Presentation

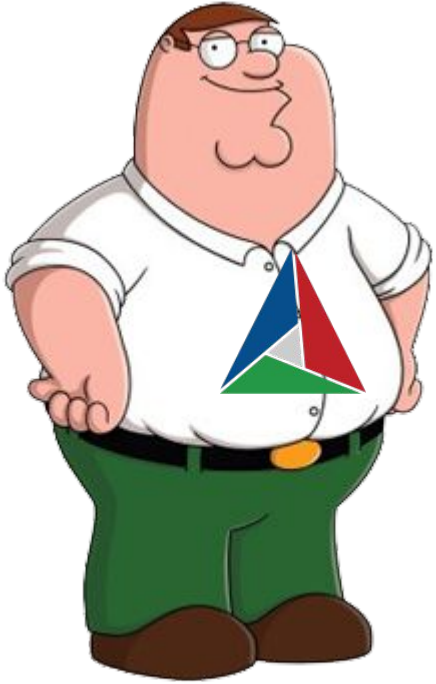
# Import CMake: 2023 State of C++20 Modules in CMake

*Bill Hoffman, Chief Technical Officer and Co-founder of Kitware*

Thursday, May 11 from 9-10:30 AM MT

C++ now

CMake / Sandal / Kitware GUY




kitware

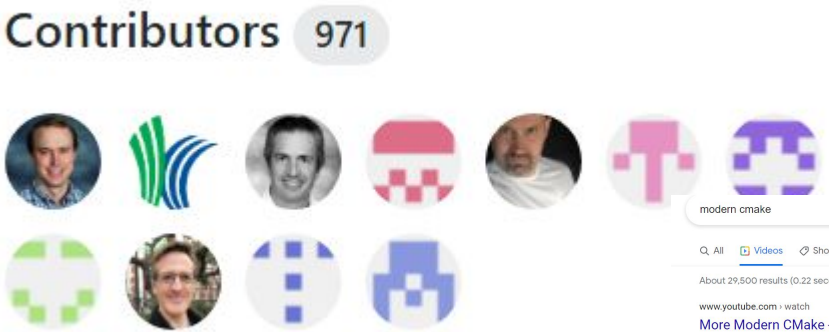
Bill Hoffman

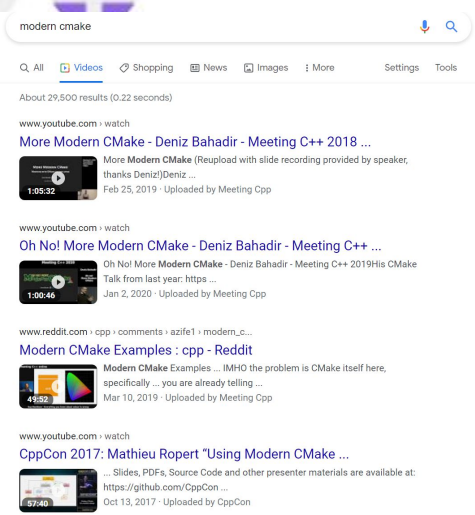
# About Me

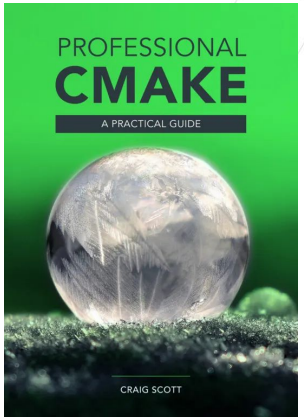
- **1990-1999 GE research in a Computer Vision group.**
  - Moved them from symbolics lisp machines to C++ on Solaris and later linux/Windows
  - Was the build and software library guy (gmake/autotools)
- **1998-Present Kitware**
  - Lots of development in CMake/ITK/VTK
  - Mostly management now, finding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

kitware

# CMake is a Community Effort



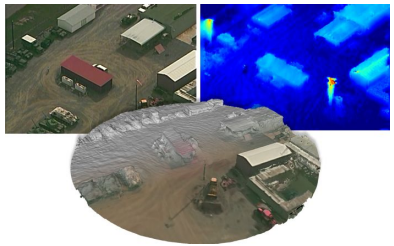



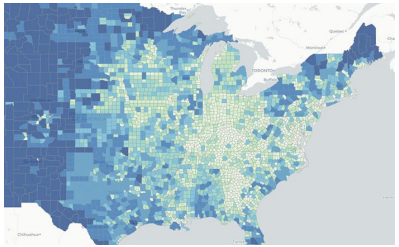



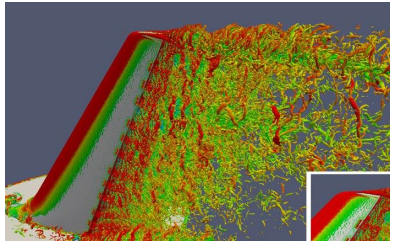

Brad King and Ben Boeckel are the CMake guys!

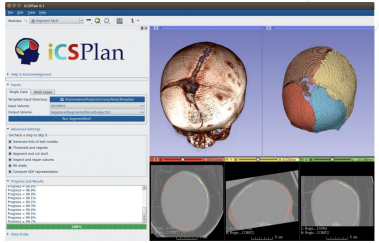
kitware

# Kitware Overview / Built on open source










Computer Vision

Data and Analytics

Scientific Computing

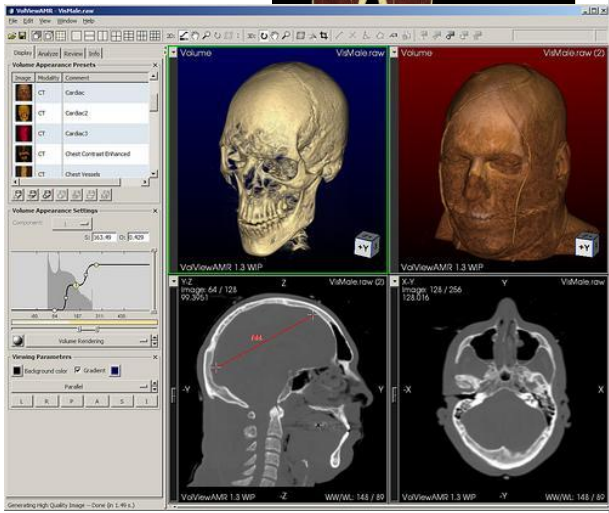
Medical Computing

Software Solutions

kitware

# Where did CMake come from?

- Kitware was the lead engineering team for the Insight Segmentation and Registration Toolkit (ITK) http://www.itk.org
- 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
- Kitware one of three commercial companies on the project.
  - tasked with making it build on Unix/Windows/Mac
- Since then funding from many projects and outside contributions



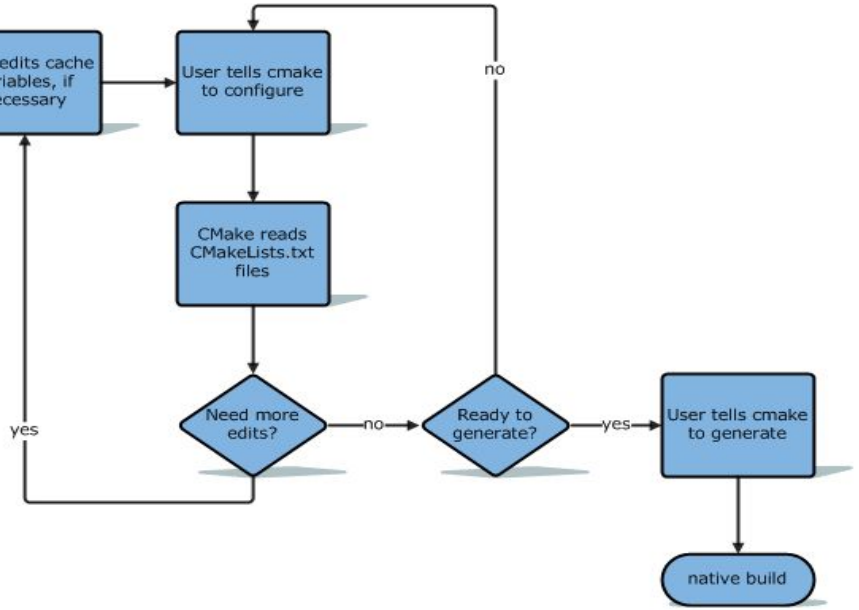
kitware

# CMake adapts to new technologies so developers don't have to

- New build IDE's and compilers
  - Visual Studio releases supported weeks after beta comes out
  - Xcode releases supported weeks after beta comes out
  - ninja (command line build tool from Google) support contributed to CMake as ninja matured
  - Apple Silicon
  - C++ 20 Modules

# Running CMake



# “Usage Requirements” aka Modern CMake

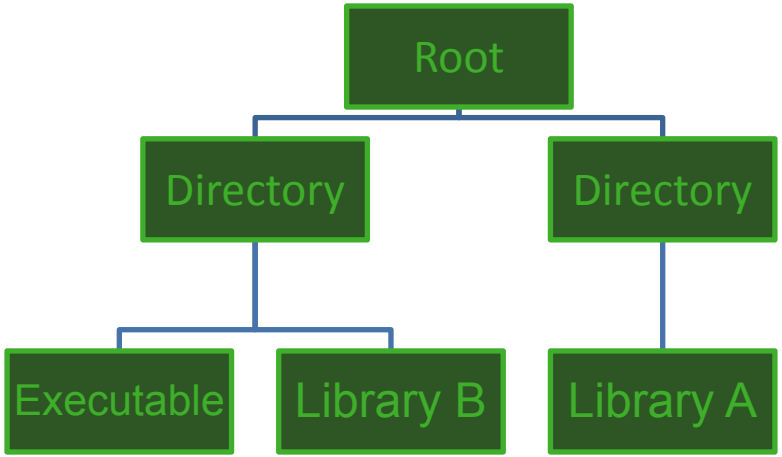
Modern style: target-centric

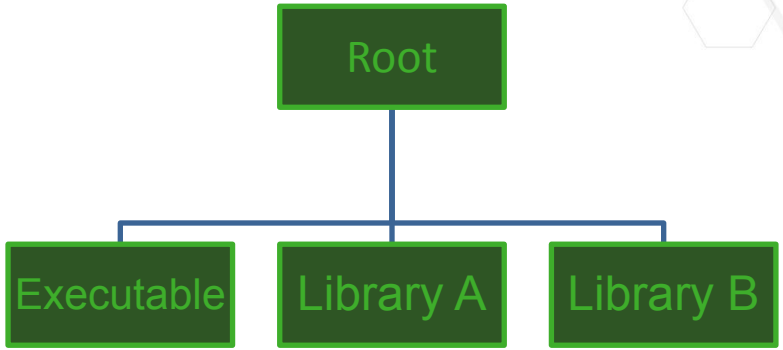
```
target_include_directories(example PUBLIC "inc")
```

example and anything that links to gets `-Iinc`

- Each target should fully describe how to properly use it
- No difference between external and internal targets

# Modern CMake





kitware

# Usage Requirements

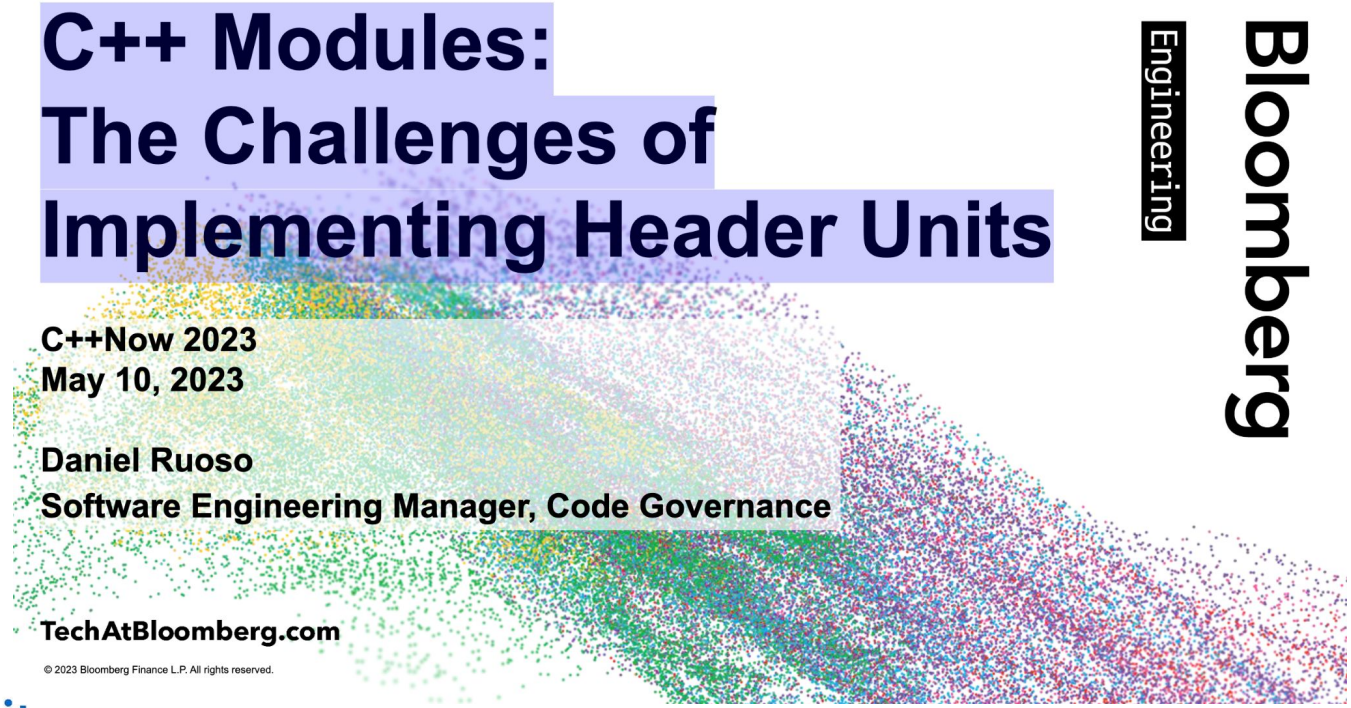
| | |
|---|---|
| `PRIVATE:` | Only the given target will use it |
| `INTERFACE:` | Only consuming targets use it |
| `PUBLIC:` | **PRIVATE + INTERFACE** |
| `$<BUILD_INTERFACE>:` | Used by consumers from this project or using the build directory |
| `$<INSTALL_INTERFACE>:` | Used by consumers after this target has been installed |

# Full cross platform install system

- Specify rules to run at install time
- Can install targets, files, or directories
- Provides default install locations

```
add_library(leaf SHARED leaf.cxx)
install(TARGETS root trunk leaf parasite)
```

# C++ Named Modules Only




kitware

Does CMake support C++20 modules

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)
```

# Built Module Interface (BMI)

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

P1838R0: Modules User-Facing Lexicon and File Extensions

ISO/IEC JTC1 SC22/WG21 - Programming Languages - C++

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.

kitware

# 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
```

kitware

# 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
```

kitware

# 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 has 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!

kitware

# 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
  - CAN NOT/WILL NOT DO THIS FOR C++
- 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
*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

kitware

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

# defining a format for compilers

## Format for describing dependencies of source files

**Ben Boeckel, Brad King**
<ben.boeckel@kitware.com, brad.king@kitware.com>
version P1689R5, 2022-06-03

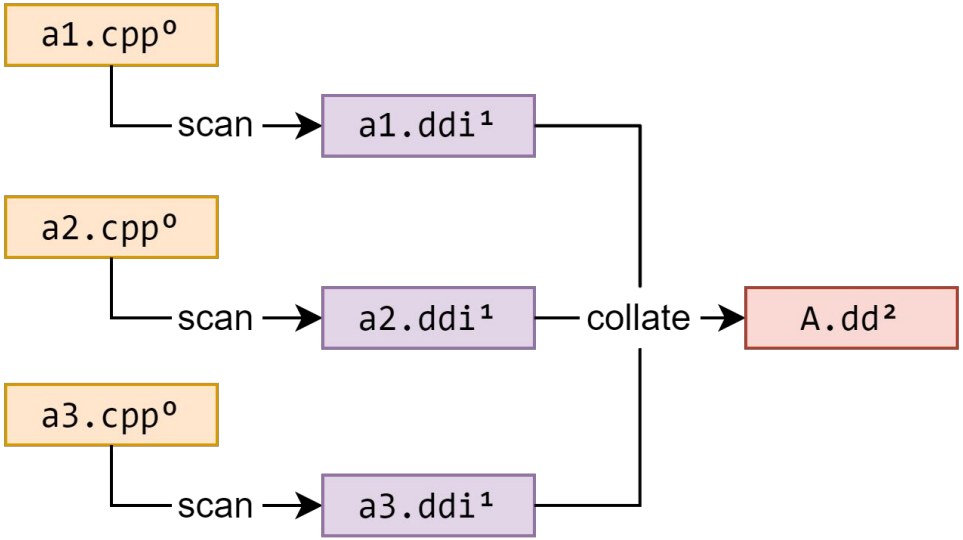
**Table of Contents**



| | |
|---|---|
| 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) |

kitware

# Per target scanning




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

collate: cmake -E cmake_ninja_dyndep

kitware

| Processes | Files produced |
| --- | --- |
| CMake | rules.ninja<br>build.nina<br>CXXDependInfo.json<br>(per target) |
| ninja | |
| c++ scan per TU | TU.o.ddi (json file from p1689r5) |
| cmake -E cmake_ninja_dyndep (per target) | CXX.dd (ninja dyndep file)<br>TU.modmap (passed to compiler paths to BMI)<br>CXXModules.json (cmake readable ver of CXX.dd) |
| CC compile per TU (in order based on dyndep) | .o .lib .exe, etc |

kitware

# Where we are today

- Support for p1689r5 in compiler releases
  - Visual studio 2022 preview
  - Clang 16
- GCC - patch for named modules
  - https://github.com/mathstuf/gcc/tree/p1689r5
- CMake 3.26 or newer has experimental support
  - set(CMAKE_EXPERIMENTAL_CXX_MODULE_CMAKE_API "2182bf5c-ef0d-489a-91da-49dbc3090d2a")

# VS 2022

## `/scanDependencies` (List module dependencies in standard form)

Article • 03/01/2022 • 3 minutes to read • 2 contributors

This compiler option generates a JSON file that lists module and header-unit dependencies according to C++ Standard proposal P1689R4 Format for describing dependencies of source files.

The `/scanDependencies` compiler option identifies which modules and header units need to be compiled before you can compile the project that uses them. For instance, it lists `import <library>;` or `import "library";` as a header unit dependency, and `import name;` as a module dependency. The intent is to provide this information in a common format consumable by build tools such as CMake.

This command-line option is similar to /sourceDependencies-directives and /sourceDependencies, but differs in the following ways:

- The output uses the P1689R4 schema, instead of the Microsoft-specific schema generated by `/sourceDependencies-directives`.
- Unlike `/sourceDependencies`, the compiler doesn't produce compiled output. Instead, the files are scanned for module directives. No compiled code, modules, or header units are produced.
- The output JSON file doesn't list imported modules and imported header units (`.ifc` files) because this option only scans the project files. There are no built modules or header units to list.
- Only directly imported modules or header units are listed. It doesn't list the dependencies of the imported modules or header units themselves.
- Textually included header files such as `#include <file>` or `#include "file"` aren't listed as dependencies unless translated to a header unit by using the /translateInclude option.
- `/scanDependencies` is meant to be used before `.ifc` files are built.

kitware

# Clang 16

```
clang-scan-deps --version
LLVM (http://llvm.org/):
  LLVM version 16.0.0

clang-scan-deps --help
USAGE: clang-scan-deps [options] The command line flags for the target of which the dependencies are to be computed.
OPTIONS:
Generic Options:
  --help                      - Display available options (--help-hidden for more)
  --help-list                 - Display list of available options (--help-list-hidden for more)
  --version                   - Display the version of this program
Tool options:
  --compilation-database=<string>     - Compilation database
  --dependency-target=<string>        - The names of dependency targets for the dependency file
  --deprecated-driver-command         - use a single driver command to build the tu (deprecated)
  --eager-load-pcm                    - Load PCM files eagerly (instead of lazily on import).
  --format=<value>                    - The output format for the dependencies
    =make                     -   Makefile compatible dep file
    =p1689                    -   Generate standard c++ modules dependency P1689 format
    =experimental-full              -   Full dependency graph suitable for explicitly building modules. This format is experimental and will
change.
```

kitware

# CMake Generators

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

# C++ Now Success Stories

kitware

# MSVC Internal Modules Fixed!

7/13/22

Howdy!

Update on this topic: VS Dev17.3 preview 3 – released yesterday afternoon – as the fixes as we discussed:

- CL.exe now emits dependency information based on the revised dependency format
  - /std:c++20 /scanDependencies has the ISO C++ interpretation of module partition declaration (by default)
  - /std:c++20 /scanDependencies:partitionImplementation has the MSVC extended interpretation
- Both cases emit the new field “is-interface” with the appropriate value

Please, let me know if you run into other issues or you have any questions related to this.

Thanks,

-- Gaby

Working on integrating this with CMake so that only if you use the MS extension do you need to flag your source files.

Thanks to Daniel Ruoso, Bret Brown, Ben Boeckel and Brad King and a Bloomberg/Kitware combined effort.

# Hot off the presses

## VS: Add support for C++ module internal partitions in VS 17.6 and newer

Code

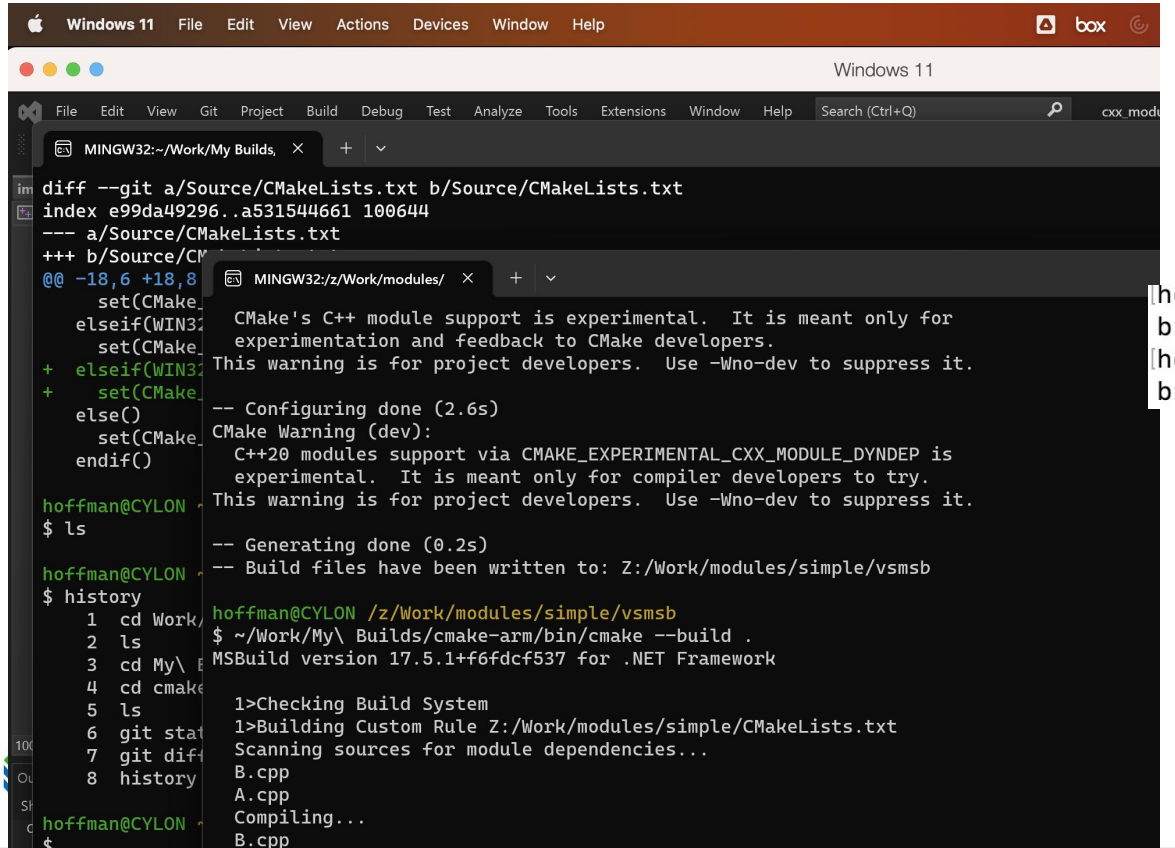
Open **Brad King** requested to merge `brad.king/cmake:vs-cxxmo…` into `master` 3 hours ago

**Overview** 2 | Commits 1 | Pipelines 1 | Changes 4 | 2 unresolved threads

VS 17.6 now implements `ScanSourceforModuleDependencies` using the same `cl /scanDependencies` scanner that our Ninja generator uses. It can distinguish module internal partitions from module interface units based on their content. Switch from `CompileAsCppModule` to `CompileAsCpp` for `CXX_MODULES` sources so that MSBuild can scan and classify them.

kitware

# Sidebar: Mac M1 with Parallels Windows ultimate test machine for CMake development




```
hoffman@caprica modules % ls llvm16-inst
bin     include lib     libexec share
hoffman@caprica modules % ls gcc-inst
bin     include lib     libexec share
```

# back to A.cpp msvc

```
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"
            }
         ]
      }
   ]
}
```

kitware

# back to B.cpp msvc

```
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": []
      }
   ]
}
```

kitware

# 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.

kitware

# 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
```

# Back to A.cxx and B.cxx with updated FILE_SET

The only acceptable types are now `HEADERS, CXX_MODULES` .

# 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.exe -GNinja ..
-- Configuring done
CMake Warning (dev):
  C++20 modules support via CMAKE_EXPERIMENTAL_CXX_MODULE_DYNDEP is
  experimental.  It is meant only for compiler developers to try.
This warning is for project developers.  Use -Wno-dev to suppress it.

-- Generating done
-- Build files have been written to: C:/Users/hoffman/Work/cxxmodules/cxx-modules-examples/simple/b
```

# Compile A.cpp and B.cpp with CMake MSVC/ninja

```
$ ninja -v
[1/6] C:\PROGRA~1\MIB055~1\2022\Preview\VC\Tools\MSVC\1432~1.313\bin\Hostx64\x64\cl.exe   /DWIN32 /D_WINDOWS /EHsc /Zi
/Ob0 /Od /RTC1 -MDd -std:c++20 -interface C:\Users\hoffman\Work\cxxmodules\cxx-modules-examples\simple\A.cpp -nologo -TP
-showIncludes -scanDependencies CMakeFiles\simple.dir\A.cpp.obj.ddi -FoCMakeFiles\simple.dir\A.cpp.obj
[2/6] C:\PROGRA~1\MIB055~1\2022\Preview\VC\Tools\MSVC\1432~1.313\bin\Hostx64\x64\cl.exe   /DWIN32 /D_WINDOWS /EHsc /Zi
/Ob0 /Od /RTC1 -MDd -std:c++20 -interface C:\Users\hoffman\Work\cxxmodules\cxx-modules-examples\simple\B.cpp -nologo -TP
-showIncludes -scanDependencies CMakeFiles\simple.dir\B.cpp.obj.ddi -FoCMakeFiles\simple.dir\B.cpp.obj
[3/6] C:\Users\hoffman\Work\cxxmodules\bencmake\ninja\bin\cmake.exe -E cmake_ninja_dyndep
--tdi=CMakeFiles\simple.dir\CXXDependInfo.json --lang=CXX --modmapfmt=msvc --dd=CMakeFiles\simple.dir\CXX.dd
@CMakeFiles\simple.dir\CXX.dd.rsp
[4/6] C:\PROGRA~1\MIB055~1\2022\Preview\VC\Tools\MSVC\1432~1.313\bin\Hostx64\x64\cl.exe  /nologo /TP   /DWIN32
/D_WINDOWS /EHsc /Zi /Ob0 /Od /RTC1 -MDd -std:c++20 -interface /showIncludes @CMakeFiles\simple.dir\B.cpp.obj.modmap
/FoCMakeFiles\simple.dir\B.cpp.obj /FdCMakeFiles\simple.dir\simple.pdb /FS -c
C:\Users\hoffman\Work\cxxmodules\cxx-modules-examples\simple\B.cpp
[5/6] C:\PROGRA~1\MIB055~1\2022\Preview\VC\Tools\MSVC\1432~1.313\bin\Hostx64\x64\cl.exe  /nologo /TP   /DWIN32
/D_WINDOWS /EHsc /Zi /Ob0 /Od /RTC1 -MDd -std:c++20 -interface /showIncludes @CMakeFiles\simple.dir\A.cpp.obj.modmap
/FoCMakeFiles\simple.dir\A.cpp.obj /FdCMakeFiles\simple.dir\simple.pdb /FS -c
C:\Users\hoffman\Work\cxxmodules\cxx-modules-examples\simple\A.cpp
[6/6] cmd.exe /C "cd . && C:\PROGRA~1\MIB055~1\2022\Preview\VC\Tools\MSVC\1432~1.313\bin\Hostx64\x64\lib.exe /nologo
/machine:x64 /out:simple.lib CMakeFiles\simple.dir\A.cpp.obj CMakeFiles\simple.dir\B.cpp.obj  && cd ."
```

kitware

# Compile A.cpp and B.cpp with CMake g++

```
$ ninja -v
[1/6] /home/bill/gcc-install/bin/c++   -std=gnu++20 -E -x c++
/home/bill/sf_hoffman/Work/cxxmodules/cxx-modules-examples/simple/A.cpp -MT CMakeFiles/simple.dir/A.cpp.o.ddi -MD -MF
CMakeFiles/simple.dir/A.cpp.o.ddi.d -fmodules-ts -fdep-file=CMakeFiles/simple.dir/A.cpp.o.ddi
-fdep-output=CMakeFiles/simple.dir/A.cpp.o -fdep-format=trtbd -o CMakeFiles/simple.dir/A.cpp.o.ddi.i
[2/6] /home/bill/gcc-install/bin/c++   -std=gnu++20 -E -x c++
/home/bill/sf_hoffman/Work/cxxmodules/cxx-modules-examples/simple/B.cpp -MT CMakeFiles/simple.dir/B.cpp.o.ddi -MD -MF
CMakeFiles/simple.dir/B.cpp.o.ddi.d -fmodules-ts -fdep-file=CMakeFiles/simple.dir/B.cpp.o.ddi
-fdep-output=CMakeFiles/simple.dir/B.cpp.o -fdep-format=trtbd -o CMakeFiles/simple.dir/B.cpp.o.ddi.i
[3/6] /home/bill/cxxm/bencmake/ninja/bin/cmake -E cmake_ninja_dyndep --tdi=CMakeFiles/simple.dir/CXXDependInfo.json
--lang=CXX --modmapfmt=gcc --dd=CMakeFiles/simple.dir/CXX.dd @CMakeFiles/simple.dir/CXX.dd.rsp
[4/6] /home/bill/gcc-install/bin/c++   -std=gnu++20 -MD -MT CMakeFiles/simple.dir/B.cpp.o -MF
CMakeFiles/simple.dir/B.cpp.o.d -fmodules-ts -fmodule-mapper=CMakeFiles/simple.dir/B.cpp.o.modmap -fdep-format=trtbd -x
c++ -o CMakeFiles/simple.dir/B.cpp.o -c /home/bill/sf_hoffman/Work/cxxmodules/cxx-modules-examples/simple/B.cpp
[5/6] /home/bill/gcc-install/bin/c++   -std=gnu++20 -MD -MT CMakeFiles/simple.dir/A.cpp.o -MF
CMakeFiles/simple.dir/A.cpp.o.d -fmodules-ts -fmodule-mapper=CMakeFiles/simple.dir/A.cpp.o.modmap -fdep-format=trtbd -x
c++ -o CMakeFiles/simple.dir/A.cpp.o -c /home/bill/sf_hoffman/Work/cxxmodules/cxx-modules-examples/simple/A.cpp
[6/6] : && /home/bill/cxxm/bencmake/ninja/bin/cmake -E rm -f libsimple.a && /usr/bin/ar qc libsimple.a
CMakeFiles/simple.dir/A.cpp.o CMakeFiles/simple.dir/B.cpp.o && /usr/bin/ranlib libsimple.a && :
```

# Compile A.cpp and B.cpp with CMake clang++

```
$ ninja -v
[1/6] "/Users/hoffman/Work/modules/llvm16-inst/bin/clang-scan-deps" -format=p1689 --
/Users/hoffman/Work/modules/llvm16-inst/bin/clang++   -std=c++20 -arch arm64 -isysroot
/Library/Developer/CommandLineTools/SDKs/MacOSX13.1.sdk -x c++ /Users/hoffman/Work/modules/simple/A.cpp -c -o
CMakeFiles/foo.dir/A.cpp.o -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
[2/6] "/Users/hoffman/Work/modules/llvm16-inst/bin/clang-scan-deps" -format=p1689 --
/Users/hoffman/Work/modules/llvm16-inst/bin/clang++   -std=c++20 -arch arm64 -isysroot
/Library/Developer/CommandLineTools/SDKs/MacOSX13.1.sdk -x c++ /Users/hoffman/Work/modules/simple/B.cpp -c -o
CMakeFiles/foo.dir/B.cpp.o -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
[3/6] "/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
[4/6] /Users/hoffman/Work/modules/llvm16-inst/bin/clang++   -std=c++20 -arch arm64 -isysroot
/Library/Developer/CommandLineTools/SDKs/MacOSX13.1.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
[5/6] /Users/hoffman/Work/modules/llvm16-inst/bin/clang++   -std=c++20 -arch arm64 -isysroot
/Library/Developer/CommandLineTools/SDKs/MacOSX13.1.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
[6/6] : && "/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/modules/llvm16-inst/bin/llvm-ranlib libfoo.a &&
"/Users/hoffman/Work/My Builds/cmake-ninja/bin/cmake" -E touch libfoo.a && :
```

# Compile A.cpp and B.cpp with CMake MSBuild

```
cmake --build .
MSBuild version 17.5.1+f6fdcf537 for .NET Framework

  1>Checking Build System
  1>Building Custom Rule Z:/Work/modules/simple/CMakeLists.txt
  Scanning sources for module dependencies...
  B.cpp
  A.cpp
  Compiling...
  B.cpp
  A.cpp
  foo.vcxproj -> Z:\Work\modules\simple\vsmsb\Debug\foo.lib
  1>Building Custom Rule Z:/Work/modules/simple/CMakeLists.txt
```

# New C++ Module Docs

## 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`

**Note:** Experimental. Gated by `CMAKE_EXPERIMENTAL_CXX_MODULE_CMAKE_API`

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.

`CXX_MODULE_HEADER_UNITS`

**Note:** Experimental. Gated by `CMAKE_EXPERIMENTAL_CXX_MODULE_CMAKE_API`

C++ header sources which may be imported by other C++ source code. This file set type may not have an `INTERFACE` scope except on `IMPORTED` targets.

kitware

# 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`

> **Note:** Experimental. Gated by `CMAKE_EXPERIMENTAL_CXX_MODULE_CMAKE_API`

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).

# CMAKE_CXX_SCAN_FOR_MODULES ¶

*New in version 3.26.*

Whether to scan C++ source files for module dependencies.

This variable is used to initialize the **CXX_SCAN_FOR_MODULES** property on all the targets. See that target property for additional information.

**Note:** This setting is meaningful only when experimental support for C++ modules has been enabled by the `CMAKE_EXPERIMENTAL_CXX_MODULE_CMAKE_API` gate.

# CXX_SCAN_FOR_MODULES

*New in version 3.26.*

`CXX_SCAN_FOR_MODULES` is a boolean specifying whether CMake will scan the source for C++ module dependencies. See also the **`CXX_SCAN_FOR_MODULES`** for target-wide settings.

When this property is set `ON`, CMake will scan the source 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 source at build time. When this property is unset, the **`CXX_SCAN_FOR_MODULES`** property is consulted.

Note that scanning is only performed if C++20 or higher is enabled for the target and the source uses the `CXX` language. Scanning for modules in sources belonging to file sets of type `CXX_MODULES` and `CXX_MODULES_HEADER_UNITS` is always performed.

**Note:** This setting is meaningful only when experimental support for C++ modules has been enabled by the `CMAKE_EXPERIMENTAL_CXX_MODULE_CMAKE_API` gate.

# Exporting Targets

```
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>`

> **Note:** Experimental. Gated by `CMAKE_EXPERIMENTAL_CXX_MODULE_CMAKE_API`

: 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.

# Kick the tires CMake C++20 Modules

kitware

# Using CMake with C++20 Modules

- Get CMake 3.26 or newer
  - Read the docs: https://github.com/Kitware/CMake/blob/master/Help/dev/experimental.rst
- Get MSVC 2022 preview, or Clang 16 or build a patched gcc
  - https://github.com/mathstuf/gcc/tree/p1689r5
- Setup your CMakeLists.txt

```
# turn feature on in CMake
set(CMAKE_EXPERIMENTAL_CXX_MODULE_CMAKE_API 3c375311-a3c9-4396-a187-3227ef642046")
# only need this for GNU now, clang and msvc ship with cmake
if (CMAKE_CXX_COMPILER_ID STREQUAL "GNU")
  include(gcc_modules.cmake) # on next slide
endif()
set(CMAKE_CXX_STANDARD 20)
```

kitware

# gcc_modules.cmake

```
set(CMAKE_EXPERIMENTAL_CXX_MODULE_DYNDEP 1)
string(CONCAT CMAKE_EXPERIMENTAL_CXX_SCANDEP_SOURCE
  "<CMAKE_CXX_COMPILER> <DEFINES> <INCLUDES> <FLAGS> -E -x c++ <SOURCE>"
  " -MT <DYNDEP_FILE> -MD -MF <DEP_FILE>"
  " -fmodules-ts -fdep-file=<DYNDEP_FILE> -fdep-output=<OBJECT> -fdep-format=trtbd"
  " -o <PREPROCESSED_SOURCE>")
set(CMAKE_EXPERIMENTAL_CXX_MODULE_MAP_FORMAT "gcc")
set(CMAKE_EXPERIMENTAL_CXX_MODULE_MAP_FLAG "-fmodules-ts -fmodule-mapper=<MODULE_MAP_FILE>
-fdep-format=trtbd -x c++")
```

kitware

# Read this blog:
# https://www.kitware.com/import-cmake-c20-modules/

Per target sc

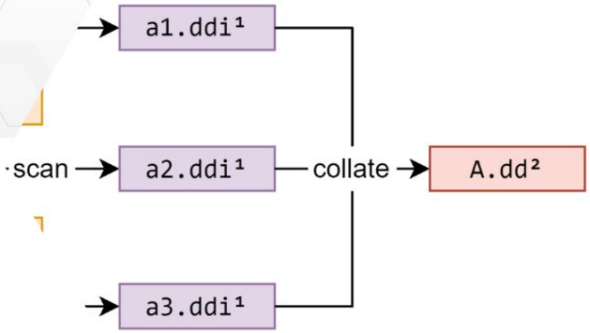
news › blog › post

## import CMake; C++20 Modules

January 19, 2023 Bill Hoffman, Ben Boeckel and Brad King

Work is underway to implement support for C++20 modules in CMake! Since the C++ standards committee started talking about adding modules to the C++ language, the CMake team at Kitware has been thinking about how they will be supported. Fortunately, CMake has supported Fortran modules since 2005. In 2015, support was added to the ninja build tool to support Fortran modules. In 2019 with news of modules being added to C++, the Kitware fork of ninja was rejoined with upstream ninja and dynamic dependencies were added to ninja. This blog describes the process that was taken and the current state of named C++ 20 modules in CMake. Header modules are not covered in this blog.

## Quick Introduction to C++ 20 Modules




compiler (i.e. cl -scanDe

E cmake_ninja_dynd

kitware

# CMake C++20 Module tests

Configure and build cmake from source with module testing enabled

```
cmake \
    -DCMake_TEST_MODULE_COMPILATION=named,shared,partitions,internal_partitions,export_bmi,install_bmi \
    -DCMake_TEST_MODULE_COMPILATION_RULES=/path/to/cmake/source/tree/.gitlab/ci/cxx_modules_rules_msvc.cmake \  # MSVC, gcc, or clang
    -DCMake_TEST_HOST_CMAKE=ON \
    -DCMAKE_EXPERIMENTAL_CXX_MODULE_CMAKE_API="2182bf5c-ef0d-489a-91da-49dbc3090d2a" \
    -S /path/to/cmake/source/tree \
    -B /path/to/build/tree  -GNinja
ninja
ctest -R CXXModules
```

The test suite is available as `RunCMake.CXXModules` (pass to `ctest -R`to run) and the sources live in `Tests/RunCMake/CXXModules`

kitware

# ctest -R CXXModules

```
531: Test timeout computed to be: 1500
531: -- compiler_introspection - PASSED
531: -- NoCXX - PASSED
531: -- NoCXX20 - PASSED
531: -- NoCXX20ModuleFlag - PASSED
531: -- FileSetModulesInterface - PASSED
531: -- FileSetModulesPrivate - PASSED
531: -- FileSetModulesPublic - PASSED
531: -- FileSetModulesInterfaceImported - PASSED
531: -- NotCXXSourceModules - PASSED
531: -- FileSetModuleHeaderUnitsInterface - PASSED
531: -- FileSetModuleHeaderUnitsPrivate - PASSED
531: -- FileSetModuleHeaderUnitsPublic - PASSED
531: -- FileSetModuleHeaderUnitsInterfaceImported -
PASSED
531: -- NotCXXSourceModuleHeaderUnits - PASSED
531: -- InstallBMI - PASSED
531: -- InstallBMIGenericArgs - PASSED
531: -- InstallBMIIgnore - PASSED
531: -- ExportBuildCxxModules - PASSED
531: -- ExportInstallCxxModules - PASSED
531: -- NinjaDependInfoFileSet - PASSED
531: -- NinjaDependInfoExport - PASSED
531: -- NinjaDependInfoBMIInstall - PASSED
531: -- examples/simple - PASSED
531: -- examples/simple-build - PASSED
531: -- FileSetModuleHeaderUnitsPrivate -
PASSED
53
531: -- examples/generated-build - PASSED
531: -- examples/generated-test - PASSED
531: -- examples/deep-chain - PASSED
531: -- examples/deep-chain-build - PASSED
531: -- examples/deep-chain-test - PASSED
531: -- examples/duplicate - PASSED
531: -- examples/duplicate-build - PASSED
531: -- examples/duplicate-test - PASSED
531: -- examples/circular - PASSED
531: -- examples/circular-build - PASSED
531: -- examples/scan_properties - PASSED
531: -- examples/scan_properties-build - PASSED
531: -- examples/scan_properties-test - PASSED
531: -- examples/library-shared - PASSED
531: -- examples/library-shared-build - PASSED
531: -- examples/library-shared-test - PASSED
531: -- examples/partitions - PASSED
531: -- examples/partitions-build - PASSED
531: -- examples/partitions-test - PASSED
531: -- examples/internal-partitions - PASSED
531: -- examples/internal-partitions-build - PASSED
531: -- examples/internal-partitions-test - PASSED
531: -- examples/install-bmi-test - PASSED
531: -- examples/install-bmi-and-interfaces - PASSED
531: -- examples/install-bmi-and-interfaces-build - PASSED
531: -- examples/install-bmi-and-interfaces-install - PASSED
531: -- examples/install-bmi-and-interfaces-test - PASSED
531: -- examples/export-interface-no-properties-install - PASSED
531: -- examples/export-interface-no-properties-install-build -
PASSED
531: -- examples/export-interface-no-properties-install-install -
PASSED
531: -- examples/export-interface-no-properties-install-test -
PASSED
531: -- examples/export-interface-install - PASSED
531: -- examples/export-interface-install-build - PASSED
531: -- examples/export-interface-install-install - PASSED
531: -- examples/export-interface-install-test - PASSED
531: -- examples/export-bmi-and-interface-install - PASSED
531: -- examples/export-bmi-and-interface-install-build - PASSED
531: -- examples/export-bmi-and-interface-install-install - PASSED
531: -- examples/export-bmi-and-interface-install-test - PASSED
1/1 Test #531: RunCMake.CXXModules .............   Passed   26.78
sec
The following tests passed:
            RunCMake.CXXModules
100% tests passed, 0 tests failed out of 1
Label Time Summary:
CMake    =  26.78 sec*proc (1 test)
run      =  26.78 sec*proc (1 test)
Total Test time (real) =  26.87 sec
```

kitware

# Module Examples in CMake

cmake/Tests/RunCMake/CXXModules/examples

# List of Examples

simple # very simple one exe main.cxx and importable.cxx

circular # meant to fail a imports b, b imports a

deep-chain # library a b c d e e import d d import c c import b b import a main import e

duplicate # use -D in a module same source but different exe

export-bmi-and-interface-build #export from the build tree using export

export-bmi-and-interface-install # install bmi and miu

export-interface-build # export to build tree

export-interface-install # install

export-interface-no-properties-build

export-interface-no-properties-install

generated # uses configure file to create the importable module file

install-bmi # installs a bmi

install-bmi-and-interfaces # install bmi and miu

internal-partitions # tests an internal partition

library # test a library and an executable

partitions # test partitions

public-req-private # public module imports private one

scan_properties # tests CXX_SCAN_FOR_MODULES

# Running the examples

- How to run them by yourself as standalone projects

```
export CMAKE_SRC="/home/hoffman/Work/My Builds/cmake"
mkdir build
cd build
cmake -DCMake_TEST_MODULE_COMPILATION_RULES="$CMAKE_SRC/.gitlab/ci/cxx_modules_rules_msvc.cmake"
-GNinja "$CMAKE_SRC/Tests/RunCMake/CXXModules/examples/internal-partitions"
```

# Take a look at what is in an examples test in CMake

```
pwd
/Users/hoffman/Work/My Builds/cmake/Tests/RunCMake/CXXModules/examples/simple
hoffman@caprica simple % ls
CMakeLists.txt   b           importable.cxx   main.cxx
hoffman@caprica simple %


cmake_minimum_required(VERSION 3.24)
project(cxx_modules_simple CXX)

include("${CMAKE_SOURCE_DIR}/../cxx-modules-rules.cmake")

add_executable(simple)
target_sources(simple
  PRIVATE
    main.cxx
  PRIVATE
    FILE_SET CXX_MODULES
      BASE_DIRS
        "${CMAKE_CURRENT_SOURCE_DIR}"
      FILES
        importable.cxx)
target_compile_features(simple PUBLIC cxx_std_20)

add_test(NAME simple COMMAND simple)
```

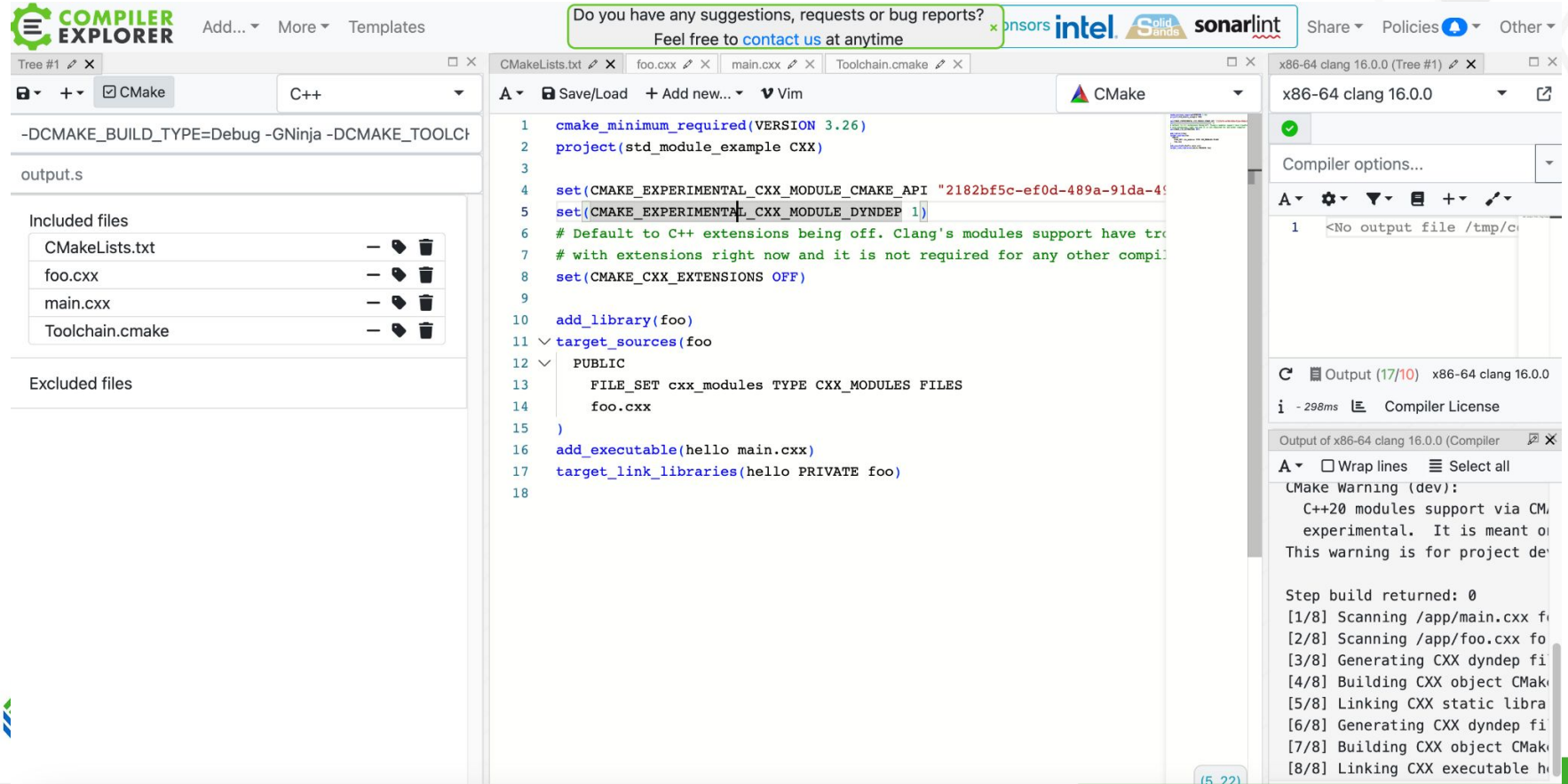
```
% export CMAKE_SRC="/Users/hoffman/Work/My Builds/cmake"
hoffman@caprica simple-b % cmake
-DCMake_TEST_MODULE_COMPILATION_RULES="$CMAKE_SRC/.gitlab/ci/cxx_modules_rules_clang.cmake" -GNinja
$CMAKE_SRC/Tests/RunCMake/CXXModules/examples/simple
-- The CXX compiler identification is Clang 16.0.0
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: /Users/hoffman/Work/modules/llvm16-inst/bin/clang++ - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
a246741c-d067-4019-a8fb-3d16b0c9d1d3
CMake Warning (dev) at CMakeLists.txt:7 (target_sources):
  CMake's C++ module support is experimental.  It is meant only for
  experimentation and feedback to CMake developers.
This warning is for project developers.  Use -Wno-dev to suppress it.

-- Configuring done (0.4s)
CMake Warning (dev):
  C++20 modules support via CMAKE_EXPERIMENTAL_CXX_MODULE_DYNDEP is
  experimental.  It is meant only for compiler developers to try.
This warning is for project developers.  Use -Wno-dev to suppress it.

-- Generating done (0.0s)
-- Build files have been written to: /Users/hoffman/Work/modules/cmake_example_builds/simple-b
```

```
hoffman@caprica simple-b % ninja -v
[1/6] "/Users/hoffman/Work/modules/llvm16-inst/bin/clang-scan-deps" -format=p1689 --
/Users/hoffman/Work/modules/llvm16-inst/bin/clang++   -std=c++20 -arch arm64 -isysroot
/Library/Developer/CommandLineTools/SDKs/MacOSX13.1.sdk -x c++ '/Users/hoffman/Work/My
Builds/cmake/Tests/RunCMake/CXXModules/examples/simple/main.cxx' -c -o CMakeFiles/simple.dir/main.cxx.o -MT
CMakeFiles/simple.dir/main.cxx.o.ddi -MD -MF CMakeFiles/simple.dir/main.cxx.o.ddi.d > CMakeFiles/simple.dir/main.cxx.o.ddi
[2/6] "/Users/hoffman/Work/modules/llvm16-inst/bin/clang-scan-deps" -format=p1689 --
/Users/hoffman/Work/modules/llvm16-inst/bin/clang++   -std=c++20 -arch arm64 -isysroot
/Library/Developer/CommandLineTools/SDKs/MacOSX13.1.sdk -x c++ '/Users/hoffman/Work/My
Builds/cmake/Tests/RunCMake/CXXModules/examples/simple/importable.cxx' -c -o CMakeFiles/simple.dir/importable.cxx.o -MT
CMakeFiles/simple.dir/importable.cxx.o.ddi -MD -MF CMakeFiles/simple.dir/importable.cxx.o.ddi.d >
CMakeFiles/simple.dir/importable.cxx.o.ddi
[3/6] "/Users/hoffman/Work/My Builds/cmake-clang-mod/bin/cmake" -E cmake_ninja_dyndep
--tdi=CMakeFiles/simple.dir/CXXDependInfo.json --lang=CXX --modmapfmt=clang --dd=CMakeFiles/simple.dir/CXX.dd
@CMakeFiles/simple.dir/CXX.dd.rsp
[4/6] /Users/hoffman/Work/modules/llvm16-inst/bin/clang++   -std=c++20 -arch arm64 -isysroot
/Library/Developer/CommandLineTools/SDKs/MacOSX13.1.sdk -MD -MT CMakeFiles/simple.dir/importable.cxx.o -MF
CMakeFiles/simple.dir/importable.cxx.o.d @CMakeFiles/simple.dir/importable.cxx.o.modmap -o
CMakeFiles/simple.dir/importable.cxx.o -c '/Users/hoffman/Work/My
Builds/cmake/Tests/RunCMake/CXXModules/examples/simple/importable.cxx'
[5/6] /Users/hoffman/Work/modules/llvm16-inst/bin/clang++   -std=c++20 -arch arm64 -isysroot
/Library/Developer/CommandLineTools/SDKs/MacOSX13.1.sdk -MD -MT CMakeFiles/simple.dir/main.cxx.o -MF
CMakeFiles/simple.dir/main.cxx.o.d @CMakeFiles/simple.dir/main.cxx.o.modmap -o CMakeFiles/simple.dir/main.cxx.o -c
'/Users/hoffman/Work/My Builds/cmake/Tests/RunCMake/CXXModules/examples/simple/main.cxx'
[6/6] : && /Users/hoffman/Work/modules/llvm16-inst/bin/clang++ -arch arm64 -isysroot
/Library/Developer/CommandLineTools/SDKs/MacOSX13.1.sdk -Wl,-search_paths_first -Wl,-headerpad_max_install_names
CMakeFiles/simple.dir/main.cxx.o CMakeFiles/simple.dir/importable.cxx.o -o simple   && :
```

```
cmake_minimum_required(VERSION 3.24)
project(cxx_modules_export_bmi_and_interfaces CXX)

include("${CMAKE_SOURCE_DIR}/../cxx-modules-rules.cmake")

add_library(export_bmi_and_interfaces STATIC)
target_sources(export_bmi_and_interfaces
  PRIVATE
    forward.cxx
  PRIVATE
    FILE_SET modules_private TYPE CXX_MODULES
      BASE_DIRS
        "${CMAKE_CURRENT_SOURCE_DIR}"
      FILES
        private.cxx
  PUBLIC
    FILE_SET modules TYPE CXX_MODULES
      BASE_DIRS
        "${CMAKE_CURRENT_SOURCE_DIR}"
      FILES
        importable.cxx)
target_compile_features(export_bmi_and_interfaces PUBLIC
cxx_std_20)

install(TARGETS export_bmi_and_interfaces
  EXPORT CXXModules
  FILE_SET modules DESTINATION "lib/cxx/miu"
  CXX_MODULES_BMI DESTINATION "lib/cxx/bmi")

install(EXPORT CXXModules
  NAMESPACE CXXModules::
  DESTINATION "lib/cmake/export_bmi_and_interfaces"
  FILE "export_bmi_and_interfaces-targets.cmake"
  CXX_MODULES_DIRECTORY
"export_bmi_and_interfaces-cxx-modules")
file(WRITE
"${CMAKE_CURRENT_BINARY_DIR}/export_bmi_and_interfaces-config
.cmake"

"include(\"\${CMAKE_CURRENT_LIST_DIR}/export_bmi_and_interfac
es-targets.cmake\")
set(\${CMAKE_FIND_PACKAGE_NAME}_FOUND 1)
")
install(FILES
"${CMAKE_CURRENT_BINARY_DIR}/export_bmi_and_interfaces-config
.cmake"
  DESTINATION "lib/cmake/export_bmi_and_interfaces")
```

```
hoffman@caprica export-bmi-and-interface-install-b % ninja install
[0/1] Install the project...
-- Install configuration: ""
-- Installing:
/Users/hoffman/Work/modules/cmake_example_builds/export-bmi-and-interface-install-b-install/lib/libexport_bmi_and_interfaces.a
-- Installing:
/Users/hoffman/Work/modules/cmake_example_builds/export-bmi-and-interface-install-b-install/lib/cxx/miu/importable.cxx
-- Installing:
/Users/hoffman/Work/modules/cmake_example_builds/export-bmi-and-interface-install-b-install/lib/cxx/bmi/importable.pcm
-- Installing:
/Users/hoffman/Work/modules/cmake_example_builds/export-bmi-and-interface-install-b-install/lib/cmake/export_bmi_and_interfaces/export_bmi_and_interfaces-targets.cmake
-- Installing:
/Users/hoffman/Work/modules/cmake_example_builds/export-bmi-and-interface-install-b-install/lib/cmake/export_bmi_and_interfaces/export_bmi_and_interfaces-targets-noconfig.cmake
-- Installing:
/Users/hoffman/Work/modules/cmake_example_builds/export-bmi-and-interface-install-b-install/lib/cmake/export_bmi_and_interfaces/export_bmi_and_interfaces-cxx-modules/cxx-modules.cmake
-- Installing:
/Users/hoffman/Work/modules/cmake_example_builds/export-bmi-and-interface-install-b-install/lib/cmake/export_bmi_and_interfaces/export_bmi_and_interfaces-cxx-modules/cxx-modules-noconfig.cmake
-- Installing:
/Users/hoffman/Work/modules/cmake_example_builds/export-bmi-and-interface-install-b-install/lib/cmake/export_bmi_and_interfaces/export_bmi_and_interfaces-cxx-modules/target-export_bmi_and_interfaces-noconfig.cmake
-- Installing:
/Users/hoffman/Work/modules/cmake_example_builds/export-bmi-and-interface-install-b-install/lib/cmake/export_bmi_and_interfaces/export_bmi_and_interfaces-config.cmake
```

# https://godbolt.org/z/84hrMa6fz



# static checks in CMake

https://www.kitware.com/static-checks-with-cmake-cdash-iwyu-clang-tidy-lwyu-cpplint-and-cppcheck/

Ride along with the compiler.  Restriction is the compiler must match the tool (clang versions) to read BMI files.

# P2473R1 https://wg21.link/P2473R1

For better observability of the build, the output of this tool should be stored in a file named module_config.json in the same location where compile_commands.json is currently saved.

Experimental implementation:

https://gitlab.kitware.com/kyle.edwards/cmake/-/tree/module-config

kitware

# CMake C++Modules next steps

- Header units - See Daniel's talk hopefully never…
- Add the import part to consume the export that is currently implemented
- Discover and consume modules from other build systems
- BMI flag compatibility checks need to be added

# CMake C++Modules status

- Support for p1689r5 in official releases
  - Visual studio 2022 preview
  - Clang 16
- GCC - patch for named modules
  - https://github.com/mathstuf/gcc/tree/p1689r5
  - working on upstreaming
- CMake 3.26 or newer has experimental support
- Experimental to full support when when gcc releases with the patch working on this now
  - ~~set(CMAKE_EXPERIMENTAL_CXX_MODULE_CMAKE_API "2182bf5c-ef0d-489a-91da-49dbc3090d2a")~~

kitware

# Special Thanks

Bret Brown

Daniel Ruoso

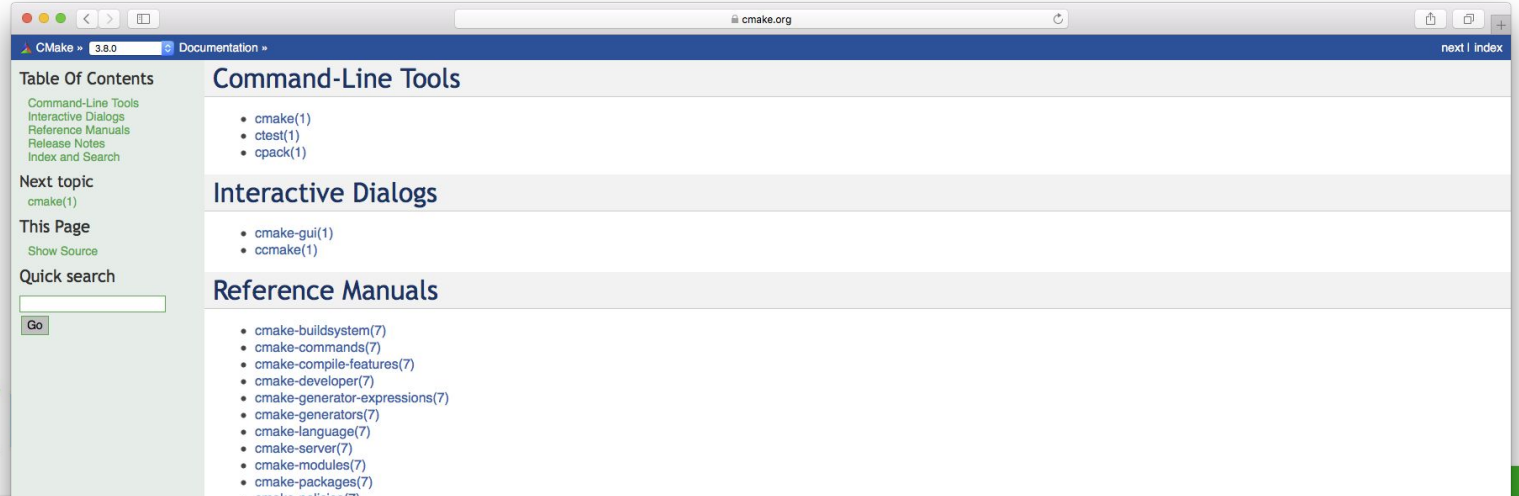
Bloomberg

Engineering

kitware

# 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



# Kitware is Hiring C++ Developers