



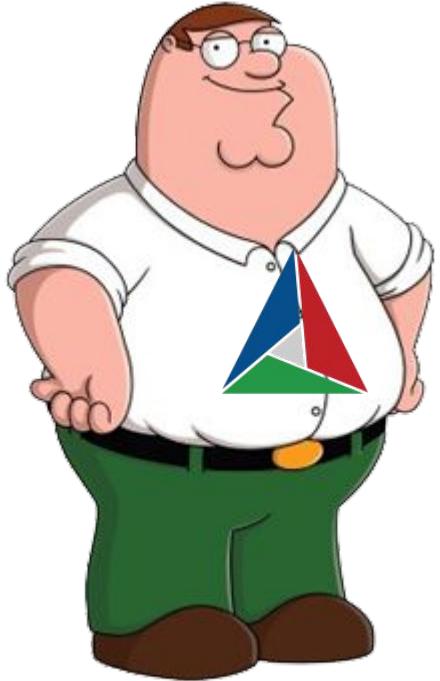
2025

import CMake; // Mastering C++ Modules

```
import std;
```

Bill Hoffman

CMake / Sandal / Kitware GUY



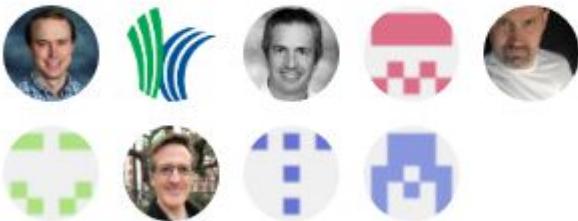
About Me

- 1990-1999 GE research in a Computer Vision group.
 - Moved them from Symbolics Lisp Machines to C++ on Solaris and later Linux/Windows
 - Was the build and software library guy (gmake/autotools)
- 1998-Present Kitware
 - Lots of development in CMake/ITK/VTK
 - Mostly management now, finding funding for CMake



CMake is a Community Effort

Contributors 971



modern cmake

Q: All Videos Shopping News Images More Settings Tools

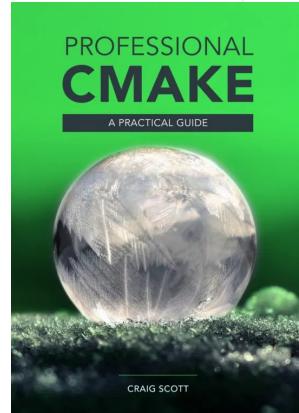
About 29,500 results (0.22 seconds)

[www.youtube.com/watch](https://www.youtube.com/watch?v=...)
More Modern CMake - Deniz Bahadir - Meeting C++ 2018 ...
More Modern CMake (Reupload with slide recording provided by speaker, thanks Deniz!Deniz ...
Feb 25, 2019 · Uploaded by Meeting Cpp
1:05:32

[www.youtube.com/watch](https://www.youtube.com/watch?v=...)
Oh No! More Modern CMake - Deniz Bahadir - Meeting C++ ...
Oh No! More Modern CMake - Deniz Bahadir - Meeting C++ 2019His CMake
Talk from last year: <https://...>
Jan 2, 2020 · Uploaded by Meeting Cpp
1:00:46

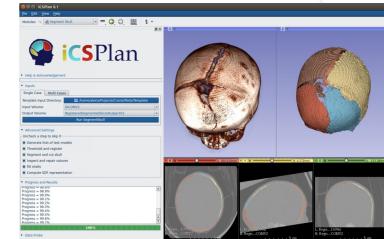
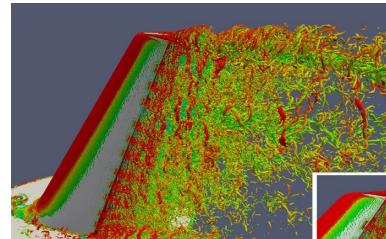
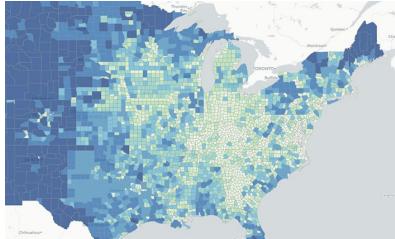
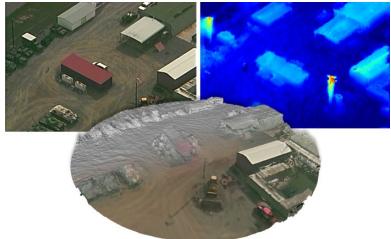
www.reddit.com/r/cpp/comments/azife1/modern_cmake/
Modern CMake Examples : cpp - Reddit
Modern CMake Examples ... IMHO the problem is CMake itself here, specifically ... you are already telling ...
Mar 10, 2019 · Uploaded by Meeting Cpp
49:52

[www.youtube.com/watch](https://www.youtube.com/watch?v=...)
CppCon 2017: Mathieu Ropert "Using Modern CMake ...
... Slides, PDFs, Source Code and other presenter materials are available at: <https://github.com/CppCon/> ...
Oct 13, 2017 · Uploaded by CppCon
57:40



Brad King, Ben Boeckel, Craig Scott are the CMake guys!

Kitware Overview / Built on open source



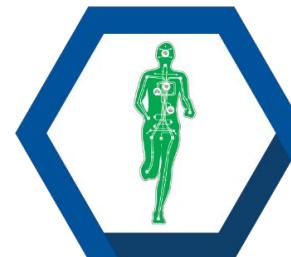
Computer
Vision



Data and
Analytics



Scientific
Computing



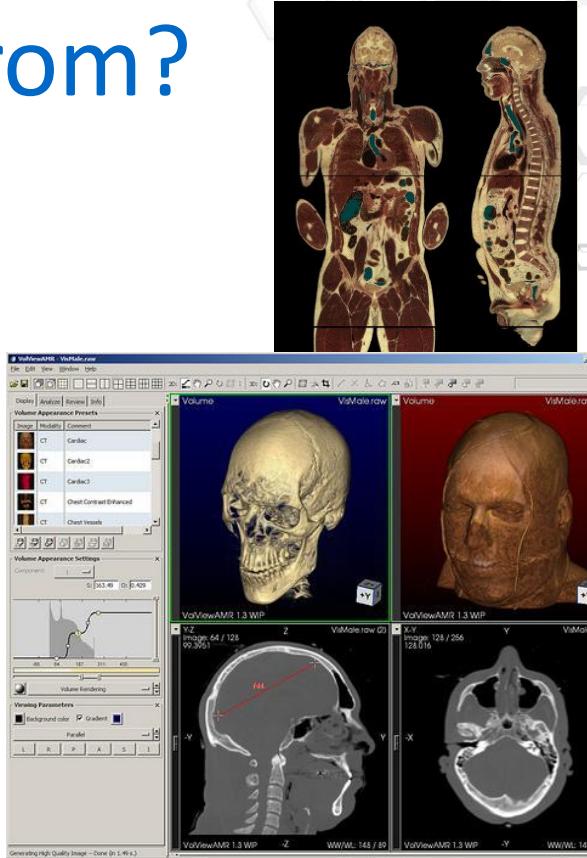
Medical
Computing



Software
Solutions

Where did CMake come from?

- Kitware was the lead engineering team for the Insight Segmentation and Registration Toolkit (ITK)
<http://www.itk.org>
 - tasked with making it build on Unix/Windows/Mac
- Funded by National Library of Medicine (NLM): part of the Visible Human Project
 - Data CT/MR/Slice 1994/1995
 - Code (ITK) 1999
- CMake Release-1-0 branch created in 2001
- Since then funding from many projects and outside contributions





Enough with the Background Let's Talk Modules!

C++ Named Modules Only

C++ Modules: The Challenges of Implementing Header Units

C++Now 2023
May 10, 2023

Daniel Ruoso
Software Engineering Manager, Code Governance

TechAtBloomberg.com

© 2023 Bloomberg Finance L.P. All rights reserved.

Bloomberg
Engineering



Does CMake support C++20 modules

Ask ChatGPT 2023



Yes, CMake has added support for C++20 modules in version 3.21. This means you can use CMake to build C++20 projects that make use of the new module system.



To use C++20 modules with CMake, you'll need to use a compiler that supports them, such as Clang 12 or later, or GCC 11 or later. You'll also need to specify the language standard and module generation options in your CMakeLists.txt file. Here's an example:

scss

Copy code

```
cmake_minimum_required(VERSION 3.21)

project(MyProject)

set(CMAKE_CXX_STANDARD 20)
set(CMAKE_CXX_STANDARD_REQUIRED ON)
set(CMAKE_CXX_EXTENSIONS OFF)

add_executable(MyExecutable main.cpp)

target_compile_features(MyExecutable PRIVATE cxx_std_20)
```



Yes, CMake does support C++20 modules, but the level of support can vary depending on the compiler and its version. As of CMake 3.19 and later, there are features to help you work with C++20 modules.

Ask ChatGPT 2024

To use C++20 modules in CMake, you typically need to:

1. Specify the C++

enable C++20 fe

2. Use the `CXX_M01`

the `MODULE` opti

3. Check compiler

C++20 modules.

may still be expe

4. Use module inte

your CMake com

For example:

```
cmake
cmake_minimum_required(VERSION 3.19)
project(MyProject LANGUAGES CXX)

set(CMAKE_CXX_STANDARD 20)

add_library(my_module MODULE my_module.cpp)
target_sources(my_module PRIVATE my_module.ixx)
```

Copy code

Make sure to consult the CMake documentation and the specific compiler documentation for details on any limitations or additional settings required for using C++20 modules effectively.

2025 Still No Luck ChatGPT

does CMake support C++20 Modules?

Example (MSVC, CMake 3.29+):

CMake has partial support for C++20 modules, but it's still evolving and compilers and platforms. Here's the current status as of 2025:

Supported (with caveats)

- Compilers:
 - Clang (>= 16) – experimental support
 - MSVC (Visual Studio 2022 and later) – best support so far
 - GCC – very limited or experimental support (mainly in GCC 13+)
- CMake:
 - Starting with CMake 3.28, there is experimental support for the `CMAKE_EXPERIMENTAL_CXX_MODULE_CMAKE_API` flag.
 - Modules must be declared using the `INTERFACE_SOURCES` and

```
c  
cmake_minimum_required(VERSION 3.29)  
project(MyProject LANGUAGES CXX)  
  
set(CMAKE_CXX_STANDARD 20)  
set(CMAKE_EXPERIMENTAL_CXX_MODULE_CMAKE_API "experimental")  
  
add_library(my_module_lib)  
target_sources(my_module_lib  
PRIVATE  
my_module.ixx  
my_module_impl.cpp  
)
```

Built Module Interface (BMI)

P1838R0: Modules User-Facing Lexicon and File Extensions
ISO/IEC JTC1 SC22/WG21 - Programming Languages - C++

- MSVC
 - .ifc file
- G++
 - .gcm file
- Clang
 - .pcm
- <http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2020/p1838r0.pdf>

Authors:

Bryce Adelstein Lelbach <brycelelbach@gmail.com>
Boris Kolpackov <boris@codesynthesis.com>

Audience:

Tooling (SG15)

Motivation

C++20 modules introduces a new compilation model for C++; as with any new large feature, we need a number of new words to discuss it. This paper seeks to define and bikeshed a user-facing lexicon for modules.

Simple Example (build order order matters)

```
B.cpp:
```

```
export module B;  
export void b() { }
```

```
A.cpp:
```

```
export module A;  
import B;  
export void a(){ b();}
```

```
cl -std:c++20 -interface -c A.cpp
```

```
A.cpp
```

```
A.cpp(2): error C2230: could not find module 'B'  
A.cpp(3): error C3861: 'b': identifier not found
```

Simple Example (build order matters)

```
B.cpp:
```

```
export module B;  
export void b() { }
```

```
A.cpp:
```

```
export module A;  
import B;  
export void a(){ b();}
```

```
cl -std:c++20 -interface -c B.cpp  
B.cpp  
  
cl -std:c++20 -interface -c A.cpp  
A.cpp  
$ ls  
A.cpp A.ifc A.obj B.cpp B.ifc B.obj
```

Chicken and the Egg

- Modules require the build system to know which files produce which BMI files and which files consume them
- Need to parse/compile file to find that out
- So... We need to compile the code before we can compile the code....

CMake had 17 years experience with modules, Fortran ones

- 2005 Initial makefile support for modules added to CMake:

commit 19f977bad7261d9e8f8d6c5d2764c079d35cc014

Author: Brad King <brad.king@kitware.com>

Date: Wed Jan 26 15:33:38 2005 -0500

ENH: Added Fortran dependency scanner implementation.

- Added support to ninja in 2015 for Fortran dep file depends - funded by the Trilinos project forked ninja
 - dyndep https://ninja-build.org/manual.html#ref_dyndep
- May 2019 ninja merged all of the changes to support Fortran because of C++ modules!

https://ninja-build.org/manual.html#ref_dyndep

Some use cases require implicit dependency information to be dynamically discovered from source file content *during the build* in order to build correctly on the first run (e.g., Fortran module dependencies). This is unlike [header dependencies](#) which are only needed on the second run and later to rebuild correctly.

How does CMake do this

- A Fortran parser based off of makedepf90
 - For obvious reasons not a good idea for C++
 - must use the C++ compiler itself
- Patches made to ninja build tool now upstreamed
- The dynamic dependency collator inside CMake

Feb 2019

Paper Describing CMake Fortran Modules

[Tooling] [D1483] How CMake supports Fortran modules and its applicability to C++

Ben Boeckel [ben.boeckel at kitware.com](mailto:ben.boeckel@kitware.com)

Fri Feb 8 16:55:20 CET 2019

- Previous message: [\[Tooling\] Clang Modules and build system requirements](#)
- Next message: [\[Tooling\] Fwd: \[D1483\] How CMake supports Fortran modules and its applicability to C++](#)
- **Messages sorted by:** [\[date \]](#) [\[thread \]](#) [\[subject \]](#) [\[author \]](#)

Hi,

Here is copy of Kitware's paper to be discussed at Kona. I have a PDF, but it was too large to attach to the list. I'll be at Kona, but the other authors are not able to make it.

An HTML version is hosted here:

<https://mathstuf.fedorapeople.org/fortran-modules/fortran-modules.html>

Feedback welcome.

Thanks,

--Ben

<https://mathstuf.fedorapeople.org/fortran-modules/fortran-modules.html>

<https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2022/p1689r5.html>

defining a format for compilers p1689



Format for describing dependencies of source files

Ben Boeckel, Brad King

<ben.boeckel@kitware.com, brad.king@kitware.com>

version P1689R5, 2022-06-03

Table of Contents

- [1. Abstract](#)
- [2. Changes](#)
 - [2.1. R5 \(pending\)](#)
 - [2.2. R4 \(June 2021 mailing\)](#)
 - [2.3. R3 \(Dec 2020 mailing\)](#)
 - [2.4. R2 \(pre-Prague\)](#)
 - [2.5. R1 \(post-Cologne\)](#)
 - [2.6. R0 \(Initial\)](#)
- [3. Introduction](#)
- [4. Motivation](#)
 - [4.1. Why Makefile snippets don't work](#)
- [5. Assumptions](#)
- [6. Format](#)
 - [6.1. Schema](#)
 - [6.2. Storing binary data](#)
 - [6.3. Filepaths](#)
 - [6.4. Rule items](#)
 - [6.5. Module dependency information](#)
 - [6.5.1. Language-specific notes](#)
 - [Fortran](#)
 - [C++](#)
 - [6.6. Extensions](#)
- [7. Versioning](#)
- [8. Full example](#)
- [9. References](#)

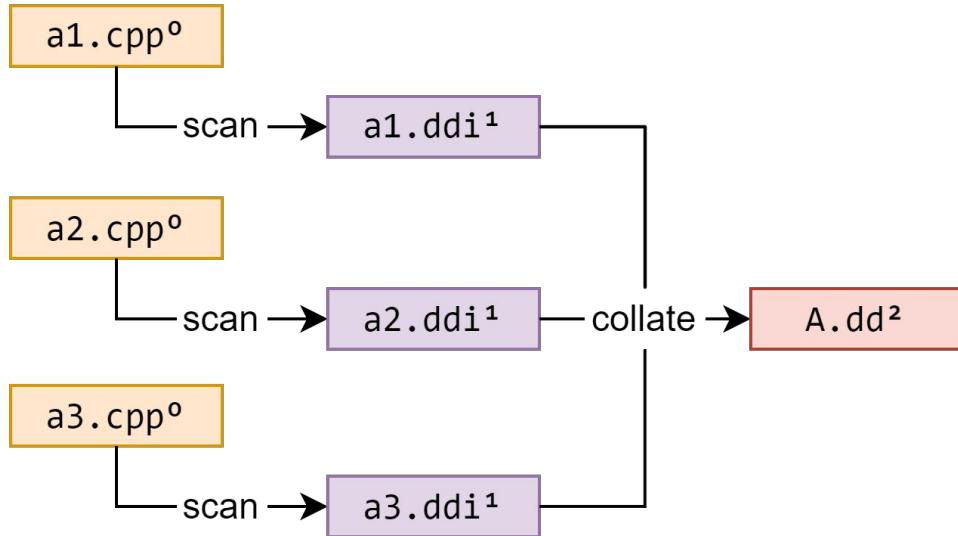
Document number ISO/IEC/JTC1/SC22/WG21/P1689R5

Date 2022-06-03

Reply-to Ben Boeckel, Brad King, ben.boeckel@kitware.com, brad.king@kitware.com

Audience EWG (Evolution), SG15 (Tooling)

Per target scanning



scan: done by compiler (i.e. `cl -scanDependencies`)

collate: `cmake -E cmake_ninja_dyndep`

Where we are today

- Support for p1689r5 in compiler releases
 - Visual studio 2022
 - Clang 16 and newer
 - GCC 14
- CMake 3.28 support for named modules
- CMake 3.30-4.0 experimental support for import std

CMake (Module able) Generators

- -GNinja
- -G"Visual Studio 17 2022" (aka MSBuild)

MSVC scan A.cpp

```
cl -std:c++20 -scanDependencies A.json -c A.cpp

{
    "version": 1,
    "revision": 0,
    "rules": [
        {
            "primary-output": "A.obj",
            "outputs": [
                "A.json",
                "A.ifc"
            ],
            "provides": [
                {
                    "logical-name": "A",
                    "source-path": "c:\\users\\hoffman\\work\\cxxmodules\\cxx-modules-examples\\simple\\a.cpp"
                }
            ],
            "requires": [
                {
                    "logical-name": "B"
                }
            ]
        }
    ]
}
```

MSVC scan B.cpp

```
cl -std:c++20 -scanDependencies B.json -c B.cpp

{
    "version": 1,
    "revision": 0,
    "rules": [
        {
            "primary-output": "B.obj",
            "outputs": [
                "B.json",
                "B.ifc"
            ],
            "provides": [
                {
                    "logical-name": "B",
                    "source-path": "c:\\users\\hoffman\\work\\cxxmodules\\cxx-modules-examples\\simple\\b.cpp"
                }
            ],
            "requires": []
        }
    ]
}
```

FILE_SET

File Sets

New in version 3.23.

```
target_sources(<target>
  [<INTERFACE|PUBLIC|PRIVATE>
    [FILE_SET <set> [TYPE <type>] [BASE_DIRS <dirs>...] [FILES <files>...]]...
  ]...)
```

Adds a file set to a target, or adds files to an existing file set. Targets have zero or more named file sets. Each file set has a name, a type, a scope of `INTERFACE`, `PUBLIC`, or `PRIVATE`, one or more base directories, and files within those directories. The only acceptable type is `HEADERS`. The optional default file sets are named after their type. The target may not be a custom target or `FRAMEWORK` target.

Header Only Libraries (with FILE_SET)

```
add_library(Eigen INTERFACE

target_sources(Eigen INTERFACE
  FILE_SET HEADERS
  BASE_DIRS src
  FILES src/eigen.h src/vector.h src/matrix.h
)

install(TARGETS Eigen EXPORT eigenExport
  FILE_SET HEADERS DESTINATION include/Eigen)
install(EXPORT eigenExport NAMESPACE Upstream::
  DESTINATION lib/cmake/Eigen
)
add_executable(exe1 exe1.cpp)
target_link_libraries(exe1 Eigen)
```

Running Install

```
# running this
cmake.exe --install-prefix=$HOME/Work/cxxmodules/file_set/b/inst .
-- Configuring done
-- Generating done
-- Build files have been written to: C:/Users/hoffman/Work/cxxmodules/file_set/b

$ ninja install
[0/1] Install the project...
-- Install configuration: "Debug"
-- Installing: C:/Users/hoffman/Work/cxxmodules/file_set/b/inst/include/Eigen/eigen.h
-- Installing: C:/Users/hoffman/Work/cxxmodules/file_set/b/inst/include/Eigen/vector.h
-- Installing: C:/Users/hoffman/Work/cxxmodules/file_set/b/inst/include/Eigen/matrix.h
-- Installing: C:/Users/hoffman/Work/cxxmodules/file_set/b/inst/lib/cmake/Eigen/eigenExport.cmake
```

File Sets

New in version 3.23.

```
target_sources(<target>
[<INTERFACE|PUBLIC|PRIVATE>
 [FILE_SET <set> [TYPE <type>] [BASE_DIRS <dirs>...] [FILES <files>...]]...
]...)
```

Adds a file set to a target, or adds files to an existing file set. Targets have zero or more named file sets. Each file set has a name, a type, a scope of `INTERFACE`, `PUBLIC`, or `PRIVATE`, one or more base directories, and files within those directories. The acceptable types include:

HEADERS

Sources intended to be used via a language's `#include` mechanism.

CXX_MODULES

New in version 3.28.

Sources which contain C++ interface module or partition units (i.e., those using the `export` keyword). This file set type may not have an `INTERFACE` scope except on `IMPORTED` targets.

Compile A.cpp and B.cpp with CMake

```
cmake_minimum_required(VERSION 3.23)
project(simple CXX)
set(CMAKE_CXX_STANDARD 20)
add_library(simple)

target_sources(simple
  PRIVATE
    FILE_SET cxx_modules TYPE CXX_MODULES FILES
      A.cpp B.cpp
)
```

Compile A.cpp and B.cpp configure with CMake

```
cmake -GNinja ..  
-- The CXX compiler identification is Clang 20.0.0  
-- Detecting CXX compiler ABI info  
-- Detecting CXX compiler ABI info - done  
-- Check for working CXX compiler: /Users/hoffman/Work/llvm/llvm-inst/bin/clang++ - skipped  
-- Detecting CXX compile features  
-- Detecting CXX compile features - done  
-- Configuring done (0.8s)  
-- Generating done (0.0s)  
-- Build files have been written to: /Users/hoffman/Work/modules/simple/b
```

Compile A.cpp and B.cpp with CMake Clang/ninja

```
$hoffman@caprica b % ninja -v
[1/10] "/Users/hoffman/Work/llvm/llvm-inst/bin/clang-scan-deps" -format=p1689 -- /Users/hoffman/Work/llvm/llvm-inst/bin/clang++ -std=gnu++20 -arch arm64 -isysroot /Library/Developer/CommandLineTools/SDKs/MacOSX14.4.sdk -x c++ /Users/hoffman/Work/modules/simple/main.cxx -c -o CMakeFiles/hello.dir/main.cxx.o -resource-dir "/Users/hoffman/Work/llvm/llvm-inst/lib/clang/20" -MT CMakeFiles/hello.dir/main.cxx.o.ddi -MD -MF CMakeFiles/hello.dir/main.cxx.o.ddi.d > CMakeFiles/hello.dir/main.cxx.o.ddi.tmp && mv CMakeFiles/hello.dir/main.cxx.o.ddi.tmp CMakeFiles/hello.dir/main.cxx.o.ddi
[2/10] "/Users/hoffman/Work/llvm/llvm-inst/bin/clang-scan-deps" -format=p1689 -- /Users/hoffman/Work/llvm/llvm-inst/bin/clang++ -std=gnu++20 -arch arm64 -isysroot /Library/Developer/CommandLineTools/SDKs/MacOSX14.4.sdk -x c++ /Users/hoffman/Work/modules/simple/A.cpp -c -o CMakeFiles/foo.dir/A.cpp.o -resource-dir "/Users/hoffman/Work/llvm/llvm-inst/lib/clang/20" -MT CMakeFiles/foo.dir/A.cpp.o.ddi -MD -MF CMakeFiles/foo.dir/A.cpp.o.ddi.d > CMakeFiles/foo.dir/A.cpp.o.ddi.tmp && mv CMakeFiles/foo.dir/A.cpp.o.ddi.tmp CMakeFiles/foo.dir/A.cpp.o.ddi
[3/10] "/Users/hoffman/Work/llvm/llvm-inst/bin/clang-scan-deps" -format=p1689 -- /Users/hoffman/Work/llvm/llvm-inst/bin/clang++ -std=gnu++20 -arch arm64 -isysroot /Library/Developer/CommandLineTools/SDKs/MacOSX14.4.sdk -x c++ /Users/hoffman/Work/modules/simple/B.cpp -c -o CMakeFiles/foo.dir/B.cpp.o -resource-dir "/Users/hoffman/Work/llvm/llvm-inst/lib/clang/20" -MT CMakeFiles/foo.dir/B.cpp.o.ddi -MD -MF CMakeFiles/foo.dir/B.cpp.o.ddi.d > CMakeFiles/foo.dir/B.cpp.o.ddi.tmp && mv CMakeFiles/foo.dir/B.cpp.o.ddi.tmp CMakeFiles/foo.dir/B.cpp.o.ddi
[4/10] "/Users/hoffman/Work/My Builds/cmake-ninja/bin/cmake" -E cmake_ninja_dyndep --tdi=CMakeFiles/foo.dir/CXXDependInfo.json --lang=CXX --modmapfmt=clang --dd=CMakeFiles/foo.dir/CXX.dd @CMakeFiles/foo.dir/CXX.dd.rsp
[5/10] "/Users/hoffman/Work/My Builds/cmake-ninja/bin/cmake" -E cmake_ninja_dyndep --tdi=CMakeFiles/hello.dir/CXXDependInfo.json --lang=CXX --modmapfmt=clang --dd=CMakeFiles/hello.dir/CXX.dd @CMakeFiles/hello.dir/CXX.dd.rsp
[6/10] /Users/hoffman/Work/llvm/llvm-inst/bin/clang++ -std=gnu++20 -arch arm64 -isysroot /Library/Developer/CommandLineTools/SDKs/MacOSX14.4.sdk -MD -MT CMakeFiles/foo.dir/B.cpp.o -MF CMakeFiles/foo.dir/B.cpp.o.d @CMakeFiles/foo.dir/B.cpp.o.modmap -o CMakeFiles/foo.dir/B.cpp.o -c /Users/hoffman/Work/modules/simple/B.cpp
[7/10] /Users/hoffman/Work/llvm/llvm-inst/bin/clang++ -std=gnu++20 -arch arm64 -isysroot /Library/Developer/CommandLineTools/SDKs/MacOSX14.4.sdk -MD -MT CMakeFiles/foo.dir/A.cpp.o -MF CMakeFiles/foo.dir/A.cpp.o.d @CMakeFiles/foo.dir/A.cpp.o.modmap -o CMakeFiles/foo.dir/A.cpp.o -c /Users/hoffman/Work/modules/simple/A.cpp
[8/10] /Users/hoffman/Work/llvm/llvm-inst/bin/clang++ -std=gnu++20 -arch arm64 -isysroot /Library/Developer/CommandLineTools/SDKs/MacOSX14.4.sdk -MD -MT CMakeFiles/hello.dir/main.cxx.o -MF CMakeFiles/hello.dir/main.cxx.o.d @CMakeFiles/hello.dir/main.cxx.o.modmap -o CMakeFiles/hello.dir/main.cxx.o -c /Users/hoffman/Work/modules/simple/main.cxx
[9/10] : && "/Users/hoffman/Work/My Builds/cmake-ninja/bin/cmake" -E rm -f libfoo.a && /usr/bin/ar qc libfoo.a CMakeFiles/foo.dir/A.cpp.o CMakeFiles/foo.dir/B.cpp.o && /Users/hoffman/Work/llvm/llvm-inst/bin/llvm-ranlib libfoo.a && "/Users/hoffman/Work/My Builds/cmake-ninja/bin/cmake" -E touch libfoo.a && :
[10/10] : && /Users/hoffman/Work/llvm/llvm-inst/bin/clang++ -arch arm64 -isysroot /Library/Developer/CommandLineTools/SDKs/MacOSX14.4.sdk -Wl,-search_paths_first -Wl,-headerpad_max_install_names CMakeFiles/hello.dir/main.cxx.o -o hello libfoo.a && :
```

Installing Targets

```
install(TARGETS targets... [EXPORT <export-name>]
        [RUNTIME_DEPENDENCIES args...|RUNTIME_DEPENDENCY_SET <set-name>]
        [ [ARCHIVE|LIBRARY|RUNTIME|OBJECTS|FRAMEWORK|BUNDLE|
            PRIVATE_HEADER|PUBLIC_HEADER|RESOURCE|FILE_SET <set-name>|CXX_MODULES_BMI]
        [DESTINATION <dir>]
        [PERMISSIONS permissions...]
        [CONFIGURATIONS [Debug|Release|...]]
        [COMPONENT <component>]
        [NAMELINK_COMPONENT <component>]
        [OPTIONAL] [EXCLUDE_FROM_ALL]
        [NAMELINK_ONLY|NAMELINK_SKIP]
    ] [...]
    [INCLUDES DESTINATION [<dir> ...]]
)
```

CXX_MODULES_BMI

New in version 3.28.

Any module files from C++ modules from `PUBLIC` sources in a file set of type `CXX_MODULES` will be installed to the given `DESTINATION`. All modules are placed directly in the destination as no directory structure is derived from the names of the modules. An empty `DESTINATION` may be used to suppress installing these files (for use in generic code).

Only pay for scanning if modules

`CXX_SCAN_FOR_MODULES`

New in version 3.28.

Must be on for
any file that has
import or export

`CXX_SCAN_FOR_MODULES` is a boolean specifying whether CMake will scan C++ sources in the target for module dependencies. See also the `CXX_SCAN_FOR_MODULES` for per-source settings which, if set, overrides the target-wide settings.

This property is initialized by the value of the `CMAKE_CXX_SCAN_FOR_MODULES` variable if it is set when a target is created.

When this property is set `ON` or unset, CMake will scan the target's `cxx` sources at build time and add module dependency information to the compile line as necessary. When this property is set `OFF`, CMake will not scan the target's `cxx` sources at build time.

Note that scanning is only performed if C++20 or higher is enabled for the target. Scanning for modules in the target's sources belonging to file sets of type `cxx_MODULES` is always performed.

```
export(TARGETS <target>... [NAMESPACE <namespace>]
      [APPEND] FILE <filename> [EXPORT_LINK_INTERFACE_LIBRARIES]
      [CXX_MODULES_DIRECTORY <directory>])
```

Creates a file `<filename>` that may be included by outside projects to import targets named by `<target>...` from the current project's build tree. This is useful during cross-compiling to build utility executables that can run on the host platform in one project and then import them into another project being compiled for the target platform.

The file created by this command is specific to the build tree and should never be installed. See the [install\(EXPORT\)](#) command to export targets from an install tree.

The options are:

`NAMESPACE <namespace>`

Prepend the `<namespace>` string to all target names written to the file.

`APPEND`

Append to the file instead of overwriting it. This can be used to incrementally export multiple targets to the same file.

`EXPORT_LINK_INTERFACE_LIBRARIES`

Include the contents of the properties named with the pattern `(IMPORTED_)?LINK_INTERFACE_LIBRARIES(_<CONFIG>)?` in the export, even when policy [CMP0022](#) is NEW. This is useful to support consumers using CMake versions older than 2.8.12.

`CXX_MODULES_DIRECTORY <directory>`

New in version 3.28.

Export C++ module properties to files under the given directory. Each file will be named according to the target's export name (without any namespace). These files will automatically be included from the export file.

Anatomy of a named module

```
module; // Global module fragment start
// If a module-unit has a global module fragment, then its first declaration must be
// module;. Then, only preprocessing directives can appear in the global module fragment.
// Then, a standard module declaration marks the end of the global module fragment and
// the start of the module content.
#define TURN_ON_FOO
#include <stdio.h>

export module A; // declare the module A after here, it's all modules
#include <foo.h>
export void A() { // do A stuff }
```

Global module fragment

- NOTE: Any headers included here must be PUBLIC because the module will need to be used to create a BMI by consumers.

Know experimental.rst

v4.0.1 ▾

cmake / Help / dev / **experimental.rst**

Find file

Blame

Permalink



ctest: Include cmake instrumentation data in XML files

Zack Galbreath authored 2 months ago and Brad King committed 2 months ago

a6d4a9a2



History

experimental.rst 5.18 KiB

</>



Edit ▾

Replace

Delete



CMake Experimental Features Guide

The following is a guide to CMake experimental features that are under development and not yet included in official documentation. See documentation on [CMake Development](#) for more information.

Features are gated behind `CMAKE_EXPERIMENTAL_` variables which must be set to specific values in order to enable their gated behaviors. Note that the specific values will change over time to reinforce their experimental nature. When used, a warning will be generated to indicate that an experimental feature is in use and that the affected behavior in the project is not part of CMake's stability guarantees.

Get a new enough compiler

- Most Recent Visual Studio Preview
- Clang origin/main - be ready to build some C++

```
%cmake -DLLVM_ENABLE_PROJECTS=clang  
-DCMAKE_INSTALL_PREFIX=/Users/hoffman/Work/llvm-main-inst  
-DCMAKE_BUILD_TYPE=Release _DLLVM_ENABLE_RUNTIMES="libcxx;libcxxabi;libc;libunwind"  
-G Ninja .. llvm-project/llvm  
%ninja
```

- Get CMake 4.0.1

Setup an env

```
export
```

```
CC=/Users/hoffman/Work/llvm/llvm-inst/bin/clang
```

```
export
```

```
CXX=/Users/hoffman/Work/llvm/llvm-inst/bin/clang++
```

```
export
```

```
CMAKE_INSTALL_PREFIX=/Users/hoffman/Work/modules/install
```

Are We Modules Yet?

arewemodulesyet.org

Are We Modules Yet? ☀️

Tools Support Docs Examples Contribute on Github ↗ Follow me ↗

Nope.

This website tracks the most popular C++ libraries, compilers, and build systems. We use vcpkg to get a rough idea of how popular a library is based on how many revisions the port has. We also have an example section to track good projects using modules natively and a document section tracks good articles about modules too. Feel free to contribute to these lists.



C++ NEEDS YOU!
Help Other Open Source Projects Support C++20 Modules!

Estimated finish by: Wed Jun 14 2626

There is a progress bar (promise)



17 / 2431 (0.70%) Confirmed Complete
Current rate: 3.95 projects/year

List Of Libraries:

Status

?

 Default: No set. Help wanted for every lib that has this status!

Full support

Worked on

Reported

Corresponding tracking issue was closed and no planned support anytime soon.

Needs support old library/compiler/build tool to even start adding support for modules.

Maintainer does want to keep it oldschoold. No plan to update anytime soon.

Help Wanted

Yes

No

Does not allow outside contributions

Popularity based on vcpkg port update: Generated Date: 2 May 2024 Count parsed from [this git commit](#).

Search for names..

?

Status
Help Wanted

Status	Help Wanted	Name	Import Statement	Popularity	Version	Min C++ Version	Modules Support Since	Tracking Issue
<input checked="" type="checkbox"/>	<input type="checkbox"/> ?	fmt	import fmt;	60	10.2.1	C++11	2022/10/11	🔗
<input checked="" type="checkbox"/>	<input type="checkbox"/> ?	glm	import glm;	27	1.0.1	C++11	2023/12/30	🔗
<input checked="" type="checkbox"/>	<input type="checkbox"/> ?	magic-enum	import magic_enum;	21	0.9.5	C++17	2024/5/9	🔗
<input checked="" type="checkbox"/>	<input type="checkbox"/> ?	tgui	import tgui;	16	1.1.0		2024/1/13	
<input checked="" type="checkbox"/>	<input type="checkbox"/> ?	boost-pfr	import Boost.PFR;	12	1.84.0		2024/08/01	🔗
<input checked="" type="checkbox"/>	<input type="checkbox"/> ?	ctre	import ctre;	11	3.8.1	C++17	2024/5/17	

<input checked="" type="checkbox"/>	?	argparse	<pre>import argparse;</pre>	10	3.0	C++17	2023/10/15	
<input checked="" type="checkbox"/>	?	fastgltf	<pre>import fastgltf;</pre>	8	0.7.1	C++17	2024/5/15	
<input checked="" type="checkbox"/>	?	vulkan-hpp	<pre>import vulkan_hpp;</pre>	8	deprec...	C++11	2020/6/23	
<input checked="" type="checkbox"/>	?	mp-units	<pre>import mp_units;</pre>	6	2.1.0		2024/1/6	
<input checked="" type="checkbox"/>	?	vulkan-memory-allocator-hpp	<pre>import vk_mem_alloc_hpp;</pre>	3	3.0.1.1	C++20	2023/9/11	
<input checked="" type="checkbox"/>	?	async-simple	<pre>import async_simple;</pre>	2	1.3	C++20	2023/6/19	
<input checked="" type="checkbox"/>	?	flux	<pre>import flux;</pre>	1	Unkno...	C++20	2023/8/3	
<input checked="" type="checkbox"/>	?	vladimirshaleev-ipaddress	<pre>import ipaddress;</pre>	1	1.0.1		2024/6/2	
<input checked="" type="checkbox"/>	?	z4kn4fein-semver	<pre>import semver;</pre>	1	0.2.1	C++17	2024/8/21	
<input checked="" type="checkbox"/>	?	seastar	<pre>import seastar;</pre>			C++20	2023/3/24	
<input checked="" type="checkbox"/>	?	cpp-lazy	<pre>import lz;</pre>			C++11	2024/4/9	

OK, so your telling me there's a chance

17 libraries with module
support, AWESOME!

Let's build them



Adventures with CTRE!

<https://github.com/hanickadot/compile-time-regular-expressions>



Follow

hanickadot Hana Dusíková · she/her

Staff Software Engineer in Woven by Toyota, Chair of Study Group for Reflection & Co-Chair of Evolution Working Group in C++ committee



📍 Brno, Czech Republic

💻 Owns this repository

-⌚ Committed to this repository in the past day



Ben Boeckel
mathstuf

CTRE no install for modules :(

- Just add a hello executable into the CTRE project cmake file itself to test
- Didn't work
- Could not find the module ctre
- CXX_SCAN_FOR_MODULES was the culprit!

5 CMakeLists.txt

```
... ... @@ -1,4 +1,4 @@  
1 - cmake_minimum_required(VERSION 3.14.0)  
1 + cmake_minimum_required(VERSION 3.14...3.29)  
2 2  
3 3 + if(CMAKE_VERSION VERSION_GREATER_EQUAL "3.29.20240416")  
4 4         set(CMAKE_EXPERIMENTAL_CXX_IMPORT_STD "0e5b6991-d74f-4b3d-a41c-cf096e0b2508")  
...  
@@ -92,7 +92,7 @@ if(CTRE_MODULE)
```

hanickadot commented 2 days ago

why is there maximum version of 3.29?



mathstuf commented 2 days ago

That is not a "maximum version". It is a "maximum tested version". It is saying "CMake 3.14 is required, but the project has been tested against all new policy behaviors up to 3.29 and accepts the new behaviors".



```
std;  
LESS 23  
XX_STANDARD OR CTRE_CXX_STANDARD LESS 23  
TANDARD 23)
```

```
113 113  
114 114  
115 115         target_compile_features(${PROJECT_NAME} INTERFACE cxx_std_${CTRE_CXX_STANDARD})  
116 116 +         set_property(TARGET ${PROJECT_NAME} PROPERTY CXX_SCAN_FOR_MODULES 0)  
117 117         install(TARGETS ${PROJECT_NAME} EXPORT ${PROJECT_NAME}-targets)  
118 119 endif()
```

Let's get it to install

Install module rules #323

Merged hanickadot merged 3 commits into hanickadot:main from mathstuf:install-module-rules 32 minutes ago

Conversation 1 Commits 3 Checks 62 Files changed 1

Changes from all commits File filter Conversations Jump to Review

0 / 1 files viewed

26 CMakeLists.txt

```
@@ -38,7 +38,10 @@ if(CTRE_MODULE)
 38   38         add_library(${PROJECT_NAME})
 39   39
 40   40         target_sources(${PROJECT_NAME} PUBLIC FILE_SET CXX_MODULES TYPE CXX_MODULES FILES ctre.cppm)
 41 -        target_sources(${PROJECT_NAME} PUBLIC FILE_SET HEADERS TYPE HEADERS FILES
 41 +        target_sources(${PROJECT_NAME} PUBLIC FILE_SET HEADERS TYPE HEADERS
 42 +        BASE_DIRS
 43 +        "${CMAKE_CURRENT_SOURCE_DIR}/include"
 44 +        FILES
 42   45             include/ctll.hpp
 43   46             include/ctre/functions.hpp
 44   47             include/ctre/utility.hpp
 45
 46
 47
 48
 49
 50
 51
 52
 53
 54
 55
 56
 57
 58
 59
 60
 61
 62
 63
 64
 65
 66
 67
 68
 69
 70
 71
 72
 73
 74
 75
 76
 77
 78
 79
 80
 81
 82
 83
 84
 85
 86
 87
 88
 89
 90
 91
 92
 93
 94
 95
 96   99         set(CTRE_CXX_STANDARD 23)
 97   100        endif()
 98   101
 99 -        target_include_directories(${PROJECT_NAME} PRIVATE ${CMAKE_CURRENT_SOURCE_DIR}/include)
100  102        target_compile_features(${PROJECT_NAME} PUBLIC cxx_std_${CTRE_CXX_STANDARD})
101
102
103 +
104 +
105 +
106 +        install(TARGETS ${PROJECT_NAME} EXPORT ${PROJECT_NAME}-targets
107 +        FILE_SET CXX_MODULES DESTINATION "${CMAKE_INSTALL_LIBDIR}/cxx/${PROJECT_NAME}"
108 +        FILE_SET HEADERS DESTINATION "include")
```

```
import std;
import ctre;

std::optional<std::string_view> extract_number(std::string_view s) noexcept {
    if (auto m = ctre::match<"[a-z]+([0-9]+)">(s)) {
        return m.get<1>().to_view();
    } else {
        return std::nullopt;
    }
}

int main()
{
    auto opt = extract_number("hello123");
    if (opt) {
        std::string s(*opt);
        std::cout << s << "\n";
    }
    return 0;
}
```

Hello modular CTRE

```
cmake_minimum_required(VERSION 3.29)
set(CMAKE_EXPERIMENTAL_CXX_IMPORT_STD "0e5b6991-d74f-4b3d-a41c-cf096e0b2508")
project(import_ctre)
find_package(ctre REQUIRED)
add_executable(ctre_hello ctre_hello.cpp)
target_link_libraries(ctre_hello PRIVATE ctre::ctre)
```

```
# Windows build with msvc
Z:\Users\hoffman\Work\modules\realworld\ctre_hello\vsb>.\ctre_hello.exe
123
# mac build with clang
hoffman@caprica b % ./ctre_hello
123
```

Clang and CMake have an issue with "import std"

CMake Error in CMakeLists.txt:

The "CXX_MODULE_STD" property on the target "ctre" requires that the
" __CMAKE::CXX23" target exist, but it was not provided by the toolchain.

Reason:

`libc++.modules.json` resource does not exist

```
diff --git a/Modules/Compiler/Clang-CXX-CXXImportStd.cmake b/Modules/Compiler/Clang-CXX-CXXImportStd.cmake
index f58f17ea65..e48c4a85fc 100644
--- a/Modules/Compiler/Clang-CXX-CXXImportStd.cmake
+++ b/Modules/Compiler/Clang-CXX-CXXImportStd.cmake
@@ -10,7 +10,7 @@ function (_cmake_cxx_import_std std variable)
    COMMAND
        "${CMAKE_CXX_COMPILER}"
        ${CMAKE_CXX_COMPILER_ID_ARC1}
-       -print-file-name=libc++.modules.json
+       -print-file-name=../../libc++.modules.json
```

HACK!!

r go to...

[CMake](#) / [CMake](#) / [Issues](#) / #25965

C++ modules: 'import std;' with libc++ does not work

[Open](#) [Issue created 4 months ago by PinieP](#)

I'm using CMake 3.29.2040507 and a clang/libc++ 19.0 build

4.7k

```
# CMakeLists.txt
cmake_minimum_required(VERSION 3.29.20240507-gf2b76d7)
set(CMAKE_CXX_STANDARD 23)
set(CMAKE_CXX_STANDARD_REQUIRED YES)

set(CMAKE_EXPERIMENTAL_CXX_IMPORT_STD "0e5b6991-d74f-4b3d-a41c-cf096e0b2508")
set(CMAKE_CXX_MODULE_STD ON)

project(example LANGUAGES CXX)
add_executable(main main.cpp)
```

```
# main.cpp
```

FIXED in clang

Note: Same error someone else's fault

CMake Error in CMakeLists.txt:

The "CXX_MODULE_STD" property on the target "ctre" requires that the
"__CMAKE::CXX23" target exist, but it was not provided by the toolchain.

Reason:

`libc++.modules.json` resource does not exist

```
cmake_minimum_required(VERSION 3.29)
#set(CMAKE_EXPERIMENTAL_CXX_IMPORT_STD "0e5b6991-d74f-4b3d-a41c-cf096e0b2508")
project(import_ctre)
find_package(ctre REQUIRED)
add_executable(ctre_hello ctre_hello.cpp)
target_link_libraries(ctre_hello PRIVATE ctre::ctre)
```

```
cmake_minimum_required(VERSION 3.29)
project(import_ctre)
set(CMAKE_EXPERIMENTAL_CXX_IMPORT_STD "0e5b6991-d74f-4b3d-a41c-cf096e0b2508")
find_package(ctre REQUIRED)
add_executable(ctre_hello ctre_hello.cpp)
target_link_libraries(ctre_hello PRIVATE ctre::ctre)
```

Better error now

CMake Error in CMakeLists.txt:

The "CXX_MODULE_STD" property on the target "ctre_ctre@synth_209e30393524" requires that the "__CMAKE::CXX23" target exist, but it was not provided by the toolchain. Reason:

Experimental `import std` support not enabled when detecting toolchain; it must be set before 'CXX' is enabled (usually a `project()` call)

```
cmake_minimum_required(VERSION 3.29)
#set(CMAKE_EXPERIMENTAL_CXX_IMPORT_STD "0e5b6991-d74f-4b3d-a41c-cf096e0b2508")
project(import_ctre)
find_package(ctre REQUIRED)
add_executable(ctre_hello ctre_hello.cpp)
target_link_libraries(ctre_hello PRIVATE ctre::ctre)
```

```
cmake_minimum_required(VERSION 3.29)
project(import_ctre)
set(CMAKE_EXPERIMENTAL_CXX_IMPORT_STD "0e5b6991-d74f-4b3d-a41c-cf096e0b2508")
find_package(ctre REQUIRED)
add_executable(ctre_hello ctre_hello.cpp)
target_link_libraries(ctre_hello PRIVATE ctre::ctre)
```

Wrong Hash Same Error

CMake Error in CMakeLists.txt:

The "CXX_MODULE_STD" property on the target "ctre_ctre@synth_209e30393524" requires that the "__CMAKE::CXX23" target exist, but it was not provided by the toolchain. Reason:

Experimental `import std` support not enabled when detecting toolchain; it must be set before 'CXX' is enabled (usually a `project()` call)

```
cmake_minimum_required(VERSION 3.20)
set(CMAKE_EXPERIMENTAL_CXX_IMPORT_STD "junk")
project(import_ctre)
find_package(ctre REQUIRED)
add_executable(ctre_hello ctre_hello.cpp)
target_link_libraries(ctre_hello PRIVATE ctre::ctre)
```

Always remove the build tree after error

This means you got it right

-- Detecting CXX compile features

CMake Warning (dev) at

/opt/homebrew/share/cmake/Modules/Compiler/CMakeCommonCompilerMacros.cmake:248 (cmake_language):

CMake's support for `import std;` in C++23 and newer is experimental. It is meant only for experimentation and feedback to CMake developers.

Call Stack (most recent call first):

/opt/homebrew/share/cmake/Modules/CMakeDetermineCompilerSupport.cmake:113

(cmake_create_cxx_import_std)

/opt/homebrew/share/cmake/Modules/CMakeTestCXXCompiler.cmake:83

(CMAKE_DETERMINE_COMPILER_SUPPORT)

CMakeLists.txt:15 (project)

This warning is for project developers. Use -Wno-dev to suppress it.

Where did the expression "winner winner chicken dinner" come from?

In the late 1920s (the Depression Era in the United States), a chicken dinner from a casino cafeteria cost just under \$2—and \$2 was the standard bet for many gamblers. So,

anyone who won a bet was effectively able to buy dinner that day, hence the phrase “Winner winner chicken dinner.”

Jun 3, 2024



<input checked="" type="checkbox"/>	fmt	import fmt;	60	+11	2022/10/11	
<input checked="" type="checkbox"/>	vladimirshaleev-ipaddress	import ipaddress;	1		2024/6/2	
<input checked="" type="checkbox"/>	z4kn4fein-semver	import semver;	1	C++17	2024/8/21	

5 more projects

Are We Modules Yet? ☺

Estimated finish by: Mon Dec 08 2555



⬆ There is a progress bar (promise)

22 / 2431 (0.90%) Confirmed Complete

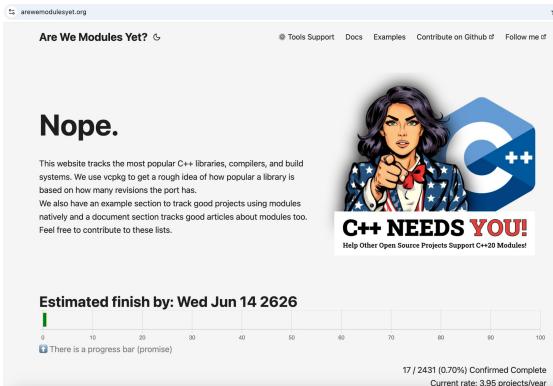
Current rate: 4.47 projects/year

Other things I learned

- CMake code is often formatted with tabs and goofy spacing, just why?
- Be very careful with versions of things
 - a wrong version of cmake, ninja, compiler can result in a vomiting of not so useful errors at build time!

Don't Despair Code!

- `import std` will pave the way!





about ▾

solutions ▾

news ▾

careers

contact

EU

Enter Keyword


🏠 news > blog > post

import std in CMake 3.30

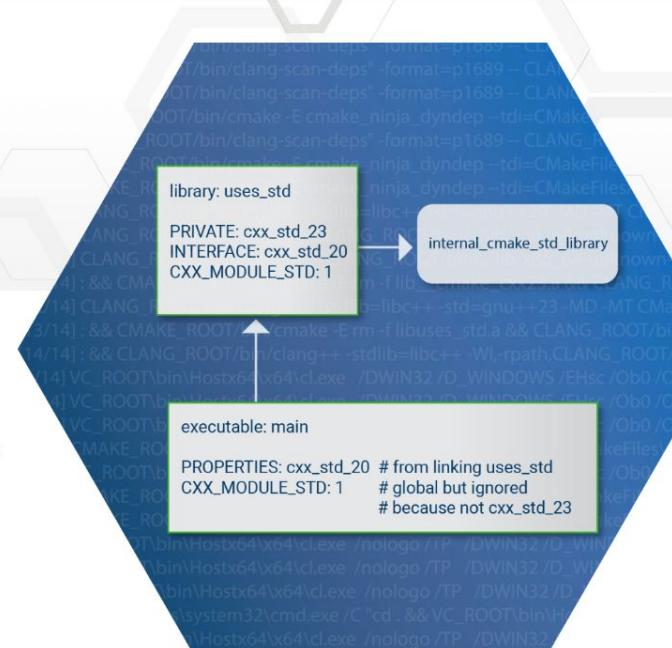
April 18, 2024

Bill Hoffman and Ben Boeckel

A New Experiment!

Since C++ named modules left the experimental state in CMake (see [import CMake: the Experiment is Over!](#)), CMake development has continued to add more support for modules. The most recent development is the experimental support of import std;. C++23 introduced support for APIs to be available via the named module std. This means that `import std;` is a valid way to get access to all non-macro C++ APIs instead of including headers one-by-one. This is great news, however to use modules the build system needs to be involved to create the BMI files before the import can happen.

The CMake support for named modules included the ability to export targets for later import. The solution leverages that support with toolchains declaring a target to represent the standard library for each standard. This target is used inside of CMake's own logic and added to projects automatically; projects do not need to mention the targets in project code at all. When CMake encounters a target which has at least C++23 enabled that also supports C++ modules¹, it will query the `CXX_MODULE_STD` property on the target (which may contain generator expressions²). This property is initialized by the `CMAKE_CXX_MODULE_STD` variable, if defined. If it is unset, the current behavior is to not use the target. The internal target is not



import std;

- A named module that is built around existing header files.
- Let's take a look under the hood.



CMake import std support (4.0.1)

C++ import std support

In order to activate support for `import std` in C++23 and newer targets, set

- variable `CMAKE_EXPERIMENTAL_CXX_IMPORT_STD` to
- value `a9e1cf81-9932-4810-974b-6eccaf14e457`.

This UUID may change in future versions of CMake. Be sure to use the value documented here by the source tree of the version of CMake with which you are experimenting. It must be set before the `CXX` toolchain is discovered by CMake, usually as part of a `:command:`project`` call.

When activated, this experimental feature provides the following:

- The `:prop_tgt:`CXX_MODULE_STD`` target property and its initializing variable `:variable:`CMAKE_CXX_MODULE_STD``.
- Targets with the property set to a true value and at least `cxx_std_23` may use `import std;` in any scanned C++ source file.

"import std" model

- Named module for an existing library little change to existing code
- Just need to create a module file like std.cppm, but what goes in that file?

Clang - std.cppm

```
// WARNING, this entire header is generated by
// utils/generate_libcxx_cppm_in.py
// DO NOT MODIFY!

module;

#include <__config>

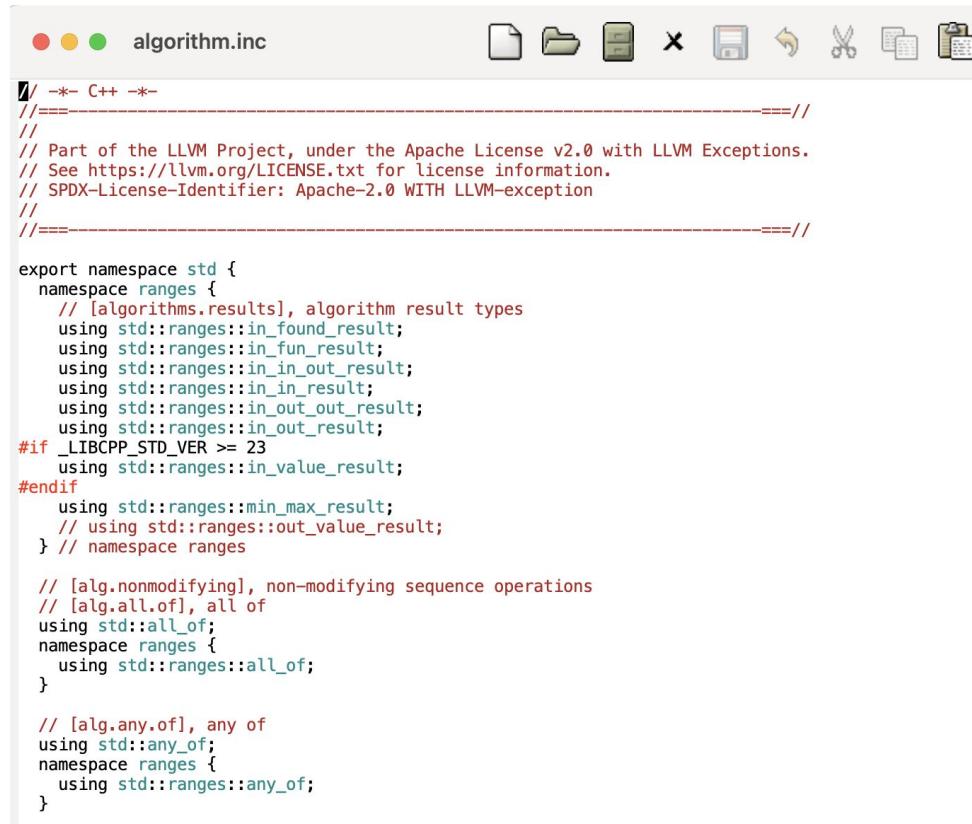
// The headers of Table 24: C++ library headers_[tab:headers.cpp]
// and the headers of Table 25: C++ headers for C library facilities_[tab:headers.cpp.c]
#include <algorithm>
#include <any>
#include <array>
#if !defined(_LIBCPP_HAS_NO_ATOMIC_HEADER)
# include <atomic>
#endif
#include <barrier>
#include <bit>
#include <bitset>
#include <cassert>
#include <cctype>
#include <cerrno>
#include <cfenv>
#include <cfloat>
#include <charconv>
#include <chrono>
#include <cinttypes>
#include <climits>
#if !defined(_LIBCPP_HAS_NO_LOCALIZATION)
# include <locale>
#endif
```

Clang - std.cppm

```
export module std;

#include "std/algorithm.inc"
#include "std/any.inc"
#include "std/array.inc"
#include "std/atomic.inc"
#include "std/barrier.inc"
#include "std/bit.inc"
#include "std/bitset.inc"
#include "std/cassert.inc"
#include "std/cctype.inc"
#include "std/cerrno.inc"
#include "std/cfenv.inc"
#include "std/cfloat.inc"
#include "std/charconv.inc"
#include "std/chrono.inc"
#include "std/cinttypes.inc"
#include "std/climits.inc"
#include "std/clocale.inc"
#include "std/cmath.inc"
#include "std/codecvt.inc"
#include "std/compare.inc"
#include "std/complex.inc"
#include "std/concepts.inc"
#include "std/condition_variable.inc"
#include "std/coroutine.inc"
#include "std/csetjmp.inc"
#include "std/csignal.inc"
#include "std/cstدارگinc"
#include "std/cstddef.inc"
#include "std/cstdint.inc"
#include "std/cstdio.inc"
```

Clang - std.cppm



The image shows a screenshot of a code editor window. The title bar says "algorithm.inc". The toolbar contains icons for file operations like new, open, save, and cut/paste. The code itself is a header file for the LLVM project, specifically for the ranges and any_of algorithms. It includes Apache 2.0 license information and defines namespaces for ranges and any_of.

```
/* -*- C++ -*-  
//==-----//  
//  
// Part of the LLVM Project, under the Apache License v2.0 with LLVM Exceptions.  
// See https://llvm.org/LICENSE.txt for license information.  
// SPDX-License-Identifier: Apache-2.0 WITH LLVM-exception  
//  
//==-----//  
  
export namespace std {  
    namespace ranges {  
        // [algorithms.results], algorithm result types  
        using std::ranges::in_found_result;  
        using std::ranges::in_fun_result;  
        using std::ranges::in_in_out_result;  
        using std::ranges::in_in_result;  
        using std::ranges::in_out_out_result;  
        using std::ranges::in_out_result;  
#if _LIBCPP_STD_VER >= 23  
        using std::ranges::in_value_result;  
#endif  
        using std::ranges::min_max_result;  
        // using std::ranges::out_value_result;  
    } // namespace ranges  
  
    // [alg.nonmodifying], non-modifying sequence operations  
    // [alg.all.of], all of  
    using std::all_of;  
    namespace ranges {  
        using std::ranges::all_of;  
    }  
  
    // [alg.any.of], any of  
    using std::any_of;  
    namespace ranges {  
        using std::ranges::any_of;  
    }
```

MSVC - std.ixx

```
// Copyright (c) Microsoft Corporation.  
// SPDX-License-Identifier: Apache-2.0 WITH LLVM-exception  
  
// In a module-file, the optional `module;` must appear first; see [cpp.pre].  
module;  
  
// This named module expects to be built with classic headers, not header units.  
#define _BUILD_STD_MODULE  
  
// The subset of "C headers" [tab:c.headers] corresponding to  
// the "C++ headers for C library facilities" [tab:headers.cpp.c]  
#include <assert.h>  
#include <ctype.h>  
#include <errno.h>  
#include <fenv.h>  
#include <float.h>  
#include <inttypes.h>  
#include <limits.h>  
#include <locale.h>  
#include <math.h>  
#include <setjmp.h>  
#include <signal.h>  
#include <stdarg.h>  
#include <stddef.h>  
#include <stdint.h>  
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include <time.h>  
#include <uchar.h>  
#include <wchar.h>  
#include <wctype.h>
```

MSVC - std.ixx

```
export module std;

#pragma warning(push)
#pragma warning(disable : 5244) // '#include <meow>' in the purview of module 'std'
// "C++ library headers"
#include <algorithm>
#if _HAS_STATIC_RTTI
#include <any>
#endif // _HAS_STATIC_RTTI
#include <array>
#include <atomic>
#include <barrier>
#include <bit>
#include <bitset>
#include <charconv>
#include <chrono>
#include <codecvt>
#include <compare>
#include <complex>
#include <concepts>
#include <condition_variadic>
#include <coroutine>
#include <deque>
#include <exception>
#include <execution>
```



??

MSVC - <vector>

```
EXPORT_STD template <class _Ty, class _Alloc = allocator<_Ty>>
class vector { // varying size array of values
private:
    template <class>
    friend class _Vb_val;
    friend _Tidy_guard<vector>;
    using _Alty      = _Rebind_alloc_t<_Alloc, _Ty>;
    using _Alty_traits = allocator_traits<_Alty>;
public:
    static_assert(!_ENFORCE_MATCHING_ALLOCATORS || is_same_v<_Ty, typename _Alloc::value_type>,
                  _MISMATCHED_ALLOCATOR_MESSAGE("vector<T, Allocator>", "T"));
    static_assert(is_object_v<_Ty>, "The C++ Standard forbids containers of non-object types "
                 "because of [container.requirements].");
    using value_type      = _Ty;
    using allocator_type  = _Alloc;
    using pointer          = typename _Alty_traits::pointer;
    using const_pointer    = typename _Alty_traits::const_pointer;
    using reference        = _Ty&;
    using const_reference  = const _Ty&;
    using size_type        = typename _Alty_traits::size_type;
    using difference_type  = typename _Alty_traits::difference_type;
```

MSVC- <vector> expanded

```
export template <class _Ty, class _Alloc = allocator<_Ty>>
class vector {
private:
    template <class>
    friend class _Vb_val;
    friend _Tidy_guard<vector>;
    using _Alty      = _Rebind_alloc_t<_Alloc, _Ty>;
    using _Alty_traits = allocator_traits<_Alty>;
public:
    static_assert(!1 || is_same_v<_Ty, typename _Alloc::value_type>,
        "vector<T, Allocator>" " requires that Allocator's value_type match \" \"T\" \" (See N4950 [container.alloc.reqmts]/5)" " Either fix the allocator value_type or define _ENFORCE_MATCHING_ALLOCATORS=0" " to suppress this error.");
    static_assert(is_object_v<_Ty>, "The C++ Standard forbids containers of non-object types "
        "because of [container.requirements].");
    using value_type      = _Ty;
    using allocator_type  = _Alloc;
    using pointer          = typename _Alty_traits::pointer;
    using const_pointer    = typename _Alty_traits::const_pointer;
    using reference         = _Ty&;
    using const_reference  = const _Ty&;
    using size_type        = typename _Alty_traits::size_type;
    using difference_type  = typename _Alty_traits::difference_type;
```

CMake import std

```
# CMake 3.30 is required for C++23 `import std` support; we use 3.29.20240416
# here so that in-development versions satisfy it.
cmake_minimum_required(VERSION 3.29.20240416 FATAL_ERROR)

# Set experimental flag to enable `import std` support from CMake.
# This must be enabled before C++ language support.
set(CMAKE_EXPERIMENTAL_CXX_IMPORT_STD
    # This specific value changes as experimental support evolves. See
    # `Help/dev/experimental.rst` in the CMake source corresponding to
    # your CMake build for the exact value to use.
    "0e5b6991-d74f-4b3d-a41c-cf096e0b2508")

# C++ needs to be enabled.
project(import_std LANGUAGES CXX)
```

```
# Tell CMake that we explicitly want `import std`. This will initialize the
# property on all targets declared after this to 1
set(CMAKE_CXX_MODULE_STD 1)

# Make a library.
add_library(uses_std STATIC)
# Add sources.
target_sources(uses_std
    PRIVATE
        uses_std.cxx)
# Tell CMake we're using C++23 but only C++20 is needed to consume it.
target_compile_features(uses_std
    PRIVATE    cxx_std_23
    INTERFACE  cxx_std_20)

# Make an executable.
add_executable(main)
# Note that this source is *not* allowed to `import std` as it ends up
# with only C++20 support due to the `uses_std` INTERFACE requirements.
target_sources(main
    PRIVATE
        main.cxx)
target_link_libraries(main PRIVATE uses_std)
```

With the following `uses_std.cxx` source:

```
import std;

void hello_world(std::string const& name)
{
    std::cout << "Hello World! My name is " << name << std::endl;
}
```

And the executable's source:

```
#include <string>

void hello_world(std::string const& name);

int main(int argc, char* argv[])
{
    hello_world(argv[0] ? argv[0] : "Voldemort?");
    return 0;
}
```

CMake import std

```
[1/14] "CLANG_ROOT/bin/clang-scan-deps" -format=p1689 -- CLANG_ROOT/bin/clang++ -stdlib=libc++ -std=gnu++20 -MD -MT CMakeFiles/main.dir/main.cxx.o -MF CMakeFiles/main.dir/main.cxx.o.d
[2/14] "CLANG_ROOT/bin/clang-scan-deps" -format=p1689 -- CLANG_ROOT/bin/clang++ -ICLANG_ROOT/bin/../lib/libc++.so.1 -std=gnu++20 -MD -MT CMakeFiles/main.dir/main.cxx.o.d
[3/14] "CLANG_ROOT/bin/clang-scan-deps" -format=p1689 -- CLANG_ROOT/bin/clang++ -stdlib=libc++ -std=gnu++20 -MD -MT CMakeFiles/main.dir/main.cxx.o.d
[4/14] CMAKE_ROOT/bin/cmake -E cmake_ninja_dyndep --tdi=CMakeFiles/main.dir/CXXDependInfo.json --lang=0
[5/14] "CLANG_ROOT/bin/clang-scan-deps" -format=p1689 -- CLANG_ROOT/bin/clang++ -ICLANG_ROOT/bin/../lib/libc++.so.1 -std=gnu++20 -MD -MT CMakeFiles/main.dir/main.cxx.o.d
[6/14] CMAKE_ROOT/bin/cmake -E cmake_ninja_dyndep --tdi=CMakeFiles/__cmake_cxx23.dir/CXXDependInfo.json --lang=0
[7/14] CMAKE_ROOT/bin/cmake -E cmake_ninja_dyndep --tdi=CMakeFiles/uses_std.dir/CXXDependInfo.json --lang=0
[8/14] CLANG_ROOT/bin/clang++ -stdlib=libc++ -std=gnu++20 -MD -MT CMakeFiles/main.dir/main.cxx.o -MF CMakeFiles/main.dir/main.cxx.o.d
[9/14] CLANG_ROOT/bin/clang++ -ICLANG_ROOT/bin/../lib/x86_64-unknown-linux-gnu/../../share/libc++/v1/libc++.so.1 -std=gnu++20 -MD -MT CMakeFiles/main.dir/main.cxx.o.d
[10/14] CLANG_ROOT/bin/clang++ -ICLANG_ROOT/bin/../lib/x86_64-unknown-linux-gnu/../../share/libc++/v1/libc++.so.1 -std=gnu++20 -MD -MT CMakeFiles/main.dir/main.cxx.o.d
[11/14] : && CMAKE_ROOT/bin/cmake -E rm - lib__cmake_cxx23.a && CLANG_ROOT/bin/llvm-ar qc lib__cmake_cxx23.a
[12/14] CLANG_ROOT/bin/clang++ -stdlib=libc++ -std=gnu++20 -MD -MT CMakeFiles/uses_std.dir/uses_std.o -MF CMakeFiles/uses_std.dir/uses_std.o.d
[13/14] : && CMAKE_ROOT/bin/cmake -E rm -f libuses_std.a && CLANG_ROOT/bin/llvm-ar qc libuses_std.a CMakeFiles/uses_std.dir/uses_std.o.d
[14/14] : && CLANG_ROOT/bin/clang++ -stdlib=libc++ -Wl,-rpath,CLANG_ROOT/lib/x86_64-unknown-linux-gnu/libc++.so.1 -std=gnu++20 -MD -MT CMakeFiles/uses_std.dir/uses_std.o.d
```

Some Module Options

- Modules and headers
 - manual "using" - glm
 - generated "using" - clang
 - `#define MODULE_EXPORT` - MSVC, boost, ctre
- Pure Modules

using approach glm

```
module;

// #define GLM_GTC_INLINE_NAMESPACE to inline glm::gtc into glm
// #define GLM_EXT_INLINE_NAMESPACE to inline glm::ext into glm
// #define GLM_GTX_INLINE_NAMESPACE to inline glm::gtx into glm

#include <glm/glm.hpp>
#include <glm/ext.hpp>

export module glm;

export namespace glm {
    // Base types
    using glm::qualifier;
    using glm::precision;
```

export approach boost pfr

```
module;
#include <version>
...
#ifndef BOOST_PFR_USE_STD_MODULE
import std;
#else
#include <array>
...
#define BOOST_PFR_INTERFACE_UNIT
export module boost.pfr;

#include <boost/pfr.hpp>    // can't include other headers
```

export approach boost pfr cont

```
#ifdef BOOST_PFR_INTERFACE_UNIT
#  define BOOST_PFR_BEGIN_MODULE_EXPORT export {
#  define BOOST_PFR_END_MODULE_EXPORT }
#else
#  define BOOST_PFR_BEGIN_MODULE_EXPORT
#  define BOOST_PFR_END_MODULE_EXPORT
#endif
namespace boost { namespace pfr {
BOOST_PFR_BEGIN_MODULE_EXPORT
template <std::size_t I, class T>
constexpr decltype(auto) get(const T& val) noexcept {
    return detail::sequence_tuple::get<I>( detail::tie_as_tuple(val) );
}
...
BOOST_PFR_END_MODULE_EXPORT
```

beman/exemplar

```
// file identity.cppm

module; // start module

#define BUILD_MODULE // let identity.hpp know we want to use it as a module
export module exemplar; // start the module exemplar
#include "identity.hpp" // include the header
```

beman/exemplar

```
#ifdef BUILD_MODULE
import std;
#define EXPORT export
#else
#define EXPORT
#include <utility> // std::forward
#endif
namespace beman::exemplar {
EXPORT struct __is_transparent; // not defined
// A function object that returns its argument unchanged.
EXPORT struct identity {
    // Returns `t`.
    template <class T>
    constexpr T& operator()(T& t) const noexcept {
        return std::forward<T>(t);
    }
    using is_transparent = __is_transparent;
};
} // namespace beman::exemplar
```

beman/exemplar

```
cmake_minimum_required(VERSION 3.40 )
set(CMAKE_EXPERIMENTAL_CXX_IMPORT_STD a9e1cf81-9932-4810-974b-6eccaf14e457)
project(
    beman.exemplar # CMake Project Name, which is also the name of the top-level
    # targets (e.g., library, executable, etc.).
    DESCRIPTION "A Beman library exemplar"
    LANGUAGES CXX
    VERSION 2.1.1
)
set(CMAKE_CXX_MODULE_STD 1)
```

beman/exemplar

```
target_sources(
    beman.exemplar
    PUBLIC
        FILE_SET cxx_modules TYPE CXX_MODULES FILES
        BASE_DIRS ${CMAKE_CURRENT_SOURCE_DIR}/../../../../../include
        FILES
${CMAKE_CURRENT_SOURCE_DIR}/../../../../include/beman/exemplar/identity.cppm
)

install(
    TARGETS beman.exemplar COMPONENT beman.exemplar
    EXPORT beman.exemplar
    DESTINATION
    ${CMAKE_INSTALL_LIBDIR}${<$<CONFIG:Debug>:/debug}
    RUNTIME DESTINATION ${CMAKE_INSTALL_BINDIR}${<$<CONFIG:Debug>:/debug}
    FILE_SET HEADERS DESTINATION ${CMAKE_INSTALL_INCLUDEDIR}
    FILE_SET cxx_modules DESTINATION ${CMAKE_INSTALL_INCLUDEDIR}
)
```

<https://github.com/ChuanqiXu9/clang-modules-converter>

ChuanqiXu9 / clang-modules-converter

Issues Pull requests Actions Projects Security Insights

 clang-modules-converter Public

 Watch 1  Fork 1  Star 32

 main  1 Branch  0 Tags

Go to file

Add file

 Code

About

A helper to convert a header-based C++ project to module-based one.

 Readme

 View license

 Activity

 32 stars

 1 watching

 1 fork

Report repository

Releases

No releases published

Packages

No packages published

Contributors 2

Intro

A clang-based helper tool to rewrite a header-based C++ project into a module-based one.



clang-modules-converter

☞ Intro

A clang-based helper tool to rewrite a header-based C++ project into a module-based one.

The tool provides three styles to rewrite the project:

- Rewrite a header-based project to make it provide the module interfaces, but still keep the project as header-based one. Which is known as header wrappers style.
- Rewrite a header-based project into a module-based one, but still provide the original headers. This style is helpful when we decide to develop new features in modules and want to keep the old interfaces.
- Rewrite a header-based project into a module-based one completely without keeping the headers. This may be wanted if no users of the project are expected to not use headers anymore.

Note this tool is **not** expected to run continuously like clang-format. This tool is only expected to save some trivial works to refactor the projects.

clang-modules-converter

How this works

The core ability of the tool is to find the preamble section for each file in the project. The preamble section is the section to introduce dependency of the file. For example,

```
// my_lib.h
// Start of preamble
#include <iostream>
#include <string>
#ifndef USE_MAP
#include <map>
#endif
#include "local_headers.h"
// End of preamble

namespace my_lib {
...
}
```



Then we can manipulate the preamble section:

```
// my_lib.h
#ifndef MY_LIB_USE_MODULES
#include <iostream>
#include <string>
#ifndef USE_BOOST
#include <boost/...>
#endif
#include "local_headers.h"
#endif // MY_LIB_USE_MODULES

namespace my_lib {
...
}
```



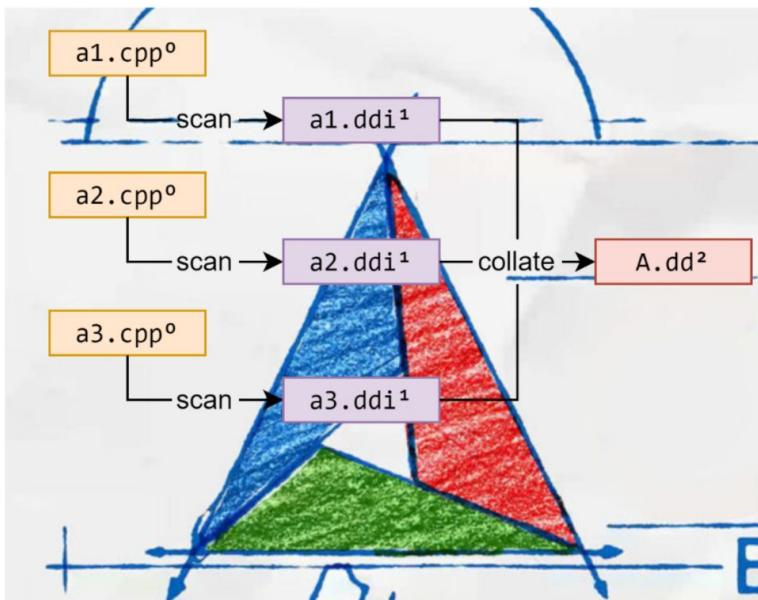
So the header will only contain its own body when `MY_LIB_USE_MODULES` is defined. Then we can export its body in the module interface:

Resources

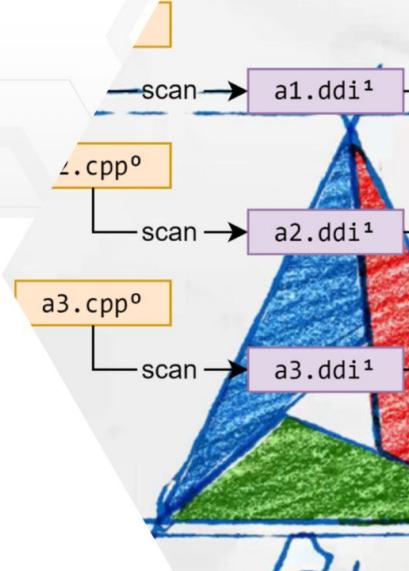
import CMake; the Experiment is Over!

October 18, 2023

Bill Hoffman, Brad King and Ben Boeckel



At last the experiment is over, and CMake 3.28 has official support for C++ 20 named modules enabled without having to set



**Don't forget to
SUBSCRIBE.**

Our news delivered right to your
Sign up now to stay up to date with
latest trends and updates from
Kitware.



Professional CMake: A Practical Guide

Learn to use CMake effectively with practical advice from a CMake co-maintainer. You can also have the author work directly with your team!

GET THE BOOK

AUTHOR SERVICES



C++20 Modules, CMake, And Shared Libraries

April 4, 2024 by Craig Scott

CMake 3.28 was the first version to officially support C++20 modules. Tutorials and examples understandably tend to focus on the fairly simple scenario of building a basic executable, perhaps also adding in a static library. The [import CMake; the Experiment is Over!](#) blog article from Kitware is perhaps one of the best known, and it covers exactly these things. And while these are important first steps, stopping there sells the reader short. The real fun starts when building, installing, and consuming shared libraries. This quickly exposes current limitations of toolchains and the build system, and it also highlights common misconceptions about what modules provide.

links for resources

- <https://gitlab.kitware.com/cmake/cmake/-/tree/master/Tests/RunCMake/CXXModules/examples>
- <https://www.kitware.com/import-std-in-cmake-3-30/>
- <https://www.kitware.com/import-cmake-the-experiment-is-over/>
- <https://github.com/ChuanqiXu9/clang-modules-converter>
- <https://crascit.com/2024/04/04/cxx-modules-cmake-shared-libraries/>

All I am saying is give modules a chance

- Try "import std"
- Create a named module for your project with install rules
- Be prepared to build compilers, cmake, and other tools and install previews.
- Be prepared for other errors from the bleeding edge compilers you will be using

Special Thanks

Bret Brown

Daniel Ruoso

Ben Boeckel

Chuanqi Xu

Bloomberg

Engineering

Thank You

- bill.hoffman@kitware.com
- Read “how to write a CMake buildsystem”
 - <https://cmake.org/cmake/help/latest/manual/cmake-buildsystem.7.html> Explore the CMake documentation
- Explore the CMake documentation

The screenshot shows a web browser displaying the CMake 3.8.0 documentation. The URL in the address bar is <https://cmake.org/cmake/help/latest/>. The page title is "Documentation". The left sidebar contains links for "Table Of Contents", "Command-Line Tools", "Interactive Dialogs", "Reference Manuals", "Release Notes", and "Index and Search". Below these are "Next topic" (cmake(1)), "This Page", "Show Source", and "Quick search" fields with a "Go" button.

Command-Line Tools

- cmake(1)
- ctest(1)
- cpack(1)

Interactive Dialogs

- cmake-gui(1)
- ccmake(1)

Reference Manuals

- cmake-buildsystem(7)
- cmake-commands(7)
- cmake-compile-features(7)
- cmake-developer(7)
- cmake-generator-expressions(7)
- cmake-generators(7)
- cmake-language(7)
- cmake-server(7)
- cmake-modules(7)
- cmake-packages(7)
- cmake-policies(7)