Parsing C++ Files with the Clang Code Model

The *code model* is the part of an IDE that understands the language you are using to write your application. It is the framework that allows Qt Creator to provide the following services:

- Code completion
- Syntactic and semantic highlighting
- Navigating in the code by using the locator, following symbols, and so on
- Inspecting code by using the class browser, the outline, and so on
- Diagnostics
- Tooltips
- Finding and renaming symbols
- Refactoring actions

Qt Creator comes with a plugin that provides some of these services for C++ on top of Clangd.

About the Clang Code Model

The Clang project provides libraries for parsing C language family source files. The feedback you get through warning and error markers is the same as a compiler will give you, not an incomplete set or a close approximation, as when using the built-in Qt Creator code model. Clang focuses on detailed information for diagnostics, which is really useful if the code contains typos, for example. We make use of these libraries via the clangd tool, which implements an LSP server.

Clang keeps up with the development of the C++ language. At the time of this writing, it supports C++98/03, C++11, C++14, C++17, C89, C99, Objective-C, and Objective-C++.

On the downside, for large projects using Clang as code model is slower than using the built-in code model. Clang does not need to generate object files, but it still needs to parse and analyze the source files. For small projects that only use STL, this is relatively fast. But for larger projects that include several files, processing a single file and all the included files can take a while.

The Clang code model plugin now provides some of the services that were previously provided by the built-in C/C++ code model. Currently, the following services are implemented:

- Code completion
- Syntactic and semantic highlighting
- Diagnostics
- Outline of symbols

- Renaming symbols
- Finding occurrences of symbols

To use the built-in code model instead, select **Edit** > **Preferences** > **C++** > **clangd**, and deselect the **Use clangd** check box. This setting also exists on the project level, so that you can have the clang-based services generally enabled, but switch them off for for certain projects, or vice versa.

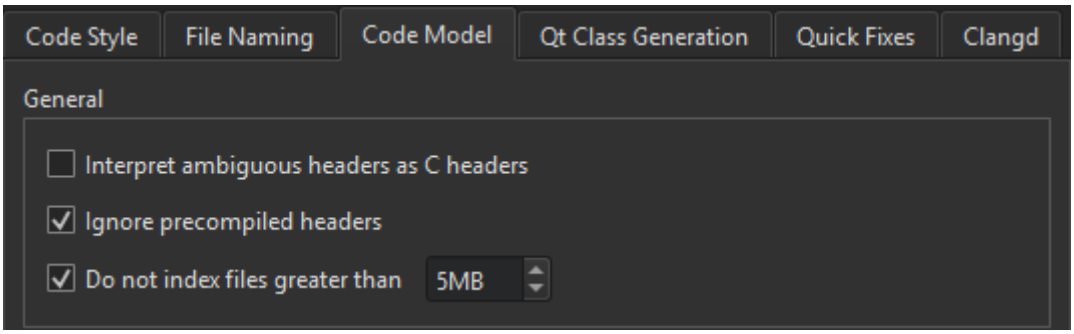
You can configure Clang diagnostics either globally or separately for:

- Clang code model (globally or at project level)
- Clang tools (globally or at project level)

Configuring Clang Code Model

To configure the Clang code model globally:

1. Select **Edit** > **Preferences** > **C++** > **Code Model**.



2. To instruct the code model to interpret ambiguous header files as C language files if you develop mainly using C, select the **Interpret ambiguous headers as C headers** check box.
3. To process precompiled headers, deselect the **Ignore precompiled headers** check box.
4. To avoid out-of-memory crashes caused by indexing huge source files that are typically auto-generated by scripts or code, the size of files to index is limited to 5MB by default. To adjust the limit, edit the value for the **Do not index files greater than** check box. To index all files, deselect the check box.

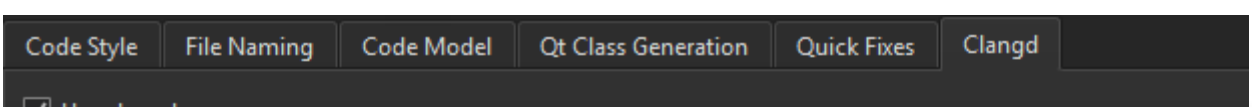
Configuring clangd

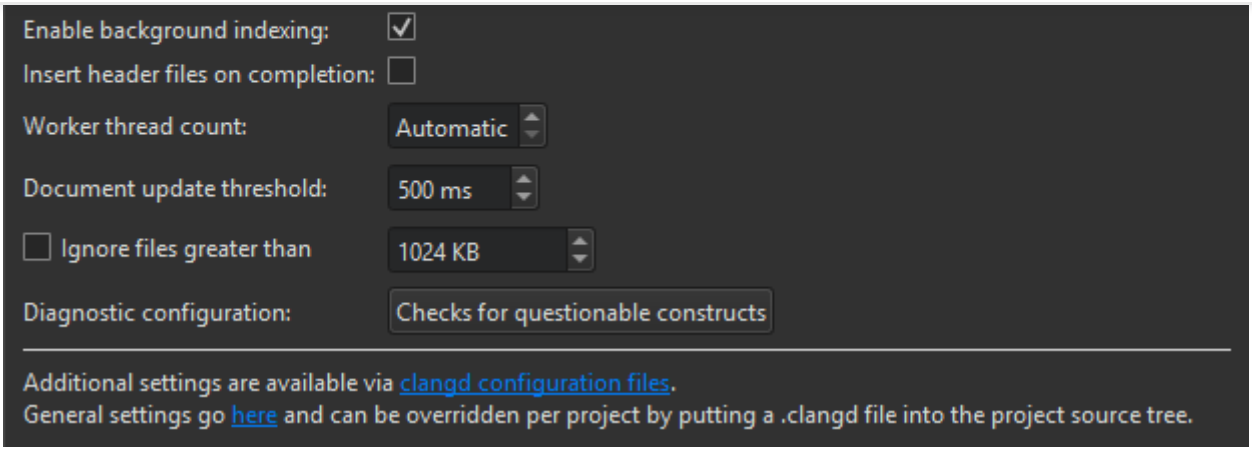
The clangd *index* provides exact and complete results for services such as finding references, following symbols under cursor, and using the locator, even for complex constructs. When you open a project, clangd scans the source files to generate the index. For large projects, this can take a while, but the index is persistent and re-scanning is incremental, so nothing is lost by closing and re-starting Qt Creator.

The document outline in the Outline view is backed by clangd's document symbol support, which makes the results more reliable than before.

To specify settings for clangd:

1. Select **Edit** > **Preferences** > **C++** > **Clangd** > **Use clangd**.





2. In **Path to executable**, enter the path to clangd version 13, or later.
3. For more accurate results during global symbol searches, select **Enable background indexing**. However, this increases the CPU load the first time you open the project.
4. Select **Insert header files on completion** to allow clangd to insert header files as part of symbol completion.
5. By default, clangd attempts to use all unused cores. You can set a fixed number of cores to use in **Worker thread count**.
6. In **Document update threshold**, specify the amount of time Qt Creator waits before sending document changes to the server. If the document changes again while waiting, this timeout is reset.
7. Select **Ignore files greater than** to make parsing faster by ignoring big files. Specify the maximum size of files to parse in the field next to the check box.
8. The **Diagnostic Configuration** field shows the Clang checks to perform. Click the value of the field to open the **Diagnostic Configurations** dialog, where you can select and edit the checks to perform.

Clang Checks

In addition to using the built-in checks, you can select **Copy** to create copies of them and edit the copies to fit your needs.

Build-system warnings displays warnings as specified by the build system.

Checks for questionable constructs combines the `-Wall` and `-Wextra` checks for easily avoidable questionable constructions and some additional issues.

Clang checks begin with `-W`. Each check also has a negative version that begins with `-Wno`.

Keep in mind that some options turn on other options. For more information, see Options to Request or Suppress Warnings or the GCC or Clang manual pages.

Specifying Clang Code Model Settings at Project Level

You can specify Clang code model settings at project level by selecting **Projects** > **clangd**.

Using Compilation Databases

The JSON compilation database format specifies how to replay single builds independently of the build system.

A *compilation database* is basically a list of files and the compiler flags that are used to compile the files. The

To generate a compilation database from the information that the code model has, select **Build** > **Generate Compilation Database**.

You can add files, such as non-C files, to the project in *compile_database.json.files*.

You can use the experimental Compilation Database Project Manager to open the files in a compilation database with access to all the editing features provided by the Clang code model.

To switch between header and source files, select **Tools** > **C++** > **Switch Header/Source**.

You can specify custom build steps and run settings for compilation database projects in the **Projects** mode. For more information, see Adding Custom Build Steps and Specifying Run Settings.

To enable the plugin, select **Help** > **About Plugins** > **Build Systems** > **Compilation Database Project Manager**. Then select **Restart Now** to restart Qt Creator and load the plugin.

‹ Comparing Files

Finding ›