

CMake

In software development, **CMake** is cross-platform free and open-source software for build automation, testing, packaging and installation of software by using a compiler-independent method.^[3] CMake is not a build system but rather it generates another system's build files. It supports directory hierarchies and applications that depend on multiple libraries. It is used in conjunction with native build environments such as Make, Qt Creator, Ninja, Android Studio, Apple's Xcode, and Microsoft Visual Studio. It has minimal dependencies, requiring only a C++ compiler on its own build system.

CMake is distributed as open-source software under permissive New BSD license.^[4]

Contents

History

Features

- Flexible project structure
- IDEs configuration support

Build process

- Types of build targets

Language

- Command syntax
- JSON strings

Internals

Modules & Tools

- CPack

Software built by CMake

- Open Source
- Scientific tools

Examples

- Hello World

See also

References

External links

CMake



<u>Developer(s)</u>	Andy Cedilnik, Bill Hoffman, Brad King, Ken Martin, Alexander Neundorf
<u>Initial release</u>	2000
<u>Stable release</u>	3.20.3 ^[1] / 27 May 2021
<u>Repository</u>	<u>gitlab.kitware.com/cmake</u> /cmake (<u>https://gitlab.kitware.com/cmake/cmake</u>)
<u>Written in</u>	<u>C</u> , <u>C++</u> ^[2]
<u>Operating system</u>	<u>Cross-platform</u>
<u>Type</u>	<u>Software development tools</u>
<u>License</u>	<u>New BSD</u>
<u>Website</u>	<u>cmake.org</u> (<u>https://cmake.org/</u>)

History

CMake development began in 1999 in response to the need for a cross-platform build environment for the [Insight Segmentation and Registration Toolkit](#).^[5] The project is funded by the [United States National Library of Medicine](#) as part of the [Visible Human Project](#). It was partially inspired by [pcmaker](#), which was made by Ken Martin and other developers to support the [Visualization Toolkit \(VTK\)](#). At [Kitware](#), Bill Hoffman blended components of [pcmaker](#) with his own ideas, striving to mimic the functionality of [Unix configure scripts](#). CMake was first implemented in 2000 and further developed in 2001.

Continued development and improvements were fueled by the incorporation of CMake into developers' own systems, including the [VXL Project](#), the [CABLE](#)^[6] features added by Brad King, and [GE Corporate R&D](#) for support of DART. Additional features were created when VTK transitioned to CMake for its build environment and for supporting [ParaView](#).

Version 3.0 was released in June 2014.^[7] It has been described as the beginning of "Modern CMake".^[8] Experts now advise to avoid variables in favor of *targets* and *properties*.^[9] The commands `add_compile_options`, `include_directories`, `link_directories`, `link_libraries` that were at the core of CMake 2 should now be replaced by target-specific commands.

Features

A key feature is the ability to (optionally) place compiler outputs (such as object files) outside the source tree. This enables multiple builds from the same source tree and [cross-compilation](#). It also keeps the source tree clean and ensures that removing a build directory will not remove the source files.

Flexible project structure

CMake can locate system-wide and user-specified executables, files, and libraries. These locations are stored in a [cache](#), which can then be tailored before generating the target build files. The cache can be edited with a graphical editor, which is shipped with the CMake.

Complicated directory hierarchies and applications that rely on several libraries are well supported by CMake. For instance, CMake is able to accommodate a project that has multiple toolkits, or libraries that each have multiple directories. In addition, CMake can work with projects that require executables to be created before generating code to be compiled for the final application. Its open-source, extensible design allows CMake to be adapted as necessary for specific projects.^[10]

IDEs configuration support

CMake can generate project files for several popular [IDEs](#), such as [Microsoft Visual Studio](#), [Xcode](#), and [Eclipse CDT](#). It can also produce build scripts for MSBuild or NMake on Windows; [Unix Make](#) on Unix-like platforms such as [Linux](#), [macOS](#), and [Cygwin](#); and [Ninja](#) on both Windows and Unix-like platforms.

Build process

The build of a program or library with CMake is a two stage process. First, standard build files are created (generated) from configuration files (`CMakeLists.txt`) which are written in [CMake language](#). Then the platform's native build tools (native toolchain) are used for actual building of programs.^{[10][11]}

The build files are configured depending on used generator (e.g. *Unix Makefiles* for `make`). Advanced users can also create and incorporate additional makefile generators to support their specific compiler and OS needs. Generated files are typically placed (by using `cmake`'s flag) into a folder outside of the sources one (out of source build), e.g., `build/`.

Each build project in turn contains a `CMakeCache.txt` file and `CMakeFiles` directory in every (sub-)directory of the projects (happened to be included by `add_subdirectory(...)` command earlier) helping to avoid or speed up regeneration stage once it's run over again.

Once the `Makefile` (or alternative) has been generated, build behavior can be fine-tuned via target properties (since version 3.1) or via `CMAKE_...`-prefixed global variables. The latter is discouraged for target-only configurations because variables are also used to configure `CMake` itself and to set up initial defaults.^[9]

Types of build targets

Depending on `CMakeLists.txt` configuration the build files may be either executables, libraries (e.g. `libxyz`, `xyz.dll` etc), `object file` libraries or pseudo-targets (including aliases). `Cmake` can produce object files that can be linked against by executable binaries/libraries avoiding dynamic (run-time) linking and using static (compile-time) one instead. This enables flexibility in configuration of various optimizations.^[12]

Build dependencies may be determined automatically.

Language

`CMake` has a relatively simple interpreted, imperative, scripting language. It supports variables, string manipulation, arrays, function/macro declarations, and module inclusion (`import`). `CMake` Language commands (or directives) are read by `cmake` from a file named `CMakeLists.txt`. This file specifies the source files and build parameters, which `cmake` will place in the project's build specification (such as a `Makefile`). Additionally, `.cmake`-suffixed files can contain scripts used by `cmake`.^[13]

To generate a project's build files, one invokes `cmake` in terminal and specifies the directory which contains `CMakeLists.txt`. This file contains one or more commands in the form of `COMMAND(argument ...)`.

Command syntax

The arguments of the commands are whitespace-separated and can include keywords to separate groups of arguments. Commands can take a keywords, for instance, in the command `SET_SOURCE_FILE_PROPERTIES(source_file ... COMPILE_FLAGS compiler_option ...)` the keyword is `COMPILE_FLAGS`. It can serve as a delimiter between list of source files and some other options.^[14]

Examples of commands that `cmake` includes to specify targets and its dependencies (to be built by the native toolchain) and which serve as starting point of the `CMakeLists.txt`:^{[15][16][17]}

- `add_executable(...)` — declares an executable binary target with sources (depend on language chosen) to be built
- `add_library(...)` — the same but for a library

- `target_link_libraries(...)` — adds dependencies etc.

JSON strings

Cmake supports extracting values into variables from the JSON-data strings (since version 3.19).^[18]

Internals

The executable programs CMake, CPack, and CTest are written in the C++ programming language.

Much of CMake's functionality is implemented in modules that are written in the CMake language.^[19]

Since release 3.0, CMake's documentation uses reStructuredText markup. HTML pages and man pages are generated by the Sphinx documentation generator.

Modules & Tools

CMake ships with numerous `.cmake` modules and tools. These facilitate work such as finding dependencies (FindXYZ modules), testing the toolchain environment and executables, packaging releases (CPack module and `cpack` command), and managing dependencies on external projects (ExternalProject module):^{[20][21]}

- **ctest** — is used for target testing commands specified by `CMakeLists.txt`
- **ccmake** and **cmake-gui** — tweaks and updates configuration variables intended for the native build system
- **cpack** — helps to package software

CPack

CPack is a packaging system for software distributions. It is tightly integrated with CMake but can function without it. ^{[22][23]}

It can be used to generate:

- Linux RPM, deb, and gzip packages (for both binaries and source code).
- NSIS files (for Microsoft Windows).
- macOS packages.

Software built by CMake

Open Source

List of the most notable examples of software built by using CMake includes:

- MySQL
- Boost (C++ libraries)
- KDE/KDE Plasma 5 — Desktop Environment for Linux-based systems

- [KiCAD](#)
- [FreeCAD](#)
- [Webkit](#)

Scientific tools

Notably, software used by the [ATLAS experiment](#) is also built by using CMake. The software itself is written in C/C++ and Python.^[24]

Examples

Hello World

The following source code files demonstrate how to build a simple [hello world](#) program written in C++ by using CMake. The source files are placed in a `src/` directory.

```
// src/Hello_world.cc
#include <iostream>

int main()
{
    std::cout << "Hello, world!\n";
}
```

```
# src/CMakeLists.txt
cmake_minimum_required(VERSION 3.10)

# set the project name
project("Hello World")

# specify the executable and corresponding source file
add_executable(hello "Hello_world.cc")
```

[bash](#) script to run CMake on a [Linux](#) system. This example assumes that the script will be kept next to the `src/` folder:

```
#!/usr/bin/env bash
# Place this file next to src/ folder

cmake -S src/ -B build/      # Request that outputs from the build be placed in the build/ folder
cmake --build build/        # Start the build
./build/hello                # Run the compiled program. Outputs "Hello, world!"
```

See also

- [List of build automation software § Build script generation](#)
- [configure script](#)
- [Monorepo](#)


References

1. "CMake 3.20.3 available for download" (<https://blog.kitware.com/cmake-3-20-3-available-for-do>)

[wnload/](#)).

2. "The CMake Open Source Project on OpenHub" (<https://www.openhub.net/p/cmake>). OpenHub. Retrieved 2016-04-09.
3. "CMake" (<https://cmake.org/>).
4. "Licenses · master · CMake / CMake" (<https://gitlab.kitware.com/cmake/cmake/-/tree/master/Licenses>). *GitLab*. Retrieved 2020-11-13.
5. "FLOSS Weekly 111: CMake" (<http://twit.tv/floss111>). *podcast*. TWiT Network. Retrieved 27 February 2011.
6. "The CABLE" (<https://web.archive.org/web/20130619224333/http://public.kitware.com/Cable/HTML/Index.html>). Archived from the original (<http://public.kitware.com/Cable/HTML/Index.html>) on 2013-06-19. Retrieved 2010-11-10.
7. Maynard, Robert (June 10, 2014). "[CMake] [ANNOUNCE] CMake 3.0.0 Released" (<https://cmake.org/pipermail/cmake/2014-June/057793.html>).
8. "Effective Modern CMake" (<https://gist.github.com/mbinna/c61dbb39bca0e4fb7d1f73b0d66a4fd1>). *Gist*.
9. https://github.com/boostcon/cppnow_presentations_2017/blob/master/05-19-2017_friday/effective_cmake_daniel_pfeifer_cppnow_05-19-2017.pdf, <https://gist.github.com/mbinna/c61dbb39bca0e4fb7d1f73b0d66a4fd1>
10. Neundorf, Alexander (2006-06-21). "Why the KDE project switched to CMake—and how" (<http://lwn.net/Articles/188693/>). *LWN.net*.
11. "cmake-toolchains(7) — CMake 3.19.0-rc2 Documentation" (<https://cmake.org/cmake/help/latest/manual/cmake-toolchains.7.html>). *cmake.org*. Retrieved 2020-10-29.
12. "cmake-buildsystem(7) — CMake 3.19.0-rc3 Documentation" (<https://cmake.org/cmake/help/latest/manual/cmake-buildsystem.7.html#object-libraries>). *cmake.org*. Retrieved 2020-11-14.
13. "cmake-language(7) — CMake 3.19.0-rc2 Documentation" (<https://cmake.org/cmake/help/latest/manual/cmake-language.7.html>). *cmake.org*. Retrieved 2020-10-29.
14. Andrej Cedilnik (2003-10-30). "Cross-Platform Software Development Using CMake Software" (<https://www.linuxjournal.com/article/6700>). *Linux Journal*. Retrieved 2021-01-29.
15. "add_executable — CMake 3.19.0-rc1 Documentation" (https://cmake.org/cmake/help/latest/command/add_executable.html). *cmake.org*. Retrieved 2020-10-25.
16. "add_library — CMake 3.19.0-rc1 Documentation" (https://cmake.org/cmake/help/latest/command/add_library.html). *cmake.org*. Retrieved 2020-10-25.
17. "target_link_directories — CMake 3.20.2 Documentation" (https://cmake.org/cmake/help/latest/command/target_link_directories.html). *cmake.org*. Retrieved 2021-05-10.
18. "CMake 3.19 Release Notes — CMake 3.19.7 Documentation" (<https://cmake.org/cmake/help/v3.19/release/3.19.html>). *cmake.org*. Retrieved 2021-03-15.
19. "cmake-language(7) — CMake 3.19.0-rc1 Documentation" (<https://cmake.org/cmake/help/latest/manual/cmake-language.7.html>). *cmake.org*. Retrieved 2020-10-25.
20. "cmake-modules(7) — CMake 3.14.7 Documentation" (<https://cmake.org/cmake/help/v3.14/manual/cmake-modules.7.html>). *cmake.org*. Retrieved 2020-10-24.
21. "ExternalProject — CMake 3.14.7 Documentation" (<https://cmake.org/cmake/help/v3.14/module/ExternalProject.html>). *cmake.org*. Retrieved 2020-10-24.
22. "Packaging With CPack" (<https://gitlab.kitware.com/cmake/community/-/wikis/doc/cpack/Packaging-With-CPack>). *CMake Community Wiki*.
23. `cpack(1)` (<https://www.mankier.com/1/cpack>) – *Linux General Commands Manual*
24. Elmsheuser, J; Krasznahorkay, A; Obreshkov, E; Undrus, A (2017). "Large Scale Software Building with CMake in ATLAS" (<https://cds.cern.ch/record/2243765/files/ATL-SOFT-PROC-2017-033.pdf>) (PDF). CERN.

External links

- [Official website \(https://cmake.org/\)](https://cmake.org/) 
 - [CMake \(https://github.com/Kitware/CMake\)](https://github.com/Kitware/CMake) on [GitHub](#)
 - [C++Now 2017: Daniel Pfeifer "Effective CMake" \(https://www.youtube.com/watch?v=bsXLMQ6WgIk\)](https://www.youtube.com/watch?v=bsXLMQ6WgIk) on [YouTube](#)
-

Retrieved from "<https://en.wikipedia.org/w/index.php?title=CMake&oldid=1025165221>"

This page was last edited on 26 May 2021, at 03:01 (UTC).

Text is available under the Creative Commons Attribution-ShareAlike License; additional terms may apply. By using this site, you agree to the Terms of Use and Privacy Policy. Wikipedia® is a registered trademark of the Wikimedia Foundation, Inc., a non-profit organization.