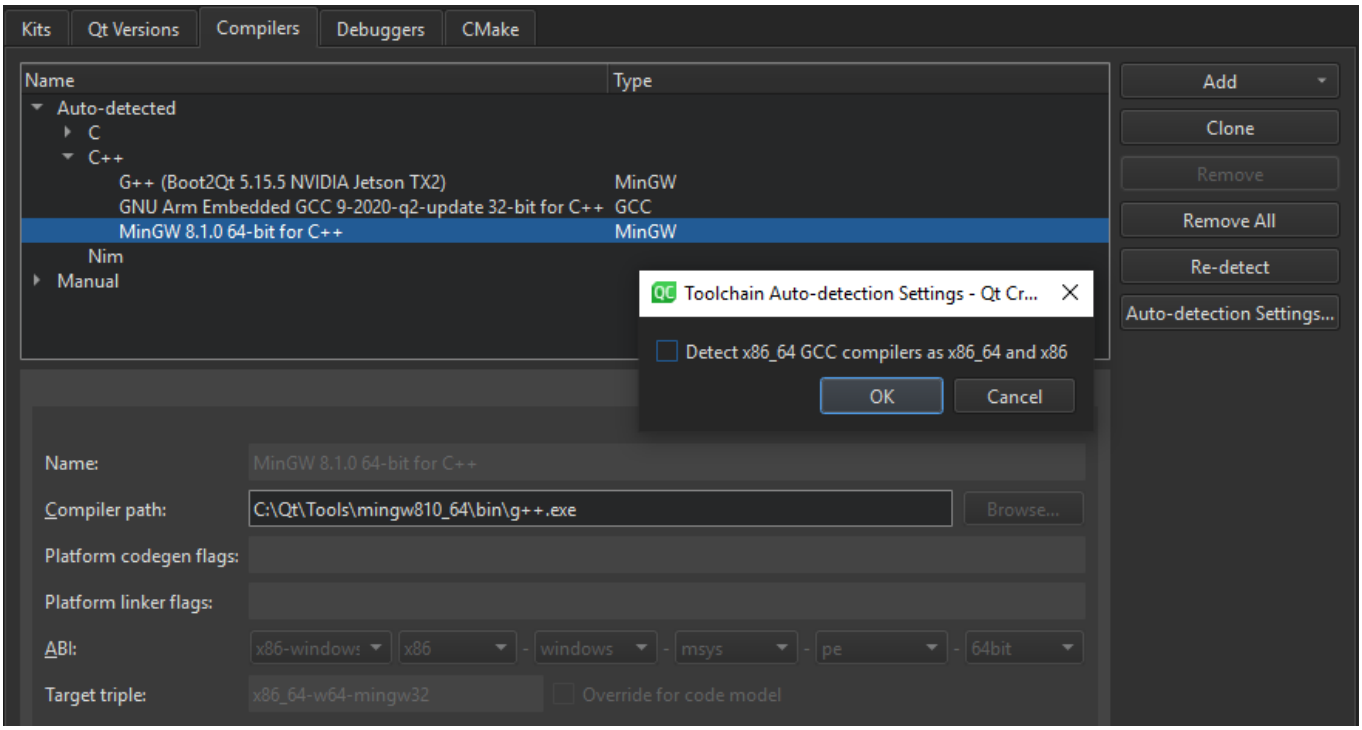
Search

Qt Creator Manual > Adding Compilers

Adding Compilers

Qt is supported on a variety of 32-bit and 64-bit platforms, and can usually be built on each platform with GCC, a vendor-supplied compiler, or a third party compiler. In Qt Creator, a kit specifies the compiler and other necessary tools for building an application for and running it on a particular platform.

Qt Creator automatically detects the compilers that are registered by your system or by the Qt Installer and lists them in **Edit** > **Preferences** > **Kits** > **Compilers**:



You can add the following compilers to build applications by using other compilers or by using additional versions of the automatically detected compilers:

- Clang is a C, C++, Objective C, and Objective C++ front-end for the LLVM compiler for Windows, Linux, and macOS.
- clang-cl is an alternative command-line interface to Clang that is compatible with the Visual C++ compiler, `.cl.exe`
- GNU Compiler Collection (GCC) is a compiler for Linux and macOS.
- ICC (Intel C++ Compiler) is a group of C and C++ compilers. Only the GCC-compatible variant, available for Linux and macOS, is currently supported by Qt Creator.

Creator and Qt for Windows.

- MSVC (Microsoft Visual C++ Compiler) is a C++ compiler that is installed with Microsoft Visual Studio.
- Nim is the Nim Compiler for Windows, Linux, and macOS.
- QCC is the interface for compiling C++ applications for QNX.

In addition, the Qt Creator Bare Metal Device plugin provides support for the following compilers:

- IAREW is a group of C and C++ bare-metal compilers from the various IAR Embedded Workbench development environments.

  > **Note:** Currently supported architectures are , , , , and .8051AVRARMSTM8MSP430

- KEIL is a group of C and C++ bare-metal compilers from the various KEIL development environments.

  > **Note:** Currently supported architectures are and .8051ARM

- SDCC is a retargetable, optimizing C bare-metal compiler for various architectures.

  > **Note:** Currently supported architectures are and .8051STM8

The emscripten compiler is tool chain for compiling to WebAssembly.

Redetecting Compilers

When Qt Creator finds an x86_64 GCC compiler, it sets up an instance for the native x86_64 target. If you plan to create also 32-bit x86 binaries without using a dedicated cross-compiler, select **Auto-detection Settings** > **Detect x86_64 GCC compilers as x86_64 and x86**. Then select **Re-detect** to refresh the list of automatically detected compilers.

To remove manually added compilers, select **Remove** or **Remove All**.

Specifying Compiler Settings

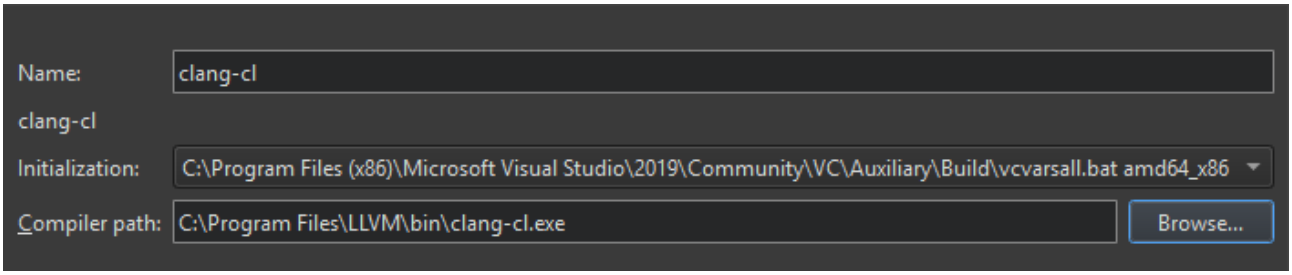
To build an application using GCC, MinGW, Clang, or QCC, specify the path to the directory where the compiler is located and select the application binary interface (ABI) version from the list of available versions. You can also create a custom ABI definition. For QCC, also specify the path to the QNX Software Development Platform (SDP) in the **SPD path** field.

To enable Microsoft Visual C++ Compilers (MSVC) and clang-cl to find system headers, libraries, and the linker, Qt Creator executes them inside a command prompt where the environment has been set up using . For these compilers, you also specify the path to the script that sets up the command prompt in the **Initialization** field.vcvarsall.bat

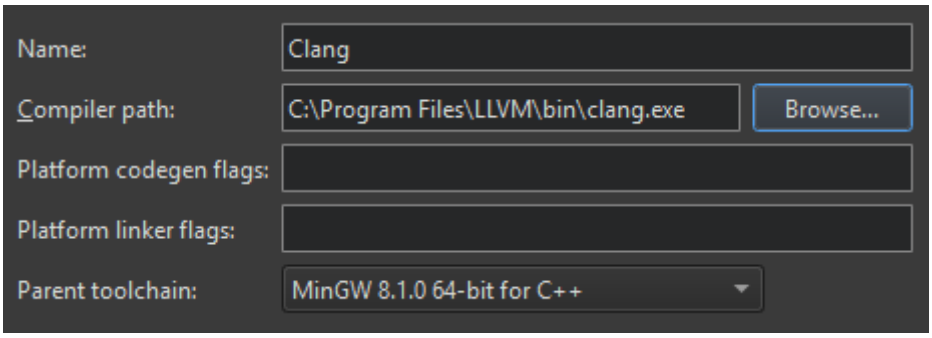
You specify the compiler to use for each kit in **Edit** > **Preferences** > **Kits**.

To add a C or C++ compiler, select **Edit** > **Preferences** > **Kits** > **Compilers** > **Add**. Select a compiler in the list, and then select **C** or **C++**.

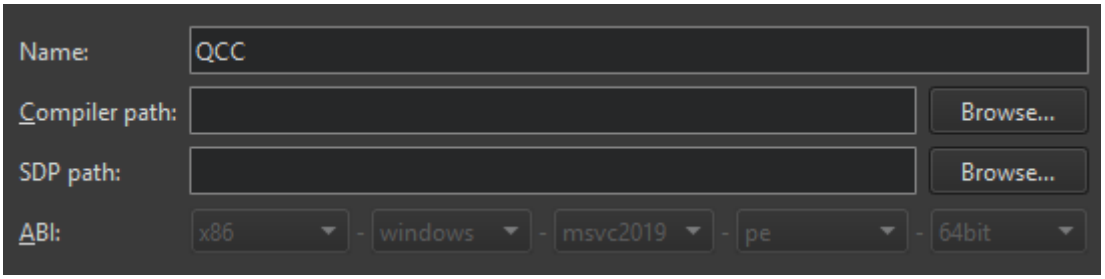
To clone the selected compiler, select **Clone**.



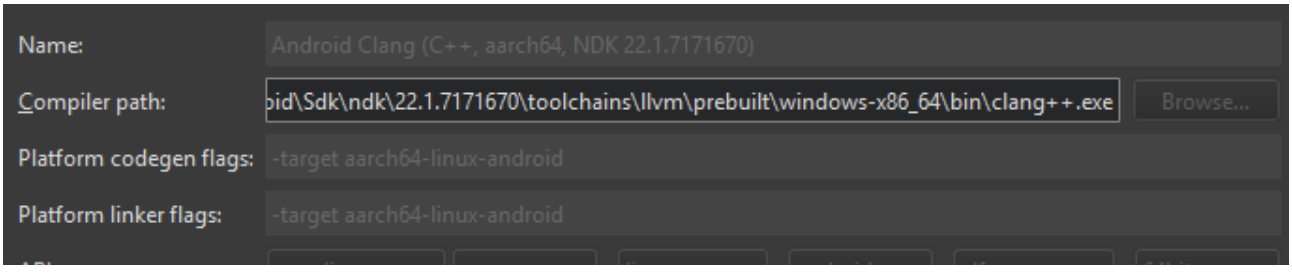
- In the **Initialization** field, select the file for setting up the command prompt to use.`vcvarsall.bat`
- In the **Compiler path** field, enter the path to the directory where the compiler is located.
- In the **Platform codegen flags** field, check the flags passed to the compiler that specify the architecture on the target platform.
- In the **Platform linker flags** field, check the flags passed to the linker that specify the architecture on the target platform. The linker flags are used only when building with Qbs.



- In the **Parent toolchain** field, select a MinGW compiler, which is needed because Clang does not have its own standard library.
- In the **SPD path** field, specify the path to the QNX Software Development Platform (SDP).



- In the **ABI** field, provide an identification for the target architecture. This is used to warn about ABI mismatches within the kits.
- In the **Target triple** field, specify the GCC target architecture. If services provided by the code model fail because Clang does not understand the target architecture, select **Override for code model**.

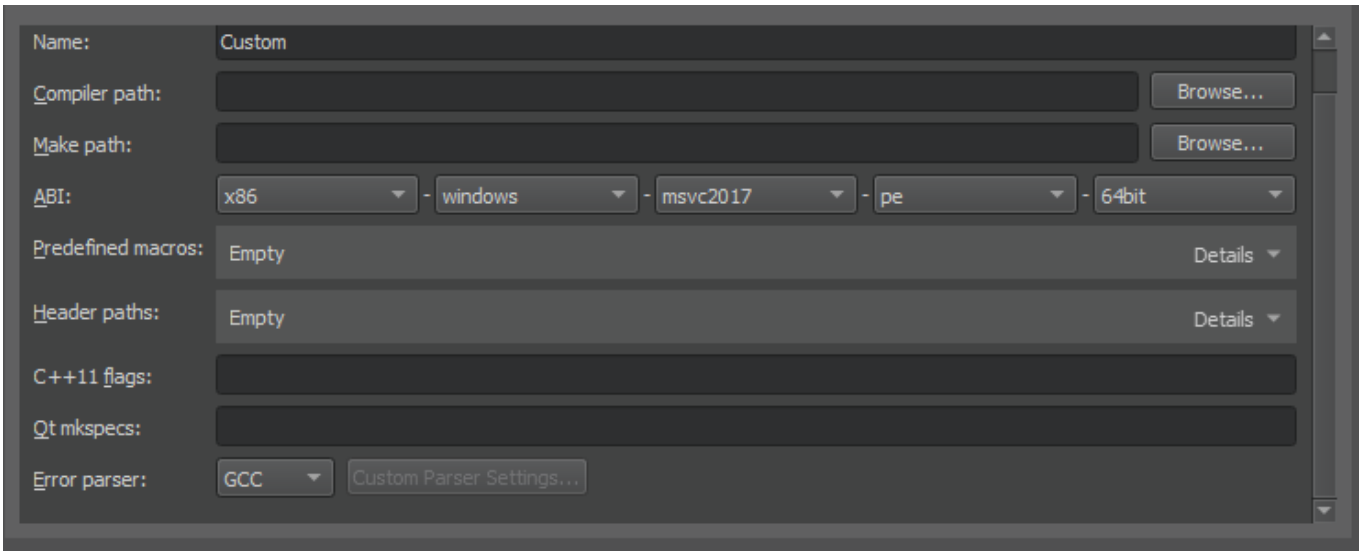


Adding Nim Compilers

To build an application using the Nim Compiler, select **Edit** > **Preferences** > **Kits** > **Compilers** > **Add** > **Nim**, and specify the path to the directory where the compiler is located.

Adding Custom Compilers

To add a compiler that is not listed above or a remote compiler, use the **Custom** option and specify the paths to the directories where the compiler and make tool are located and options for the compiler.



To add other compilers:

1. Select **Edit** > **Preferences** > **Kits** > **Compilers** > **Add** > **Custom** > **C** or **C++**.
2. In the **Name** field, enter a name for the compiler.
3. In the **Compiler path** field, enter the path to the directory where the compiler is located.
4. In the **Make path** field, enter the path to the directory where the make tool is located.
5. In the **ABI** field, specify the ABI version.
6. In the **Predefined macros** field, specify the macros that the compiler enables by default. Specify each macro on a separate line, in the following format: MACRO[=value].
7. In the **Header paths** field, specify the paths to directories that the compiler checks for headers. Specify each path on a separate line.
8. In the **C++11 flags** field, specify the flags that turn on C++11 support in the compiler.
9. In the **Qt mkspecs** field, specify the path to the directory where mkspecs are located. Usually, the path is specified relative to the Qt mkspecs directory.
10. In the **Error parser** field, select the error parser to use. You can add custom output parsers to the list. For more information, see Using Custom Output Parsers.

Troubleshooting MinGW Compilation Errors

If error messages displayed in Compile Output contain paths where slashes are missing (for example, `C:QtSDK`), check your PATH variable. At the command line, enter the following commands:

```
where make.exe
where mingw32-make.exe
```

If these commands show paths, they have been added to the global PATH variable during the installation of a tool chain based on Cygwin or MinGW, even though this is against Windows conventions.

To keep working with the third-party tool chain, create a new shell link that adds the required paths (as Visual Studio and Qt do). The shell link must point to cmd.exe, as illustrated by the following example:

```
C:\Windows\System32\cmd.exe /K C:\path_to\myenv.bat
```

where the /K parameter carries out the command specified in the bat file.

Create the myenv.bat file at *path_to*, which should be in a convenient location. In the file, specify the paths to the tool chains. For example,

```
set PATH=C:\path1;C:\path2;%PATH%
```

where *path1* and *path2* are paths to the tool chains.

Finally, remove the paths from the global PATH, reboot the computer, and run the `where` commands again to verify that the global PATH is now clean.

You can use the shell link to run the tools in the third-party tool chains.

‹ Adding Qt Versions

Adding Debuggers ›