# FAQ

This section contains answers to some frequently asked questions about Qt Creator. You might also find answers to your questions in the Known Issues and How-tos sections, or the Troubleshooting sections for a special area, such as debugging.

## General Questions

**How do I reset all Qt Creator settings?**

Remove the settings files created by Qt Creator.

For more information about where the files are located on each supported platform, see Location of Settings Files.

**Qt Creator comes with MinGW, should I use this version with Qt?**

Use the version that was built against the Qt version.

**Qt Creator does not find a helper application, such as Git or a compiler. What should I do?**

Make sure that the application is in your system PATH when starting Qt Creator. Also select **Edit** > **Preferences** to check the settings specified for the application. Many plugins specify either the path to the tool they need or the environment they run in.

This is especially relevant for the macOS where `/usr/local/bin` might not be in the path when Qt Creator is started.

**How do I change the interface language for Qt Creator?**

Qt Creator has been localized into several languages. If the system language is one of the supported languages, it is automatically selected. To change the language, select **Edit** > **Preferences** > **Environment** and select a language in the **Language** field. Select **Restart Now** to restart Qt Creator and have the change take effect.

**Has a reported issue been addressed?**

You can look up any issue in the Qt Project Bug Tracker.

## Qt Designer Integration Questions

**Why are custom widgets not loaded in the Design mode even though it works in standalone Qt Designer?**

Qt Designer fetches plugins from standard locations and loads the plugins that match its build key. The locations are different for standalone and integrated Qt Designer.

For more information, see Adding Qt Designer Plugins.

**Why is there a red line below my QML import, even though I have the module?**

By default, Qt Creator looks in the QML import path of Qt for QML modules. Sometimes, it does not get it right and you need to tell it where the modules are. When using qmake as the build system, specify the `QML_IMPORT_PATH` in the `.pro` file of your application. When using CMake, add the `set` command to the CMakeLists.txt file.

This also enables code completion of QML code and removes error messages.

The following example illustrates how to specify the import path for qmake projects so that it works when switching between build and run kits for different target platforms:

```
TEMPNAME = $${QMAKE_QMAKE}
MY_QTPATH = $$dirname(TEMPNAME)
QML_IMPORT_PATH += $$MY_QTPATH/../qml
message("my QML Import Path: "$$QML_IMPORT_PATH)
```

For more information about how to set the import path for CMake projects, see Importing QML Modules.

**What should I do when Qt Creator complains about missing OpenGL support?**

Some parts of Qt Creator, such as QML Profiler, use Qt Quick 2, which relies on OpenGL API for drawing. Unfortunately, the use of OpenGL can cause problems, especially in remote setups and with outdated drivers. In these cases, Qt Creator displays OpenGL-related error messages on the console or records them in the Windows debugger log.

The fixes and workarounds differ, depending on your setup. As a last resort, you can disable the affected plugins.

*Virtual Machines*

Try to enable *3D acceleration* in your virtual machine's settings. For VirtualBox, also make sure you have installed the Guest Addons, including experimental *Direct3D support*.

*Windows*

Check whether Qt Creator has been compiled with OpenGL/Desktop, or ANGLE as a backend. The official binaries are always built with ANGLE (a library that maps OpenGL ES API to DirectX).

- ANGLE backend: This requires a Windows version newer than Windows XP. If you have problems, try updating your graphics drivers or update your DirectX version. Run `dxdiag.exe` to check whether *Direct3D Acceleration* is indeed enabled.
- OpenGL backend: Make sure your graphics driver supports OpenGL 2.1 or newer. Try to update your graphics driver.

*Unix*

Run `glxgears` for a quick check whether OpenGL works in general. Check the output of `glxinfo` to get more details like the OpenGL driver and renderer (search for *OpenGL* in the application's output).

If you are using the Mesa driver, you can force OpenGL to be rendered in software by setting the `LIBGL_ALWAYS_SOFTWARE` environment variable.

*Disabling plugins*

You can disable the Qt Creator plugins, at the expense of losing functionality:

- Launch Qt Creator from command line, with the `-noload QmlProfiler -noload QmlDesigner` arguments.

# Help Questions

**The Qt API Reference Documentation is missing and context help does not find topics. What can I do?**

Qt Creator comes fully integrated with Qt documentation and examples using the Qt Help plugin. The integrated Qt Reference Documentation is available for Qt 4.4 and later. Qt Creator and other Qt deliverables contain documentation as .qch files. All the documentation is accessible in the **Help** mode.

To view the documentation that is available and to add documentation, select **Edit** > **Preferences** > **Help** > **Documentation**. For more information, see Adding External Documentation.

# Debugger Questions

For information on troubleshooting debugger, see Troubleshooting Debugger.

**If I have a choice of GDB versions, which should I use?**

On Linux and Windows, use the Python-enabled GDB versions that are installed when you install Qt Creator and Qt. On macOS, GDB is no longer officially supported. To build your own Python-enabled GDB, follow the instructions in Building GDB.

You must use Python version 2.6 or 2.7.

For more information on setting up debugger, see Setting Up Debugger.

**How do I generate a core file in Qt Creator?**

To trigger the GDB command that generates a core file while debugging, select **View** > **Views** > **Debugger Log**. In the **Command** field, type `gcore` and press **Enter**. The core file is created in the current working directory. You can specify another location for the file, including a relative or absolute path, as an argument of the command.

# Compiler Questions

**How can I make use of my multi-core CPU with Qt Creator?**

On Linux and macOS, go to **Projects** mode, select your configuration in the **Build Settings**, locate the **Build Steps**, and add the following value, where <num> is the amount of cores in your CPU: `-j <num>`

On Windows, nmake does not support the `-j` parameter. Instead, we provide a drop-in replacement called jom. You can download a precompiled version of jom from Qt Downloads. Put jom.exe in a location in the %PATH%. Go to the **Build Settings** and set jom.exe as the make command.

> **Note:** Unlike GNU make, jom automatically detects your cores and spawns as many parallel processes as your CPU has cores. You can override this behavior by using the `-j` parameter as described above.

# Qt Installation Questions

**I cannot use QSslSocket with the Qt I installed from binary packages. What should I do?**

The Qt in the binary packages is built with QT_NO_OPENSSL defined. Rebuilding it is possible. For more information, see http://www.qtcentre.org/threads/19222-Qssl.

**Which development packages from the distribution are needed on Ubuntu or Debian?**

```
sudo apt-get install libglib2.0-dev libsm-dev libxrender-dev libfontconfig1-dev libxext-dev
```

If you use QtOpenGL, you also need:

```
sudo apt-get install libgl-dev libglu-dev
```

# Platform Related Questions

**Where is application output shown in Qt Creator?**

**On Unix (Linux and macOS):** `qDebug()` and related functions use the standard output and error output. When you run or debug the application, you can view the output in Application Output.

For console applications that require input, select **Projects** > **Run Settings** > **Run in terminal**. To specify the terminal to use, select **Edit** > **Preferences** > **Environment** > **System**.

**On Windows:** Output is displayed differently for *console applications* and *GUI applications*.

For qmake projects, the `CONFIG += console` setting in the .pro file specifies that the application is built as a console application using some other runtime.

This is the standard behavior for CMake projects. To create a GUI application on Windows or an application bundle on macOS, you must specify the `WIN32` or `MACOSX_BUNDLE` property for the qt_add_executable command in the CMakeLists.txt file.

When you run a console application, you can view the output in the console window of the calling application. If the calling application is a GUI application (for example, a release-built version of Qt Creator), a new console window is opened. For this type of application, `qDebug()` and related functions use standard output and error output.

We recommend that you select **Projects** > **Run Settings** > **Run in terminal** for console applications.

For GUI applications, `qDebug()` and related functions use the Windows API function `OutputDebugString()`. The output is displayed in **Application Output**. However, Qt Creator can show output from only one source at the time for it to be displayed correctly. You can use an external debug output viewer, such as the DebugView for Windows to display output from GUI applications.

# Questions about New Features

**Will a requested feature be implemented?**

If it is a scheduled feature, you can see this in the task tracker. If a feature already has been implemented, it is mentioned in the changes file for the upcoming release.

**Why does Qt Creator not use tabs for editors?**

This question comes up from time to time, so we have considered it carefully. Here are our main reasons for not using tabs:

- Tabs do not scale. They work fine if you have 5 to 6 editors open, they become cumbersome with 10, and if you need more horizontal space than the tab bar, the interface does not work at all.
- Tabs do not adapt to your working set.
- The common solution is to give the user the ability to reorder tabs. Now user has to manage tabs instead of

Consider the following use case: *Developers want to switch editors.*

In fact, developers do not want to switch editors, but might have to do so to accomplish their tasks. We need to figure out what the tasks are to provide developers with better ways to navigate while performing the tasks.

One common factor in many use cases is switching editors while working on a set of open files. While working on files A and B, users sometimes need to look at file C. They can press **Ctrl+Tab** to move between the files and have the files open in the correct editor according to file type. The list is sorted by last used.

Typically, users also work on multiple classes or functions that are related, even though they are defined or declared in different files. Qt Creator provides two shortcuts for that: **F2** to follow the symbol under cursor and **Ctrl+Shift+U** to find references to it.

In addition, developers can:

- Press **F4** to switch between header and source.
- Press **Alt+Left** to move backwards in the navigation history.
- Use the locator (Ctrl+K) to simply tell Qt Creator where to go.

The locator can be used to open files, but opening files is also just a step on the way to accomplish a task. For example, consider the following use case: *Fix AFunction in SomeClass which comes from someclass.cpp/someclass.h.*

With a tabbed user interface, developers would search for someclass.cpp in the tab bar, and then search for `::AFunction`, only to find out that the function is not located in that file. They would then search for someclass.h in the tab bar, find our that the function is inline, fix the problem, and forget where they came from.

With Qt Creator, developers can type `Ctrl+K m AFun` to find the function. Typically, they only need to type 3 to 4 characters of the function name. They can then fix the problem and press **Alt+Back** to go back to where they were.

Other locator filters include `c` for classes, `:` for all symbols, and (thanks to a community contribution) `.` for symbols in the current file.

‹ Using the Help Mode

How-tos ›