# How to Install and Run CMake on Windows

**Pitch:** Visual Studio's native support for CMake enables you to edit, build, and debug CMake projects on Windows. In this article, you will guide the reader through the process of how to install and run Cmake on a Windows machine.

Reader Goal: Leave the article knowing how to install and run Cmake on a windows machine.

**Audience:** Beginner developers looking to better understand how to install and run Cmake on Windows with Visual Studio

**Client:** Earthly.dev

**What sets the company apart?** CI/CD Framework that allows you to develop locally and run anywhere.

**Writer Deliverables:** ~1500 word article,Screenshots

**Outline:** (this is just a starting point, feel free to modify it as you see fit)

- ​​Introduction (1-3 paragraphs)

- Introduce the reader to Cmake (briefly)

- What will the reader learn in this article?

- Reasons for using Cmake on windows

- Advantages to using instead of visual studio projects?

- Disadvantages?

- Guidelines for when to use

- Implementing Cmake and Visual Studio on Windows (meat of article)

- Step-by-step instructions on How to Install and Run Cmake on Windows / Visual Studio

- NOTE TO WRITER: Be sure to include \_why\_ each step is required

- Installation

- Briefly explain that both C++ CMake tools for Windows and Linux Development with C++ are required for cross-platform CMake development on Windows

- https://docs.microsoft.com/en-us/cpp/linux/download-install-and-setup-the-linux-development-workload?view=msvc-170

- Briefly explain IDE integration

- Show how to configure CMake projects

- Explain how to build CMake projects on Visual Studio

- Explain how to debug CMake projects

- Conclusion (2-3 paragraphs)

- Restate what the reader has learned

- Introduce the reader to Earthly, the effortless CI/CD framework that runs everywhere

**Reference(s):**

https://cmake.org/runningcmake/

**Example Article(s):**

https://docs.microsoft.com/en-us/cpp/build/cmake-projects-in-visual-studio?view=msvc-170

**Target Keyword:**

Be sure to [review our style guide before writing](https://learn.draft.dev/content-management/styleguide).

**Writer Name:** Sarah Lean

**Writer Profile URL:** https://portal.draft.dev/writers/rece9JL2jq8tbTpiC

----- Article

# How to Install and Run CMake on Windows

CMake is an open source multiplatform tool that you can use to help with build automation, testing, packaging, and the installation of software. One of the main advantages of CMake is that it supports generating build scripts for a wide variety of platforms including Windows, macOS, and Linux. This gives developers the flexibility they need when building a new project and hopefully helps speed up the build.

In this article, you'll learn more about CMake and if it's the right solution for you. You'll also walk through the process of installing it on Windows.

## Pros and Cons of CMake

There are a number of benefits to using CMake to help build your solutions, including the fact that CMake can run on all major platforms, including Windows, Linux, and macOS. This means that a team of developers who are using different operating systems can still have common development tools.

Using CMake for projects, instead of something like [Visual Studio projects](https://learn.microsoft.com/visualstudio/ide/solutions-and-projects-in-visual-studio?view=vs-2022&WT.mc\_id=AZ-MVP-5004737#projects), helps you avoid being locked into one integrated development environment (IDE). A team can create a project using CMake, and then each developer can use their preferred IDE to work on it. CMake integration is supported by a variety of IDE tools, including [CLion](https://www.jetbrains.com/clion/), [Atom](https://atom.io/), and [Visual Studio](https://visualstudio.microsoft.com/).

It may be useful for some developers and software architects to understand what dependencies are within their projects and the nature of the dependency. CMake can create a visualization of any dependencies within the project, that you can use to create documentation.

While many would assume that CMake's active development is an advantage (which it is!), it also has its downsides: Not everyone within a developer team will be running the same version of CMake. However, to help reduce this issue, CMake has introduced policies that define how certain CMake features should behave. This means your team can set a common policy regardless of what version people are using.

Another disadvantage of CMake is the lack of information available for some versions. It can be hard to find accurate information to resolve an issue or answer a question.

## When to Use CMake

As you can see, CMake can be a useful build system. Developers using CMake can quickly check performance on different build backends and easily move between them.  
  
CMake can also help to keep your source code folder clean. It can separate the build system, intermediaries, and output files from the source code, keeping your code clean for the future.

## Install CMake on Windows

There are several ways to install CMake on Windows. The first option is to use [precompiled binaries](https://cmake.org/download/) as either the Windows Installer file (MSI) or the compressed version (ZIP) files. You just have to download the files and then walk through the interactive installation wizard.

The second option is to [download and build CMake from the source](https://cmake.org/download/), and CMake even includes a source release download option.

The third option is to use a package manager tool, such as [Windows Package Manager](https://learn.microsoft.com/training/modules/explore-windows-package-manager-tool/) or [Chocolatey](https://chocolatey.org/). This can be especially convenient if you already have either installed on your machine. We will use the package manager method you'll use for your installation here.

If you're using Windows Package Manager, run the following command to install CMake:

```bash

winget install kitware.cmake

```

If you're using Chocolatey, use the following command:

```bash

choco install cmake

```

### Install a CMake Extension with Visual Studio

Once you've finished installing CMake on Windows, you need to install an extension that lets you integrate the functionality of CMake into your IDE (Visual Studio is used here).

To install the CMake extension, you need to select \*\*Start\*\* and then open the \*\*Visual Studio Installer\*\* from the menu:

![\*\*Visual Studio Installer\*\*](https://i.imgur.com/rL2NEtk.png)

Once the \*\*Visual Studio Installer\*\* loads, select \*\*Modify\*\*:

![Modify your \*\*Visual Studio Installer\*\*](https://i.imgur.com/J0F3Zy4.png)

Under \*\*Workloads\*\*, locate \*\*Desktop development with C++\*\*, which can be found under \*\*Desktop & Mobile\*\*:

![\*\*Desktop development with C++\*\* component](https://i.imgur.com/N97KbHm.jpg)

On this same page, you also need to find and select the \*\*Linux and embedded development with C++\*\* component. This component ensures you have cross-platform CMake development capabilities:

![Linux development with C++ component](https://i.imgur.com/gw4mUcU.png)

Once you've selected both, select \*\*Modify\*\* again, and the additional tools will be installed.

### Integrate an IDE

Now that you've installed CMake and a few components, it's time to use them. Start a new project by opening Visual Studio and selecting \*\*Create a new project\*\* > \*\*CMake project\*\*:

![New project selection](https://i.imgur.com/tZE06Kr.png)

After selecting \*\*CMake project\*\*, Visual Studio, and CMake will create a directory and a file called [`CMakeLists.txt`](<https://www.jetbrains.com/help/clion/cmakelists-txt-file.html>), which is a set of instructions describing certain settings within the project, such as source files and targets.

When Visual Studio detects that there is a `CMakeLists.txt` file within a project that's been opened, it adds CMake items to the \*\*Project\*\* menu, which gives you access to commands for viewing and editing CMake scripts:

![\*\*Project\*\* menu](https://i.imgur.com/gDWPB2V.png)

### Configure CMake Projects

CMake supports two files that let you specify build and test options. These files are `CMakePresets.json` and `CMakeUserPresets.json`. [Microsoft recommends](https://learn.microsoft.com/cpp/build/cmake-presets-vs?view=msvc-170&WT.mc\_id=AZ-MVP-5004737) that you use `CMakePresets.json`.

To ensure that Visual Studio uses the `CMakePresets.json` file, you need to enable it. You can do so via \*\*Tools\*\* > \*\*Options\*\* > \*\*CMake\*\* > \*\*General\*\*. Please ensure that the \*\*Always use CMake Presets\*\* option is selected:

![Options menu settings](https://i.imgur.com/C0zS8rF.png)

If Visual Studio can't find a `CMakePresets.json` file, it will fall back to using the default [configure presets](https://learn.microsoft.com/cpp/build/cmake-presets-vs?view=msvc-170&WT.mc\_id=AZ-MVP-5004737#default-configure-presets).

There are three key settings within your `CMakePresets.json` file: target systems, [configure presets](https://cmake.org/cmake/help/latest/manual/cmake-presets.7.html#id6), and [build presets](https://cmake.org/cmake/help/latest/manual/cmake-presets.7.html#build-preset). The target system is the system on which CMake is invoked to configure and build your project. You can use systems installed on your local machine, SSH connections, and all Windows Subsystem for Linux (WSL) installations.

The configure preset option is used when CMake is invoked to generate the project build system. Depending on your target system, the options for this will change. So if you are using Ubuntu as your target system, your configure preset could be Linux debug.

The build preset value is used when CMake is invoked to build the project and should align with the configure preset. If Visual Studio can't find any build presets that are associated with the configure preset, then it will provide the default build preset.

As you can see, there are a lot of options available to configure your CMake projects to suit your needs and project. If you're looking for even more information on how to configure your projects, check out this [Visual Studio documentation](https://learn.microsoft.com/cpp/build/cmake-presets-vs?view=msvc-170&WT.mc\_id=AZ-MVP-5004737&WT.mc\_id=AZ-MVP-5004737#default-configure-presets).

### Build CMake Projects on Visual Studio

When you're ready, Visual Studio gives you several options to build your CMake projects, including the toolbar or the Solution Explorer.

If you want to use the toolbar, you need to find the \*\*Startup Item\*\* drop-down:

![\*\*Startup Item\*\* drop-down](https://i.imgur.com/cAePEXq.png)

Select your preferred target to build, and either press \*\*F5\*\* or choose \*\*Run\*\*.

If you want to use the Solution Explorer, navigate to the \*\*Solution Explorer\*\* on the right-hand side of your screen. By default, it will be set to the \*\*Folder View\*\*:

![\*\*Solution Explorer - Folder View\*\*](https://i.imgur.com/dqElsX9.png)

You need to change the \*\*Folder View\*\* to the \*\*CMake Targets View\*\*. To do this, select the view icon:

![View icon](https://i.imgur.com/Fy0Jy4m.png)

Then select \*\*CMake Targets View\*\*:

![\*\*CMake Targets View\*\*](https://i.imgur.com/FBAsxCY.png)

Then right-click on \*\*CMake Target\*\* and select \*\*Build\*\* from the context menu:

![\*\*Build\*\*](https://i.imgur.com/vuMsozD.png)

Lastly, you can select \*\*Build\*\* > \*\*Build All\*\* from the main menu of Visual Studio to build your CMake project:

![\*\*Build All\*\*](https://i.imgur.com/GcG7Y2o.png)

After completing either one of these options, you can find the build results within the

\*\*Output\*\* window and \*\*Error List\*\*:

![\*\*Output\*\* window](https://i.imgur.com/4JE6Fpn.png)

### Debug CMake Projects

As you know, when writing code, it's never perfect the first time, which is why you'll need to be able to debug your code.

To debug your CMake project using Visual Studio, start by selecting one of the targets shown in the \*\*Startup Item\*\* drop-down within the toolbar:

![\*\*Startup Item\*\* drop-down](https://i.imgur.com/cAePEXq.png)

Then select \*\*Debug\*\* > \*\*Start Debugging\*\* from the main toolbar:

![\*\*Debug\*\*](https://i.imgur.com/XcBC4Ta.png)

If any changes have been made since the last time you built the project, the debug command will build it first before the debugging can take place.

The [`launch.vs.json`](https://learn.microsoft.com/cpp/build/configure-cmake-debugging-sessions?view=msvc-170&WT.mc\_id=AZ-MVP-5004737#launchvsjson-reference) file can be used to customize your CMake debugging session. This file can be used to input any environment variables or command line arguments that you might want to pass into the program during debugging.

## Conclusion

In this article, you learned that CMake is a tool that can help with build automation, testing, packaging, and installation of software. It can be integrated into several IDE solutions, allowing developers the option of using their favorite IDE while taking advantage of CMake for their build solution.

CMake gives you the flexibility to ensure your application can be built and executed on any platform. While CMake might not be the right tool for everyone, for the right developer, team, or project, it can be a powerful solution.