwendungen während der Einrichtung sofort funktionieren werden, da erforderliche DLL's unter Umständen noch nicht gefunden werden. Die erforderlichen Pfade werden erst im Schritt [Umgebungsvariablen](#) definitiv eingerichtet, bitte gedulden sie sich bis dann.

Visual Studio 2019

- Laden Sie Visual Studio Community 2019 herunter
- Wählen sie «Desktopentwicklung mit C++»
 - Unter «Optional»:
 - MSVC v142 - VS 2019 C++ x64/x86 Buildtools (neuste)
 - Windows 10 SDK (10.0.19041-0)
 - C++ CMake Tools für Windows

Installation der Debugger

Nachdem die obigen Komponenten installiert sind können die Debugger im SDK installiert werden:

- Systemsteuerung > Programme > Programme und Features
- Suchen nach «Windows Software Development Kit - Windows 10.0.19041.658»
- «Ändern»
- Nochmals «Change», dann «Next»
- «Debugging Tools for Windows» ankreuzen
- Button «Change»

Visual Studio IDE testweise starten und wieder beenden. (optional)

Perl

Homepage: <http://strawberryperl.com/>

- 64 bit Ausgabe wählen
- strawberry-perl-5.30.2.1-64bit.msi ausführen
- Installation nach: C:\dev\perl

Python

- Binärversion wählen auf: <http://www.python.org>
- Download «Windows x86-64 executable installer»
- python-3.8.5-amd64.exe ausführen
- Unten Abwählen:
 - Install launcher for all users
 - Add Python 3.8 to PATH
- Customize installation
- Optional Features
 - Alles abwählen
- Advanced Options
 - Alles abwählen
- Customize install location: C:\dev\python

Clang

- Holen sie die vorkompilierte Binärversion für Windows 64-bit:
https://download.qt.io/development_releases/prebuilt/libclang/
- Rechtsklick auf libclang-release_100-based-windows-vs2019_64.7z
 - 7-Zip
 - Dateien entpacken...
 - Entpacken nach: C:\dev\ (Dateinamen darunter abwählen)
- Die Verzeichnisse bin, include,... sollten danach in C:\dev\libclang sein

Jom

Replacement for nmake, compiles on multiple CPU cores simultaneously.

Homepage: <http://wiki.qt.io/Jom>

Download latest binary release.

- Rechtsklick auf jom_1_1_3.zip
 - 7-Zip
 - Dateien entpacken...
 - Entpacken nach: C:\dev\jom (Dateinamen darunter ggf. abwählen)

NASM

Net-wide Assembler. (Zum Bauen von OpenSSL benötigt)

Homepage: <http://www.nasm.us/>

- Rechtsklick auf nasm-2.14.02-win64.zip
 - 7-Zip

- Dateien entpacken...
- Entpacken nach: C:\dev\ (Dateinamen darunter ggf. abwählen)
- Umbenennen von C:\dev\nasm-2.14.02 nach C:\dev\nasm

OpenSSL

Homepage: <http://www.openssl.org/>

- Neustes Tar-Archiv herunterladen

Quellcode vorbereiten:

- Rechtsklick auf openssl-1.1.1g.tar.gz
 - 7-Zip
 - Hier entpacken
- Rechtsklick auf openssl-1.1.1g.tar
 - 7-Zip
 - Dateien entpacken...
 - Entpacken nach: C:\dev\ (Dateinamen darunter ggf. abwählen)
- Umbenennen von C:\dev\openssl-1.1.1g nach C:\dev\openssl.src

Bauen:

- Start / Visual Studio 2019 / x64 Native Command Prompt **ALS ADMINISTRATOR AUSFÜHREN**
- cd \dev\openssl.src
- PATH=C:\dev\nasm;C:\dev\perl\perl\bin;%PATH%
- perl Configure VC-WIN64A --prefix=c:\dev\openssl --openssldir=c:\dev\openssl
- nmake
- nmake test
- nmake install
- nmake clean
- exit

Qt

Quellcode beschaffen und vorbereiten

Quellcode holen:

- Download zip Quellen von http://download.qt.io/official_releases/qt/5.15/5.15.2/single/
- Rechtsklick auf qt-everywhere-src-5.15.2.zip
 - 7-Zip
 - Dateien entpacken...
 - Entpacken nach: C:\dev\ (Dateinamen darunter ggf. abwählen)
- Umbenennen von C:\dev\qt-everywhere-src-5.15.2 nach C:\dev\qt.src

Patches vornehmen:

- C:\dev\qt.src\qtbase\src\widgets\itemviews\qmainwindow.h
C:\dev\qt.src\qtbase\src\widgets\widgets\qcombobox.h
 - Funktion „addItem“ nach „public Q_SLOTS“ verschieben
- C:\dev\qt.src\qtbase\src\corelib\kernel\qvariant.h
 - Am Ende der Datei unmittelbar vor QT_END_NAMESPACE einfügen:

```
Q_CORE_EXPORT const QVariant::Handler* userVariantHandler();  
Q_CORE_EXPORT void setUserVariantHandler(const QVariant::Handler*  
aHandler);
```

- C:\dev\qt.src\qtbase\src\corelib\kernel\qvariant.cpp
 - Nach der Funktion QVariantPrivate::registerHandler (ca. Zeile 1591) einfügen:

```
Q_CORE_EXPORT const QVariant::Handler* userVariantHandler()  
{  
    return handlerManager[QMetaType::User];  
}  
  
Q_CORE_EXPORT void setUserVariantHandler(const QVariant::Handler*  
aHandler)  
{  
    handlerManager.registerHandler(QModulesPrivate::Unknown,  
aHandler);  
}
```

- C:\dev\qt.src\qtbase\src\network\socket\qlocalsocket_win.cpp
 - Anweisung in Zeile 144 anpassen:

```
if  
(d->serverName.startsWith(QLatin1String("\\\\" /*pipePath*/))
```

- C:\dev\qt.src\qtbase\src\printsupport\kernel\qprintengine_win.cpp
 - Anpassungen in Funktion QWin32PrintEnginePrivate::setPageSize(const QSize &pageSize) die ca. auf Zeile 1620 beginnt
 - Ersetzen: printerPageSize.isValid()
durch: pageSize.id() != QSize::Custom && printerPageSize.isValid()
 - Ca. in Zeile 1629:

```
const QSize usePageSize =  
pageSize.id() != QSize::Custom && printerPageSize.isValid()  
? printerPageSize : pageSize;
```

- Ca. in Zeile 1635:

```
if (pageSize.id() != QSize::Custom &&  
printerPageSize.isValid()) {
```

Qt Kommandozeile

Datei C:\dev\bat\qtbuildenv.bat erzeugen:

```
@echo off
call "C:\Program Files (x86)\Microsoft Visual
Studio\2019\Community\VC\Auxiliary\Build\vcvarsall.bat" x64
set INCLUDE=C:\dev\openssl\include;%INCLUDE%
set LIB=C:\dev\openssl\lib;%LIB%
set LLVM_INSTALL_DIR=C:\dev\libclang
PATH=C:\dev\libclang\bin;%PATH%
PATH=C:\dev\python;%PATH%
PATH=C:\dev\perl\perl\bin;%PATH%
PATH=C:\dev\nasm;%PATH%
PATH=C:\dev\jom;%PATH%
PATH=C:\dev\qt.src\gnuwin32\bin;%PATH%
PATH=C:\dev\qt.src\qtbase\bin;%PATH%
PATH=C:\dev\qt\qtbase\bin;%PATH%
cd /d C:\dev
```

Neue Verknüpfung *Qt Command Prompt* in C:\dev:

- Ziel: %COMSPEC% /k C:\dev\bat\qtbuildenv.bat

Qt bauen

Qt Command Prompt ausführen:

- mkdir \dev\qt
- cd \dev\qt
- ..\qt.src\configure -prefix C:\dev\qt\qtbase -nomake examples -nomake tests -skip qtwebengine -opensource -confirm-license
- jom

Wenn Qt5Script.dll/Qt5ScriptTools.dll in C:\dev\qt\qtbase\bin fehlen:

- jom module-qtscript

Dokumentation bauen:

- jom docs

Qt Solutions

Beschaffen und vorbereiten

- Quellcode von <https://github.com/qtproject/qt-solutions> holen
 - Klick auf *Clone or download*

- Download ZIP
- Verzeichnis anlegen: C:\dev\qtsol
- qt-solutions-master.zip öffnen und folgende Unterordner nach C:\dev\qtsol kopieren:
 - qtpropertybrowser
 - qtservice
 - qtsingleapplication

Patches:

- C:\dev\qtsol\qtpropertybrowser\common.pri:
QTPROPERTYBROWSER_LIBNAME = \$\$qtLibraryTarget(QtSolutions_PropertyBrowser-head)
QTPROPERTYBROWSER_LIBNAME = \$\$qtLibraryTarget(QtPropertyBrowser)
- C:\dev\qtsol\qtservice\common.pri:
QTSERVICE_LIBNAME = QtSolutions_Service-head
QTSERVICE_LIBNAME = QtService
- C:\dev\qtsol\qtsingleapplication\common.pri:
QTSINGLEAPPLICATION_LIBNAME = \$\$qt5LibraryTarget(QtSolutions_SingleApplication-head)
QTSINGLEAPPLICATION_LIBNAME = \$\$qt5LibraryTarget(Qt5SingleApplication)

QTSINGLEAPPLICATION_LIBNAME = \$\$qtLibraryTarget(QtSolutions_SingleApplication-head)
QTSINGLEAPPLICATION_LIBNAME = \$\$qtLibraryTarget(QtSingleApplication)
- C:\dev\qtsol\qtsingleapplication\src\qtsingleapplication.cpp:
- Includes im Kopfbereich ergänzen mit:

```
#if defined(Q_OS_WIN)
    #include <windows.h>
#endif
```

- Am Anfang der Funktion sysInit(...) einfügen:

```
#if defined(Q_OS_WIN)
    AllowSetForegroundWindow(ASFW_ANY);
#endif
```

- C:\dev\qtsol\qtsingleapplication\builddlib\builddlib.pro:
 - Unten anhängen:

```
win32: LIBS += -lUser32
```

Qt Solutions bauen

Qt Command Prompt ausführen:

- cd \dev\qtsol\qtpropertybrowser
 - configure -library
 - qmake
 - jom debug
 - jom release

- cd doc\html
- qhelpgenerator qtpropertybrowser.qhp
- cd \dev\qtsol\qt-service
 - configure -library
 - qmake
 - jom debug
 - jom release
 - cd doc\html
 - qhelpgenerator qt-service.qhp
- cd \dev\qtsol\qt-singleapplication
 - configure -library
 - qmake
 - jom debug
 - jom release
 - cd doc\html
 - qhelpgenerator qt-singleapplication.qhp

PostgreSQL

<http://www.postgresql.org/>

- Binaries im ZIP Archiv für Win X86-64 herunterladen
- Rechtsklick auf postgresql-13.2-2-windows-x64-binaries.zip
 - 7-Zip
 - Dateien entpacken...
 - Entpacken nach: C:\dev\ (Dateinamen darunter ggf. abwählen)

Oracle

<https://www.oracle.com/ch-de/database/technologies/instant-client/winx64-64-downloads.html>

- Instant Client Herunterladen:
 - Basic Light-Paket
 - SQL*Plus-Paket
 - Tools-Paket
 - SDK-Paket
- Ordner C:\dev\oracle anlegen
- Rechtsklick auf instantclient-basiclite-windows.x64-19.6.0.0.0dbru.zip
 - 7-Zip
 - Dateien entpacken...
 - Entpacken nach: C:\dev\oracle\ (Dateinamen darunter ggf. abwählen)
- Dito jeweils mit
 - instantclient-sqlplus-windows.x64-19.6.0.0.0dbru.zip
 - instantclient-tools-windows.x64-19.6.0.0.0dbru.zip
 - instantclient-sdk-windows.x64-19.6.0.0.0dbru.zip
- Vorhandenes tnsnames.ora für die eigenen Oracle Server kopieren nach C:\dev\oracle

Umgebungsvariablen

Folgende Verzeichnisse zu PATH in System-Umgebungsvariablen hinzufügen:

- C:\dev\openssl\bin
- C:\dev\libclang\bin
- C:\dev\qt\qtbase\bin
- C:\dev\oracle\instantclient_19_6
- C:\dev\pgsql\bin
- C:\dev\pgsql\lib
- C:\dev\bps\Release64\bin

Neue System-Umgebungsvariablen hinzufügen:

- QT_PLUGIN_PATH=C:\dev\qt\qtbase\plugins
- TNS_ADMIN=C:\dev\oracle

Qt Assistant Verwendung

C:\dev\qt\qtbase\bin\assistant.exe ausführen:

- An Taskleiste anheften
- Warten bis die Indexierung beim ersten Start fertig durchgelaufen ist, dann:
- *Edit - Preferences - Documentation - Add...*
 - C:\dev\qtsol\qtpropertybrowser\doc\html\qtpropertybrowser.qch
 - C:\dev\qtsol\qtservice\doc\html\qtservice.qch
 - C:\dev\qtsol\qtsingleapplication\doc\html\qtsingleapplication.qch

Qt Creator

- Opensource Quellcode holen von http://download.qt.io/official_releases/qtcreator/4.14/4.14.0/
- Rechtsklick auf qt-creator-opensource-src-4.14.0.zip
 - 7-Zip
 - Dateien entpacken...
 - Entpacken nach: C:\dev\ (Dateinamen darunter ggf. abwählen)
- Umbenennen von C:\dev\qt-creator-opensource-src-4.14.0 nach C:\dev\qtcre.src

Qt Command Prompt ausführen:

- mkdir c:\dev\qtcre
- cd c:\dev\qtcre
- c:\dev\qt\qtbase\bin\qmake.exe -r c:\dev\qtcre.src
- jom
- jom docs

C:\dev\qtcre64\bin\qtcreator.exe ausführen:

- An Taskleiste anheften

- Weiter mit Konfiguration unten

Konfiguration

Help - About Plugins

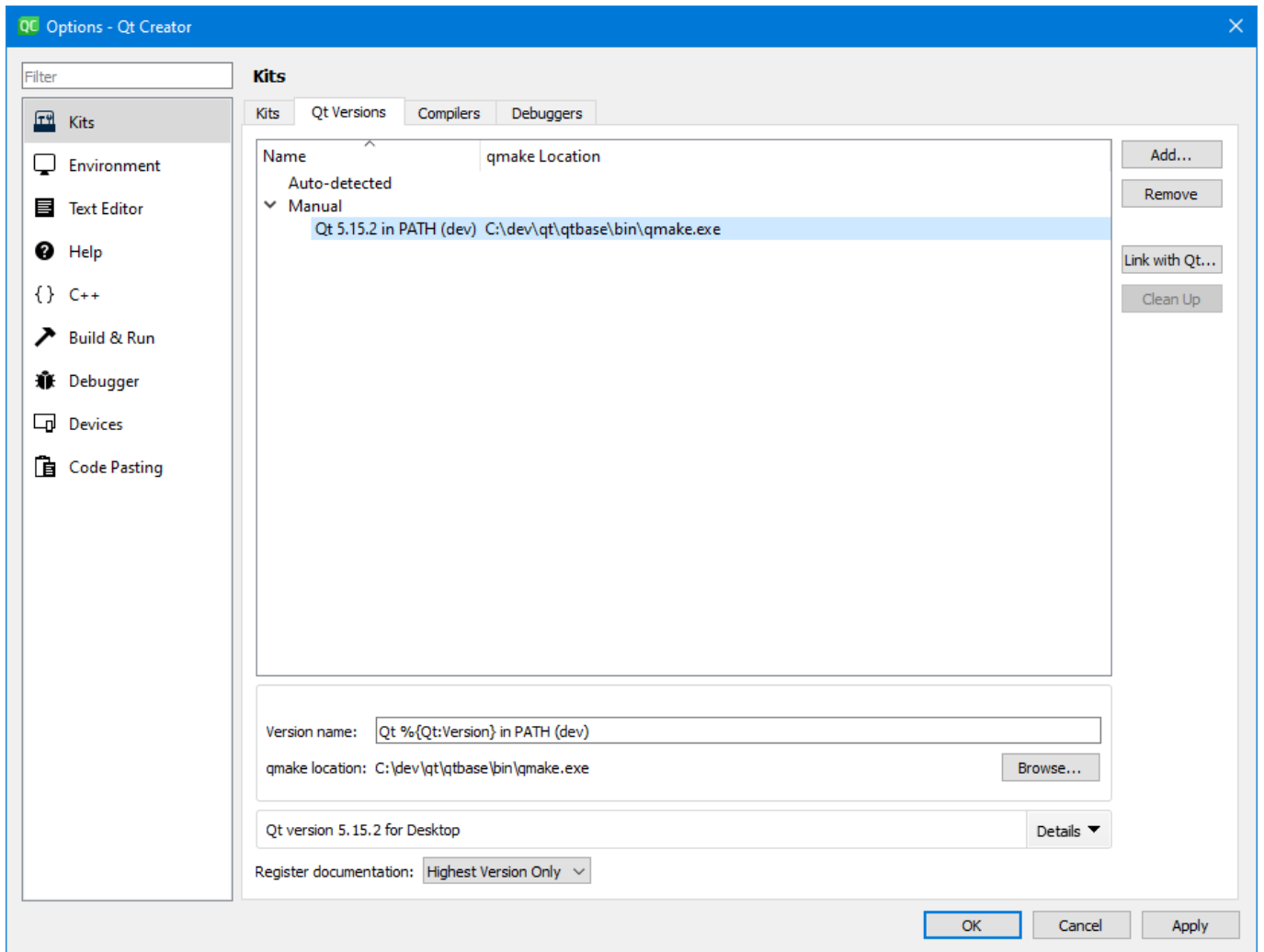
Empfohlen:

- Build Systems
 - QmakeProjectManager
 - QtSupport
- C++
 - ClassView
 - CppEditor
 - CppTools
- Code Analyzer
 - (nichts)
- Core
 - Bookmarks
 - Help
 - ProjectExplorer
 - TextEditor
 - Welcome
- Device Support
 - (nichts)
- Modeling
 - (nichts)
- Other Languages
 - (nichts)
- Qt Creator
 - BinEditor
 - Debugger
 - Designer
 - ResourceEditor
- Qt Quick
 - (nichts)
- Utilities
 - CodePaster
 - DiffEditor
 - ImageViewer
 - Macros
 - TaskList
- Version Control
 - (nichts)

Tools - Options

[Kits](#)

The screenshot shows the 'Options - Qt Creator' dialog box, specifically the 'Kits' tab. On the left is a sidebar with navigation icons for Kits, Environment, Text Editor, Help, C++, Build & Run, Debugger, Devices, and Code Pasting. The main area is titled 'Kits' and has sub-tabs for 'Kits', 'Qt Versions', 'Compilers', and 'Debuggers'. A tree view shows 'Auto-detected' and 'Manual' categories, with 'Desktop (default)' selected under 'Manual'. To the right of the tree are buttons: Add, Clone, Remove, Make Default, Settings Filter..., and Default Settings Filter... Below the tree is a form for configuring the selected kit. Fields include: Name (Desktop), File system name (empty), Device type (Desktop), Device (Local PC (default for Desktop)), Sysroot (empty), Compiler (C: Microsoft Visual C++ Compiler 16.9.31205.134 (amd64), C++: Microsoft Visual C++ Compiler 16.9.31205.134 (amd64)), Environment (No changes to apply, with a checkbox for 'Force UTF-8 MSVC compiler output'), Debugger (Auto-detected CDB at C:\Program Files (x86)\Windows Kits\10\Debuggers\x64\cdb.exe), Qt version (Qt 5.15.2 in PATH (dev)), and Qt mkspec (empty). Each field has a 'Manage...' button. At the bottom right are 'OK', 'Cancel', and 'Apply' buttons.



Options - Qt Creator

Filter

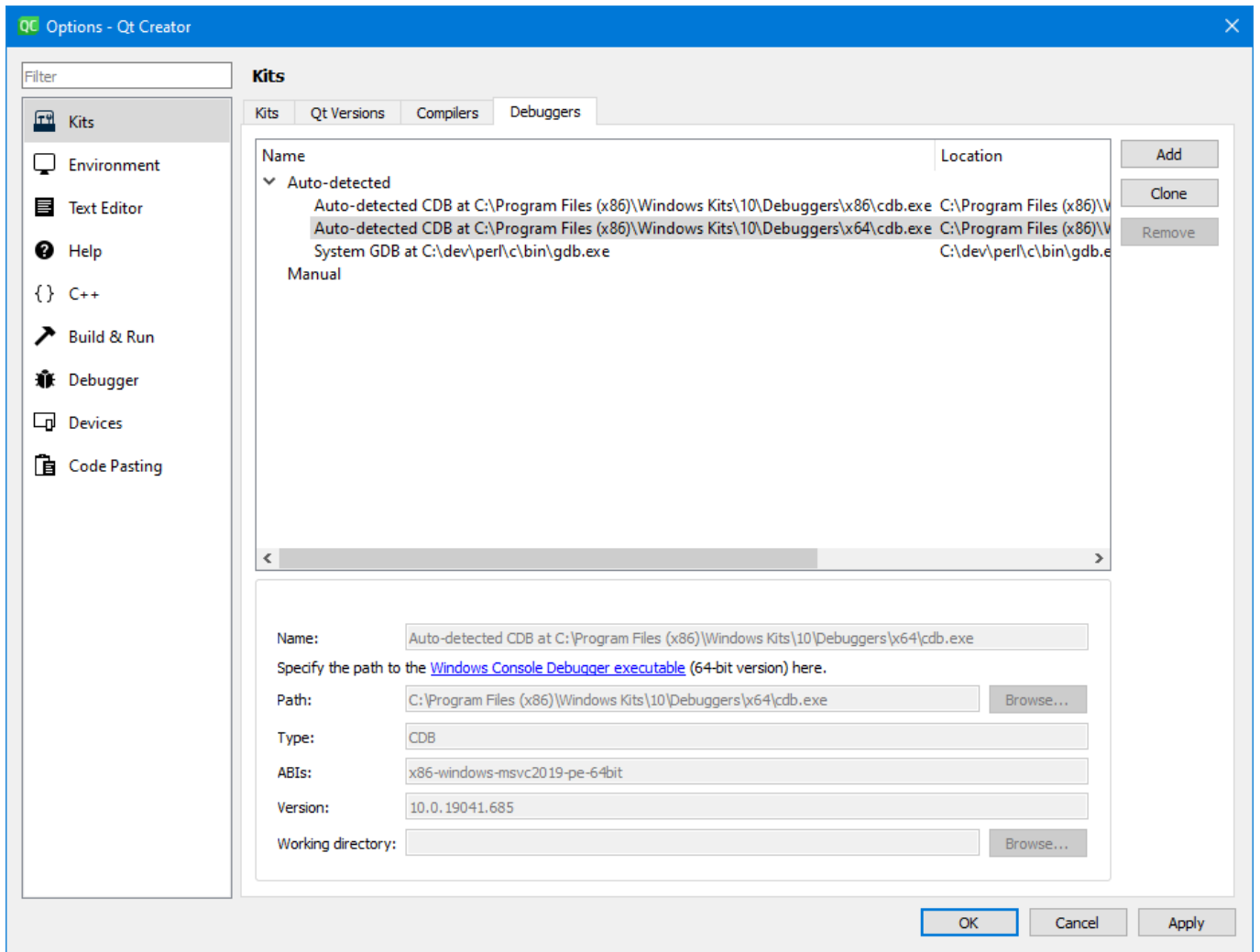
Kits

Kits Qt Versions Compilers Debuggers

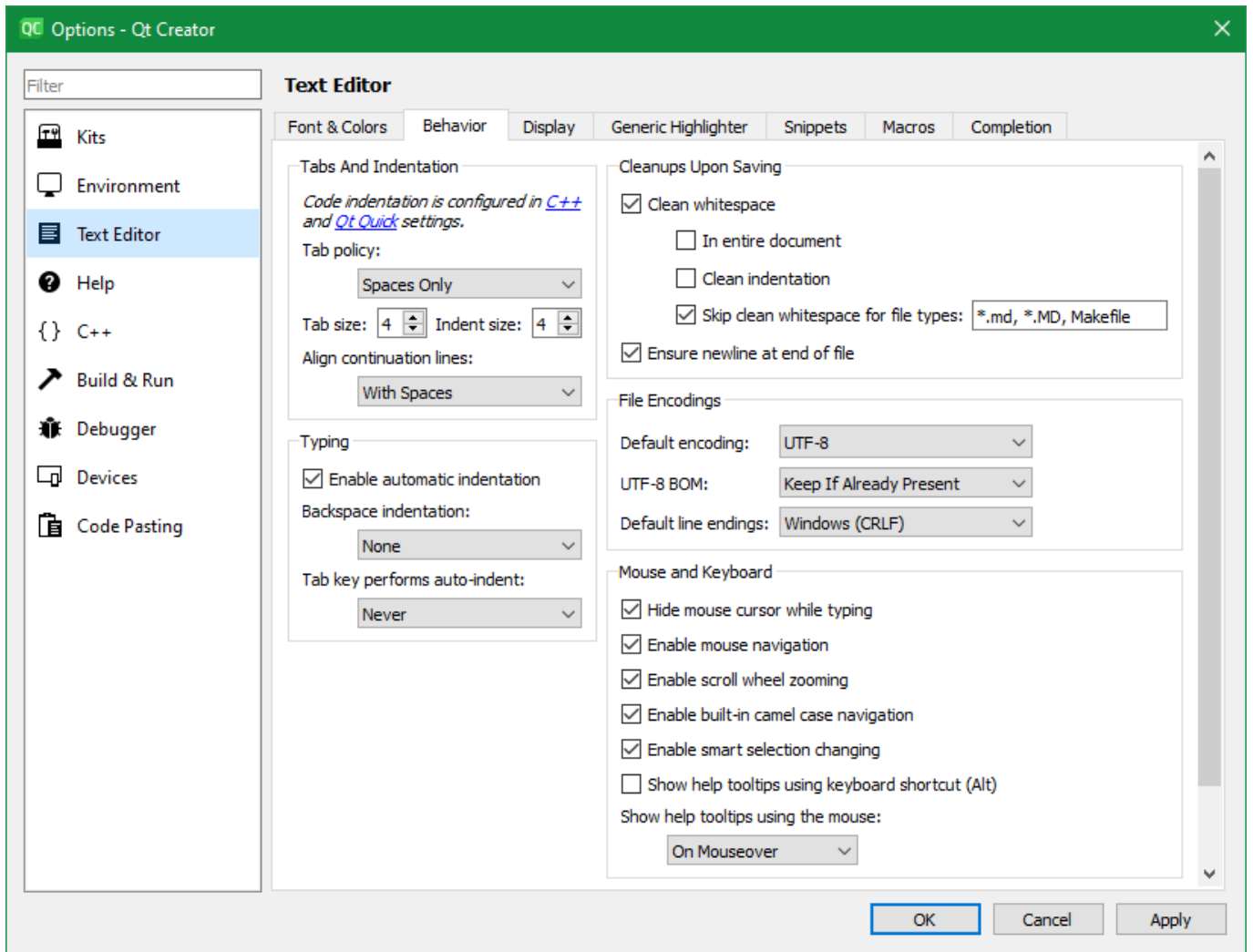
Name	Type
Auto-detected	
C	
Microsoft Visual C++ Compiler 16.9.31205.134 (x86)	MSVC
Microsoft Visual C++ Compiler 16.9.31205.134 (amd64_x86)	MSVC
Microsoft Visual C++ Compiler 16.9.31205.134 (amd64)	MSVC
Microsoft Visual C++ Compiler 16.9.31205.134 (x86_amd64)	MSVC
Default LLVM 64 bit based on MSVC2019	Clang
MinGW (C, x86 64bit in C:\dev\perl\c\bin)	MinGW
Clang (C, x86 64bit in C:\dev\libclang\bin)	Clang
C++	
Microsoft Visual C++ Compiler 16.9.31205.134 (x86)	MSVC
Microsoft Visual C++ Compiler 16.9.31205.134 (amd64_x86)	MSVC
Microsoft Visual C++ Compiler 16.9.31205.134 (amd64)	MSVC
Microsoft Visual C++ Compiler 16.9.31205.134 (x86_amd64)	MSVC
Default LLVM 64 bit based on MSVC2019	Clang
MinGW (C++, x86 64bit in C:\dev\perl\c\bin)	MinGW
Clang (C++, x86 64bit in C:\dev\libclang\bin)	Clang
Manual	
C	
C++	

Buttons: Add, Clone, Remove, Remove All, Re-detect, Auto-detection Settings...

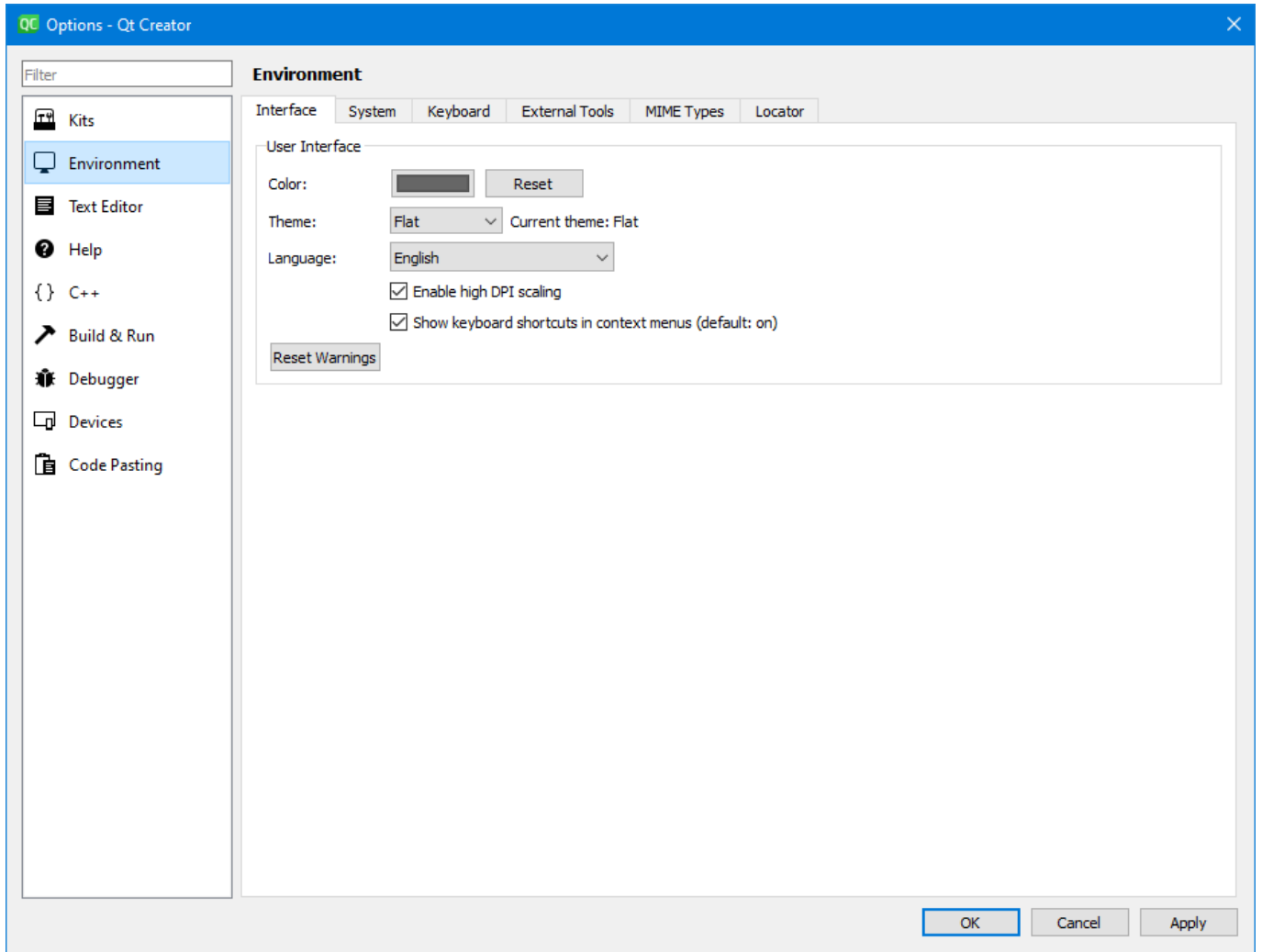
Buttons: OK, Cancel, Apply

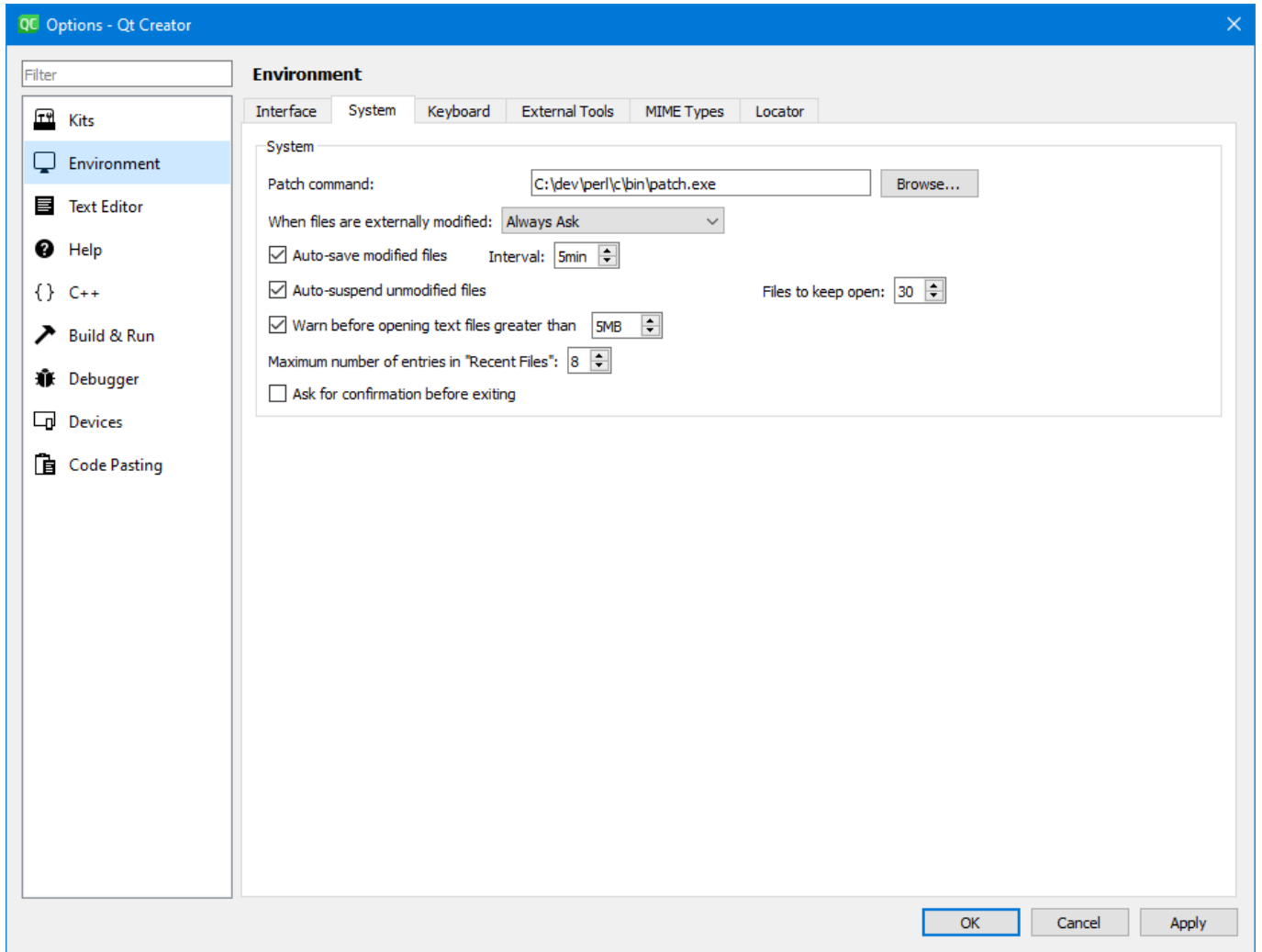


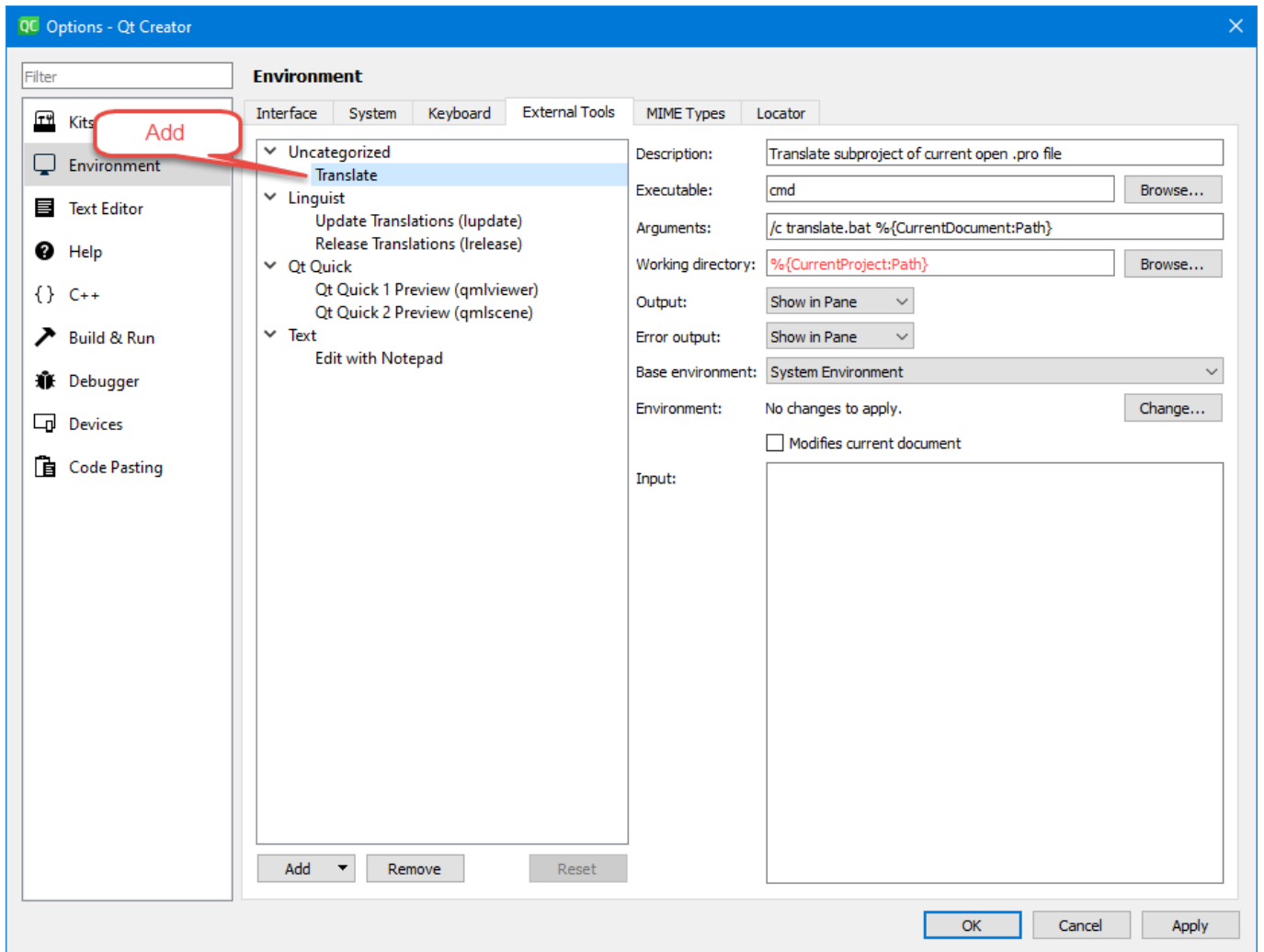
Text Editor

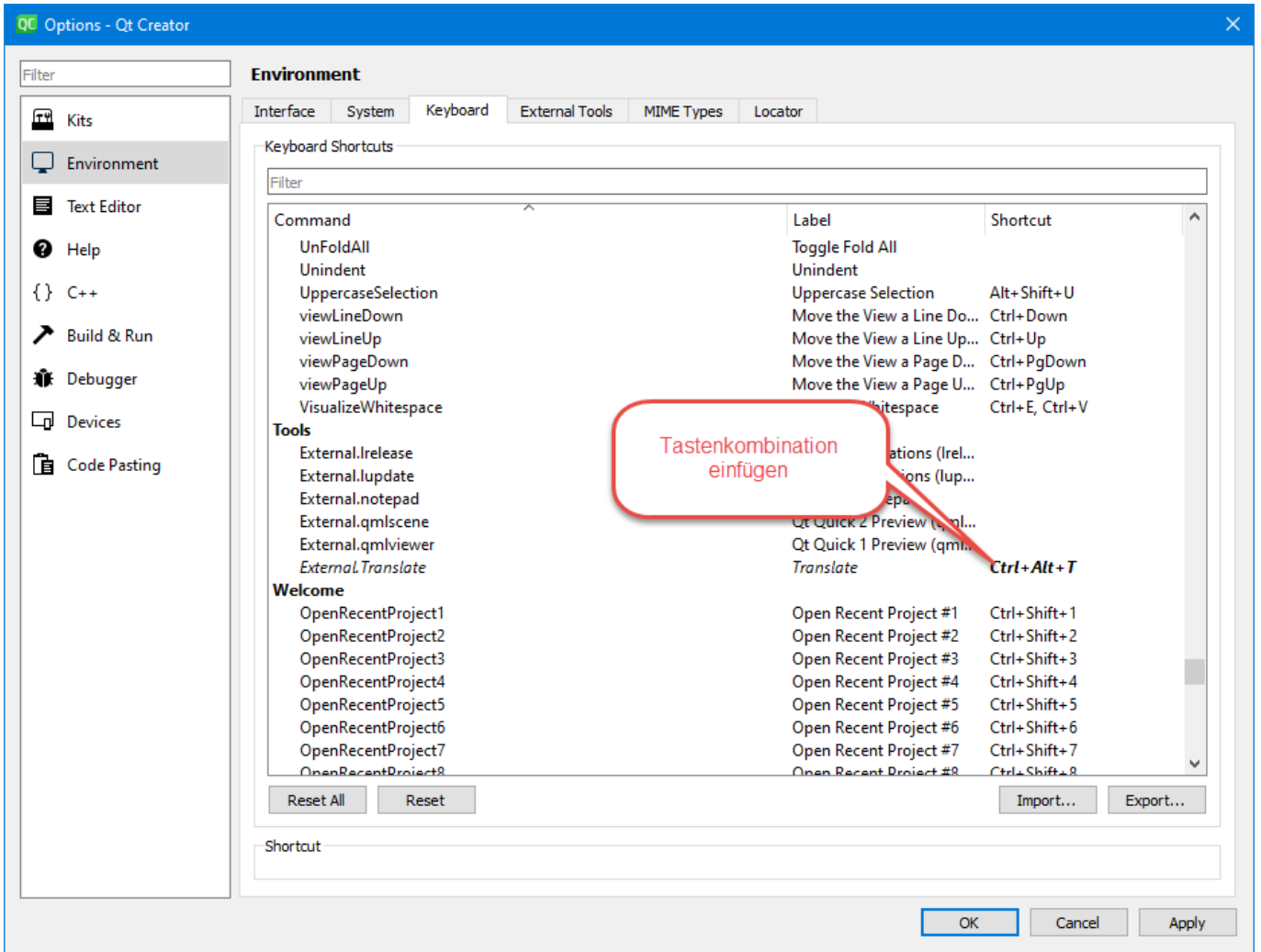


Environment

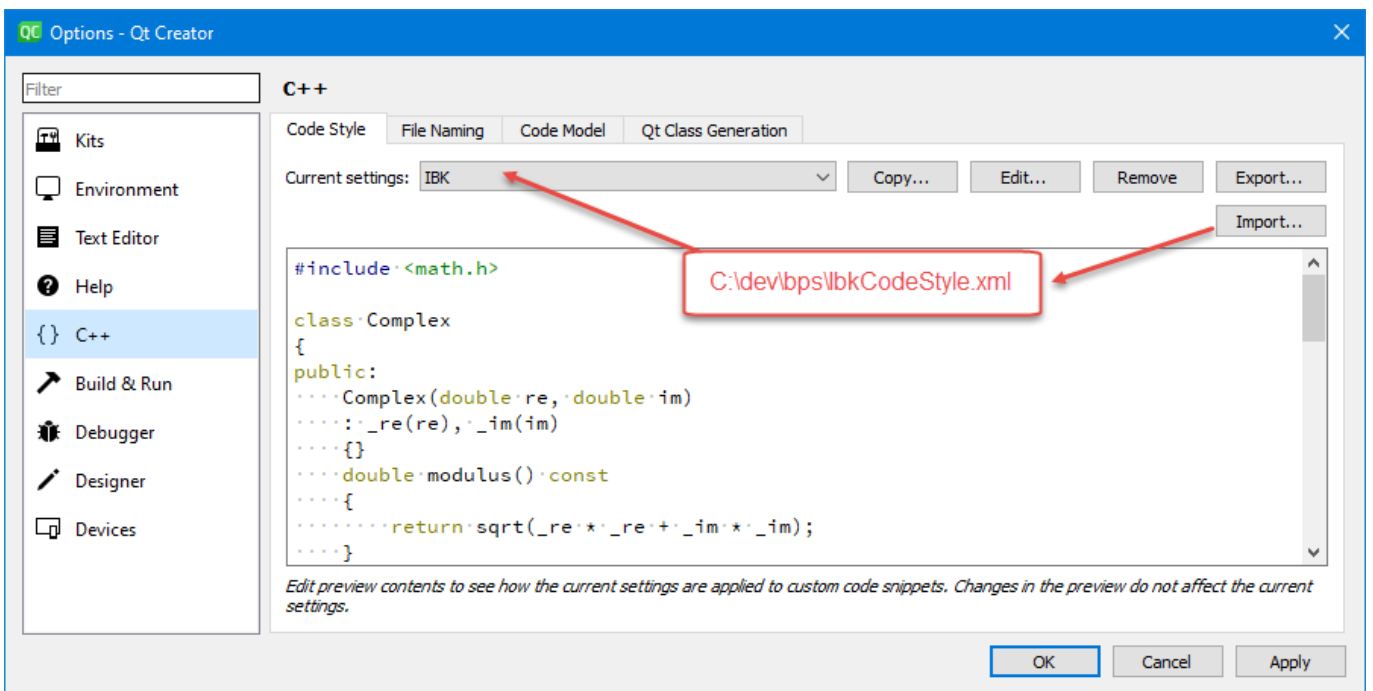






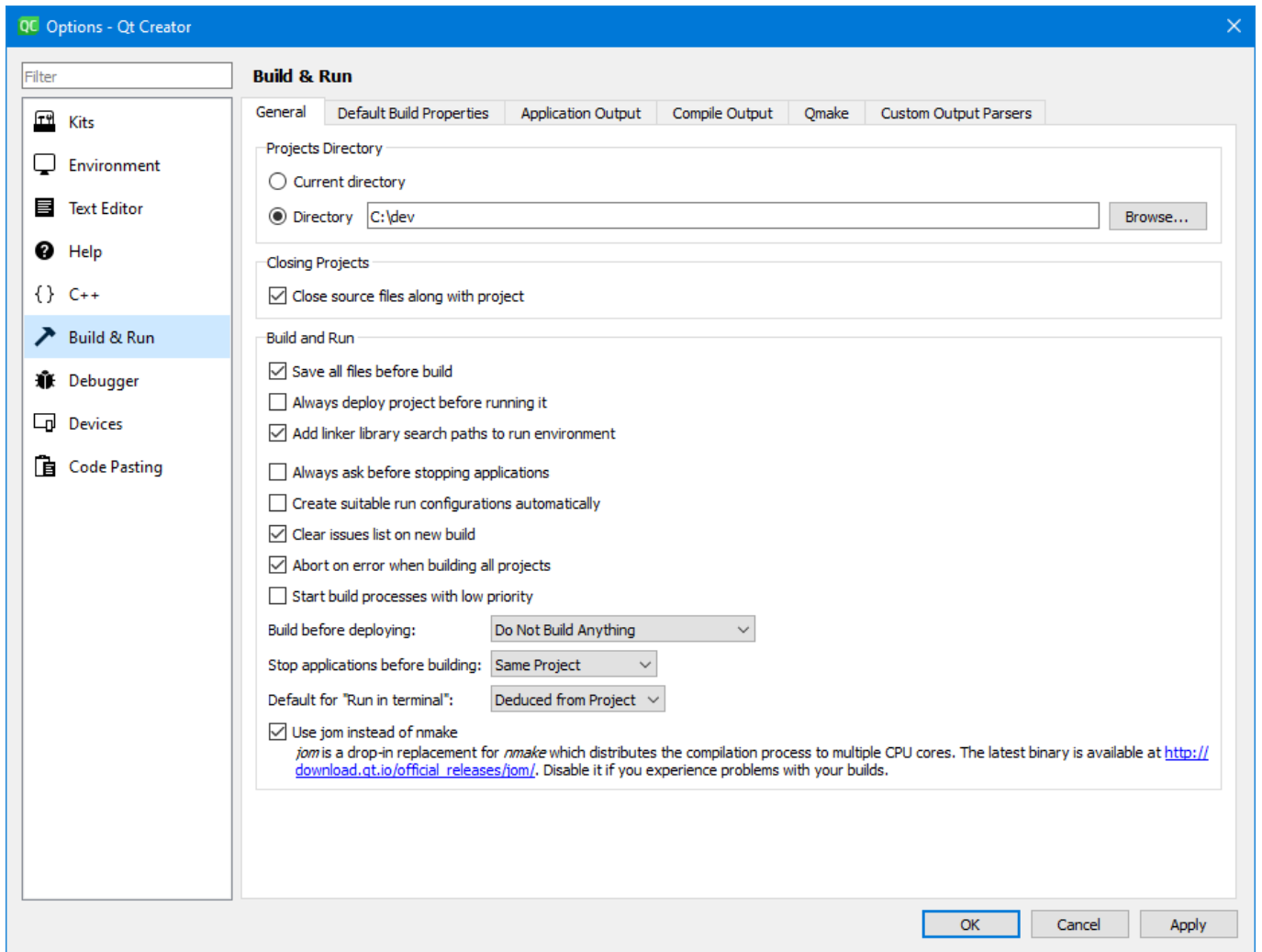


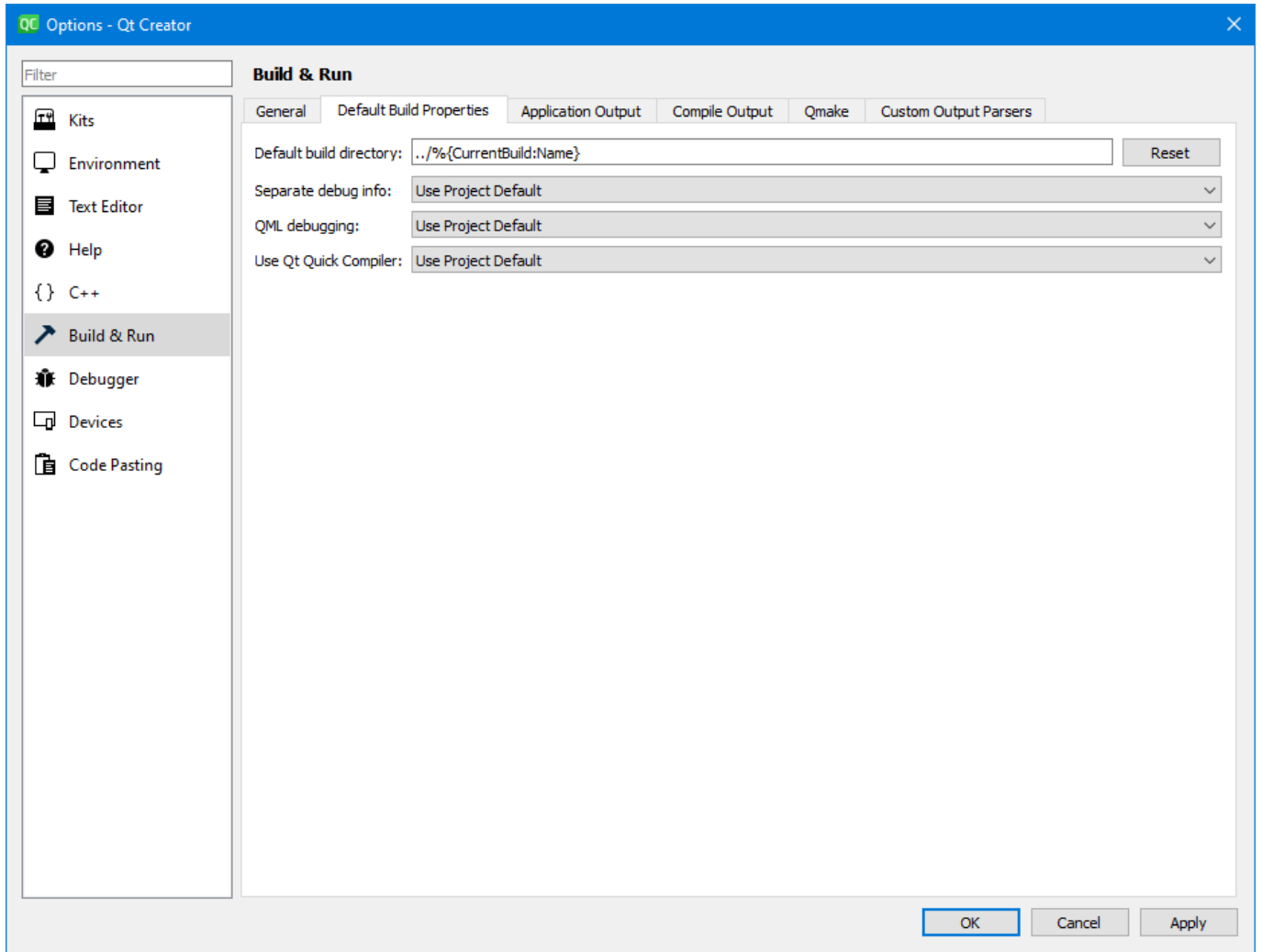
C++

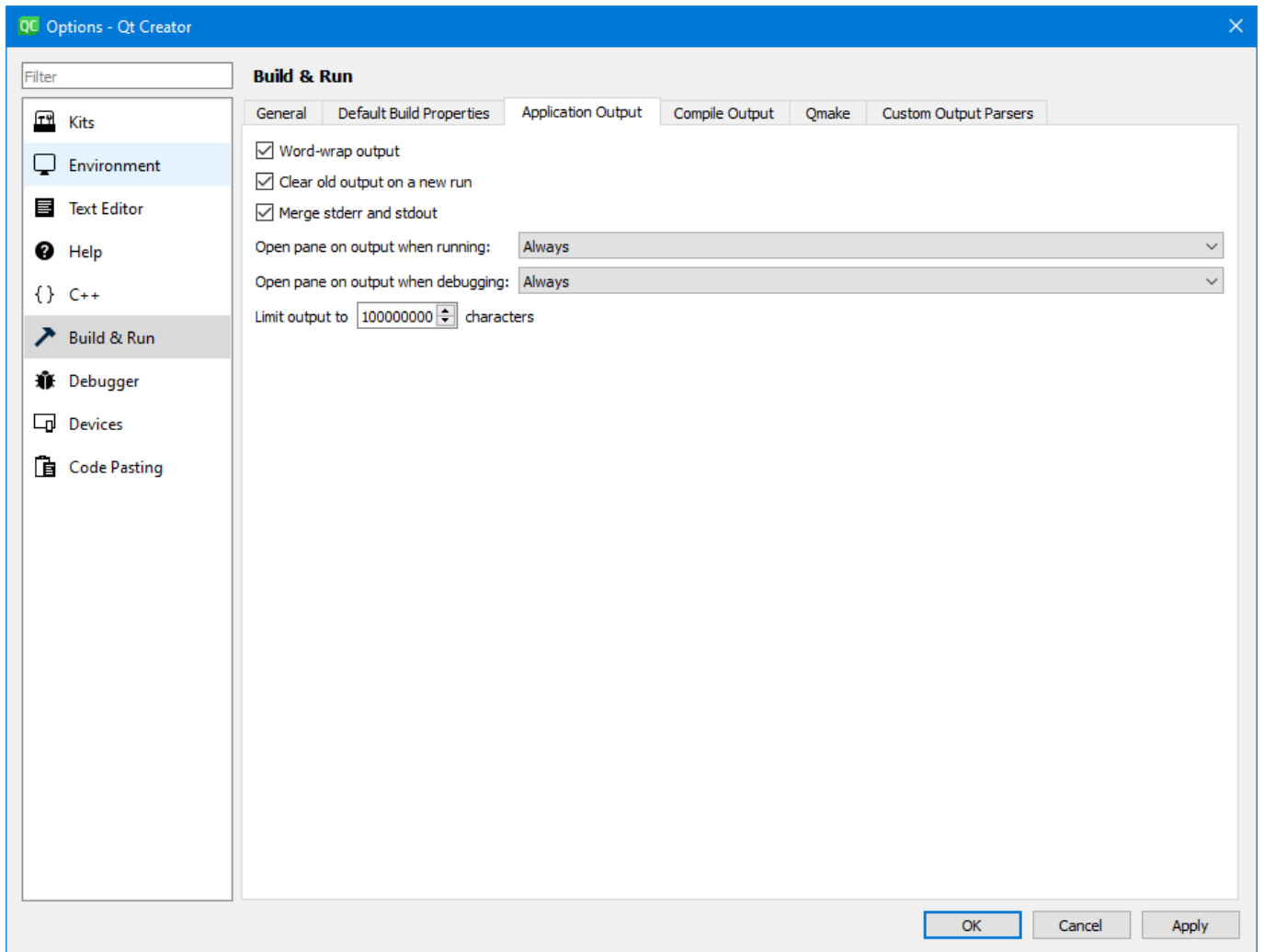


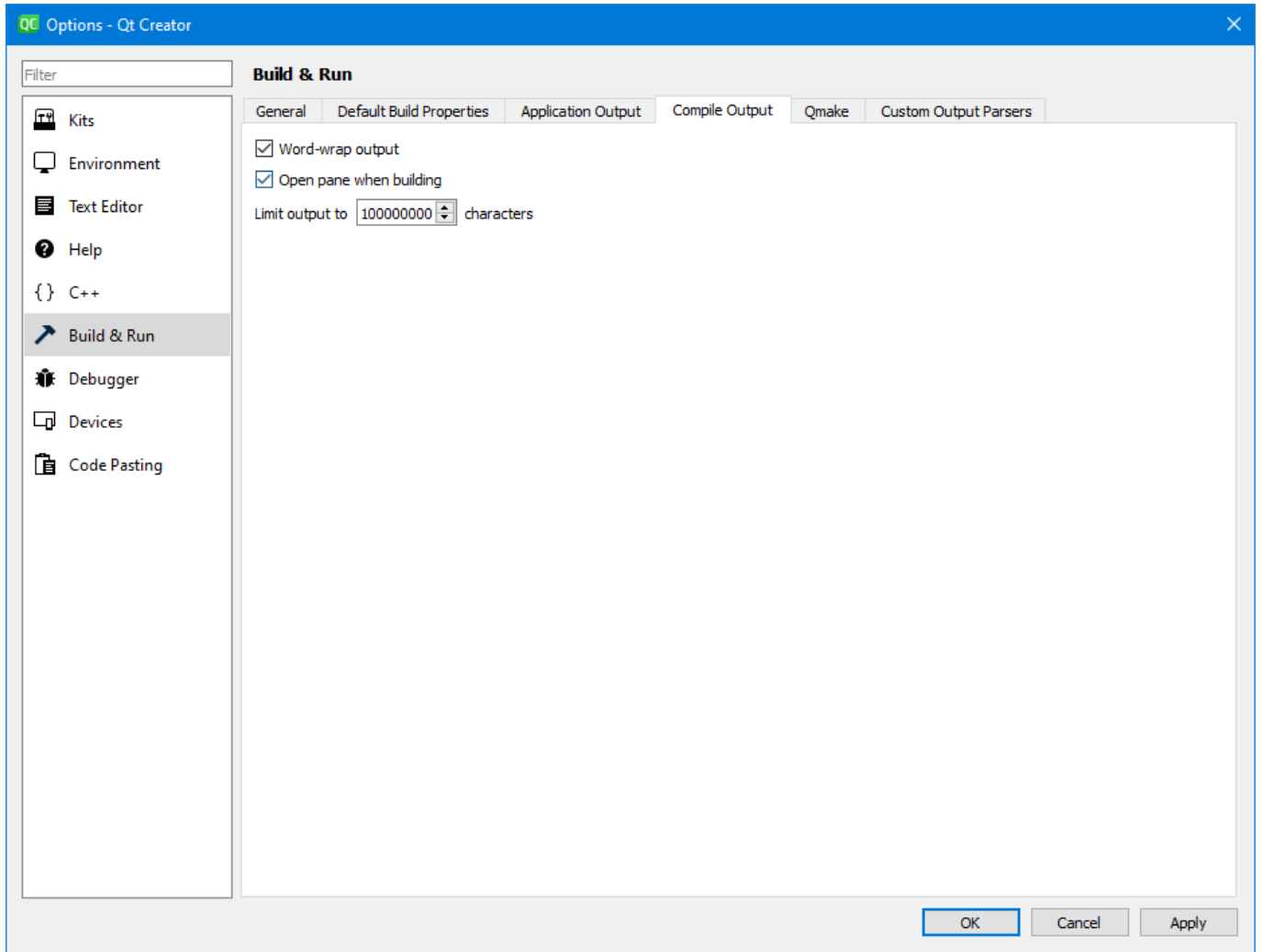
\\}}

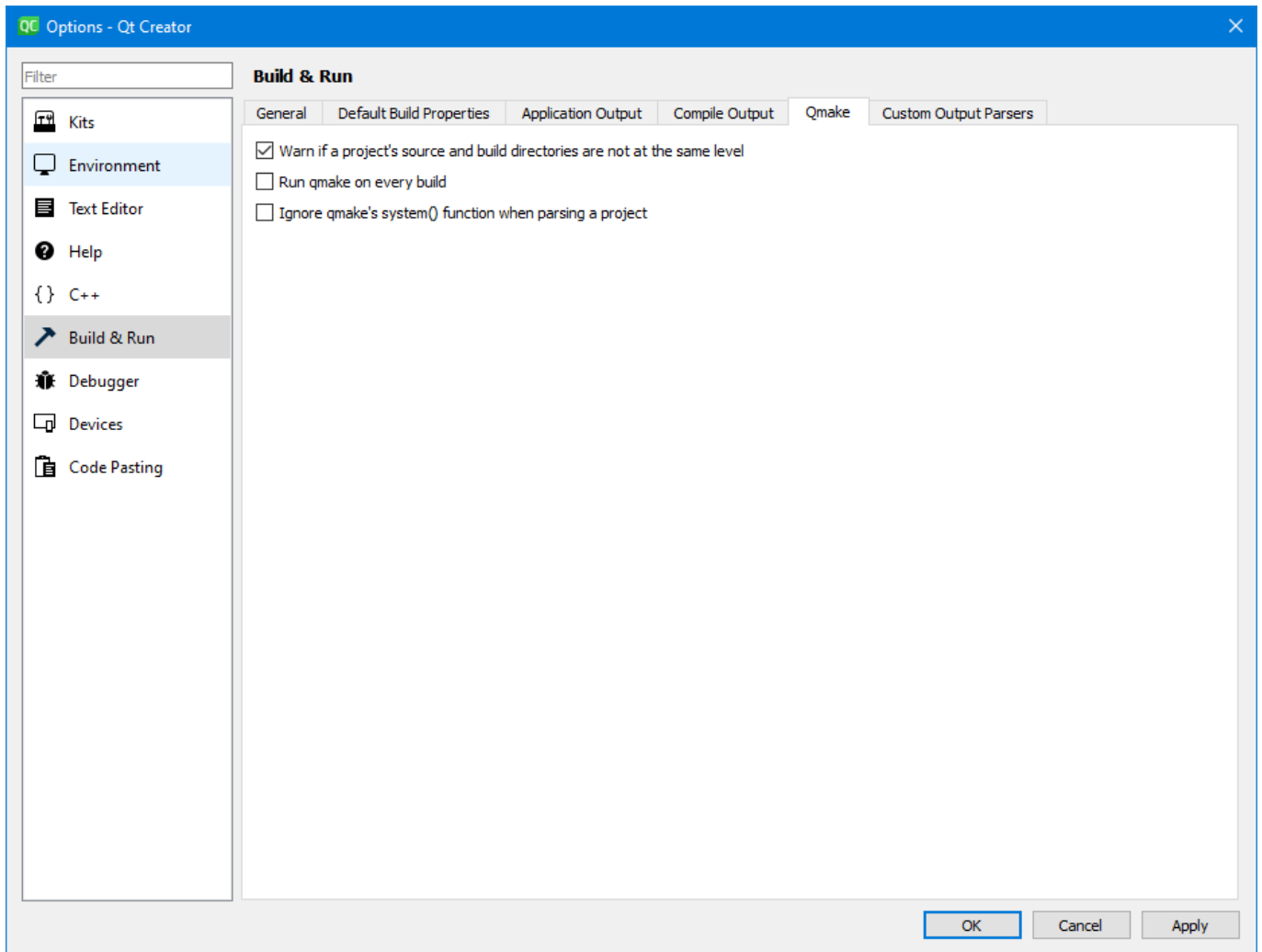
Build & Run











Nächste Schritte

Von hier können sie mit folgenden Kapiteln weiter machen:

- [Grundlegende Skripte](#) wenn sie Skripte für BPS entwickeln wollen
- [Eigene Programme und Plugins](#) wenn sie eigene Programme und Module mit C++ entwickeln wollen
- [Kernentwicklung](#) wenn sie BPS Kern selbst mit C++ entwickeln wollen

From:

<https://bps.ibk-software.com/> - **BPS WIKI**

Permanent link:

<https://bps.ibk-software.com/dok:devenv>

Last update: **23.04.2021 04:33**

