c_cpp_properties.json reference

This article explains the scheme for the `c_cpp_properties.json` settings file.

Looking to get started with configuring your project? See Configure Intellisense (/docs/cpp/configure-intellisense).For more information about changing these settings, see Customizing Default Settings (/docs/cpp/customize-default-settings-cpp).

Example of Variables

Note, this is an example of all fields. You do not need to specify all fields in your `c_cpp_properties.json` file. The extension will automatically fill in any missing fields with default values.

```
{
  "env": {
    "myIncludePath": ["${workspaceFolder}/include", "${workspaceFolder}/src"],
    "myDefines": ["DEBUG", "MY_FEATURE=1"]
  },
  "configurations": [
    {
      "name": "Linux",
      "compilerPath": "/usr/bin/gcc",
      "compilerArgs": ["-m32"],
      "intelliSenseMode": "linux-gcc-x86",
      "includePath": ["${myIncludePath}", "/usr/include"],
      "defines": ["${myDefines}"],
      "cStandard": "gnu11",
      "cppStandard": "gnu++14",
      "configurationProvider": "ms-vscode.cmake-tools",
      "forcedInclude": ["${workspaceFolder}/common.h"],
      "compileCommands": "${workspaceFolder}/build/compile_commands.json",
      "dotconfig": "${workspaceFolder}/.config",
      "mergeConfigurations": true,
      "customConfigurationVariables": {
        "myVar": "myvalue"
      },
      "browse": {
        "path": ["${myIncludePath}", "/usr/include", "${workspaceFolder}"],
        "limitSymbolsToIncludedHeaders": true,
        "databaseFilename": "${workspaceFolder}/.vscode/browse.vc.db"
      }
    },
    {
      "name": "Mac",
      "compilerPath": "/usr/bin/clang",
      "intelliSenseMode": "macos-clang-x64",
      "includePath": ["${myIncludePath}"],
      "defines": ["${myDefines}"],
      "cStandard": "c11",
      "cppStandard": "c++17",
      "macFrameworkPath": ["/System/Library/Frameworks", "/Library/Frameworks"],
      "browse": {
        "path": ["${myIncludePath}", "${workspaceFolder}"]
      }
    },
    {
      "name": "Win32",
      "compilerPath": "C:/Program Files (x86)/Microsoft Visual Studio/2019/Communit
y/VC/Tools/MSVC/14.28.29333/bin/Hostx64/x64/cl.exe",
      "intelliSenseMode": "windows-msvc-x64",
      "includePath": ["${myIncludePath}"],
      "defines": ["${myDefines}", "_WINDOWS"],
      "cStandard": "c17",
      "cppStandard": "c++20",
      "windowsSdkVersion": "10.0.19041.0",
      "browse": {
        "path": ["${myIncludePath}", "${workspaceFolder}"]
      }
    }
  ],
  "version": 4,
  "enableConfigurationSquiggles": true
}
```

Top-level properties

- `env` An array of user-defined variables that will be available for substitution in the configurations via the standard environment variable syntax: `${<var>}` or `${env:<var>}`. Strings and arrays of strings are accepted.
- `configurations` An array of configuration objects that provide the IntelliSense engine with information about your project and your preferences. By default, the extension creates a configuration for you based on your operating system. You may also add additional configurations.
- `version` We recommend you don't edit this field. It tracks the current version of the `c_cpp_properties.json` file so that the extension knows what properties and settings should be present and how to upgrade this file to the latest version.
- `enableConfigurationSquiggles` Set to `true` to report errors detected in `c_cpp_properties.json` file to the C/C++ Extension.

Configuration properties

- `name` A friendly name that identifies a configuration. `Linux`, `Mac`, and `Win32` are special identifiers for configurations that will be autoselected on those platforms. The status bar in VS Code will show you which configuration is active. You can also select the label in the status bar to change the active configuration.
- `compilerPath` (optional) The full path to the compiler you use to build your project, for example `/usr/bin/gcc`, to enable more accurate IntelliSense. The extension will query the compiler to determine the system include paths and default defines to use for IntelliSense.

  Putting `"compilerPath": ""` (empty string) will skip querying a compiler. This is useful if a specified compiler doesn't support the arguments that are used for the query, as the extension will default back to any compiler it can find (like Visual C). Leaving out the `compilerPath` property does not skip the query.
- `compilerArgs` (optional) Compiler arguments to modify the includes or defines used, for example `-nostdinc++`, `-m32`, etc. Arguments that take additional space-delimited arguments should be entered as separate arguments in the array, for example, for `--sysroot <arg> use \"--sysroot\", \"<arg>\"`.
- `intelliSenseMode` The IntelliSense mode to use that maps to an architecture-specific variant of MSVC, gcc, or Clang. If not set or if set to `${default}`, the extension will choose the default for that platform.

Platform defaults:

  - Windows: `windows-msvc-x64`
  - Linux: `linux-gcc-x64`
  - macOS: `macos-clang-x64`

  IntelliSense modes that only specify `<compiler>-<architecture>` variants (for example, `gcc-x64`) are legacy modes and are automatically converted to the `<platform>-<compiler>-<architecture>` variants based on the host platform.

- `includePath` An include path is a folder that contains header files (such as `#include "myHeaderFile.h"`) that are included in a source file. Specify a list of paths for the IntelliSense engine to use while searching for included header files. Searching on these paths is not recursive. Specify `**` to indicate recursive search. For example, `${workspaceFolder}/**` will search through all subdirectories while `${workspaceFolder}` will not. If on Windows with Visual Studio installed, or if a compiler is specified in the `compilerPath` setting, it is not necessary to list the system include paths in this list.

- `defines` A list of preprocessor definitions for the IntelliSense engine to use while parsing files. Optionally, use `=` to set a value, for example `VERSION=1`.

- `cStandard` The version of the C language standard to use for IntelliSense. For example, `c17`, `gnu23`, or `${default}`. Note that GNU standards are only used to query the set compiler to get GNU defines, and IntelliSense will emulate the equivalent C standard version.

- `cppStandard` The version of the C++ language standard to use for IntelliSense. For example, `c++20`, `gnu++23`, or `${default}`. Note: GNU standards are only used to query the set compiler to get GNU defines, and IntelliSense will emulate the equivalent C++ standard version.

- `configurationProvider` The ID of a VS Code extension that can provide IntelliSense configuration information for source files. For example, use the VS Code extension ID `ms-vscode.cmake-tools` to provide configuration information from the CMake Tools extension. If you have specified a configurationProvider, the configurations that provides will take precedence over your other settings in `c_cpp_properties.json`.

  A `configurationProvider` candidate extension must implement vscode-cpptools-api (https://github.com/microsoft/vscode-cpptools-api).

- `windowsSdkVersion` The versions of the Windows SDK include path to use on Windows, for example `10.0.17134.0`.

- `macFrameworkPath` A list of paths for the IntelliSense engine to use while searching for included headers from Mac frameworks. Only supported on configurations for macOS.

- `forcedInclude` (optional) A list of files that should be included before any other characters in the source file are processed. Files are included in the order listed.

- `compileCommands` (optional) The full path to the `compile_commands.json` file for the workspace. If there is a matching entry in `compile_commands.json` for a file open in the editor, that command line will be used to configure IntelliSense for that file, instead of the other fields of `c_cpp_properties.json`. For more information about the file format, see the Clang documentation (https://clang.llvm.org/docs/JSONCompilationDatabase.html). Some build systems, such as CMake, simplify generating this file (https://cmake.org/cmake/help/v3.5/variable/CMAKE_EXPORT_COMPILE_COMMANDS.html).

- `dotconfig` A path to a .config file created by the Kconfig system. The Kconfig system generates a file with all the defines needed to build a project. Examples of projects that use the Kconfig system are the Linux Kernel and NuttX RTOS.
- `mergeConfigurations` Set to `true` to merge include paths, defines, and forced includes with those from a configuration provider.
- `customConfigurationVariables` Custom variables that can be queried through the command `${cpptools:activeConfigCustomVariable}` to use for the input variables in `launch.json` or `tasks.json`.
- `browse` The set of properties used when `"C_Cpp.intelliSenseEngine"` is set to `"Tag Parser"` (also referred to as "fuzzy" IntelliSense, or the "browse" engine). These properties are also used by the **Go to Definition/Declaration** features, or when the "default" IntelliSense engine is unable to resolve the `#includes` in your source files.

Browse properties

- `path` A list of paths for the Tag Parser to search for headers included by your source files. If omitted, `includePath` will be used as the `path`. Searching on these paths is recursive by default. Specify `*` to indicate nonrecursive search. For example: `${workspaceFolder}` will search through all subdirectories while `${workspaceFolder}/*` will not.
- `limitSymbolsToIncludedHeaders` When true, the Tag Parser will only parse code files that have been directly or indirectly included by a source file in `${workspaceFolder}`. When false, the Tag Parser will parse all code files found in the paths specified in the `browse.path` list.
- `databaseFilename` The path to the generated symbol database. This property instructs the extension to save the Tag Parser's symbol database somewhere other than the workspace's default storage location. If a relative path is specified, it will be made relative to the workspace's default storage location, not the workspace folder itself. The `${workspaceFolder}` variable can be used to specify a path relative to the workspace folder (for example `${workspaceFolder}/.vscode/browse.vc.db`)

Supported variables

You can allow `tasks.json` or `launch.json` to query the current active configuration from `c_cpp_properties.json`. To do this, use the variable `${command:cpptools.activeConfigName}` as an argument in a `tasks.json` or `launch.json` script.