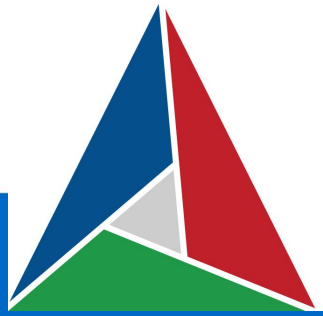


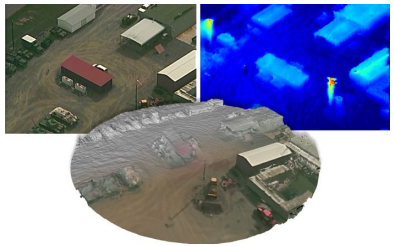
CMake

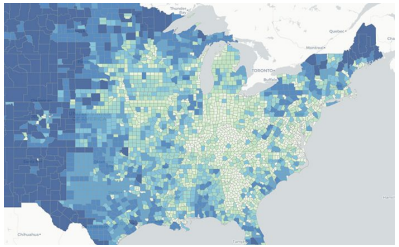
import CMake;
CMake and C++20 Modules

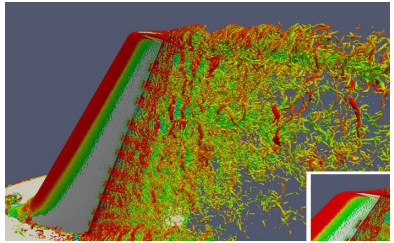
Bill Hoffman, CTO @ Kitware creator of CMake

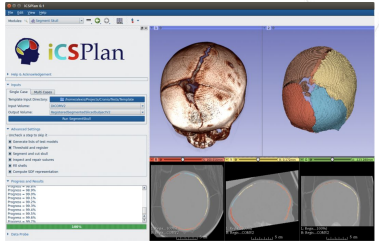
kitware

Kitware Overview / Built on open source









Computer Vision

Data and Analytics

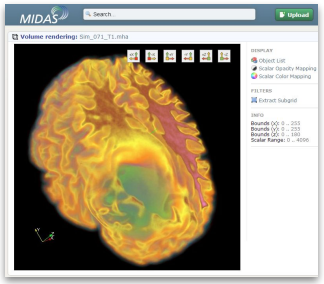
Scientific Computing

Medical Computing

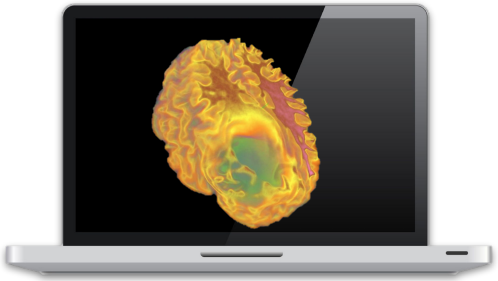
Software Solutions

kitware

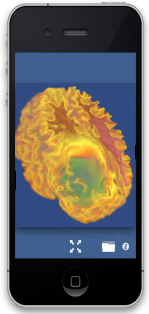
Applications / Universal Platforms



Web



Desktop



Mobile

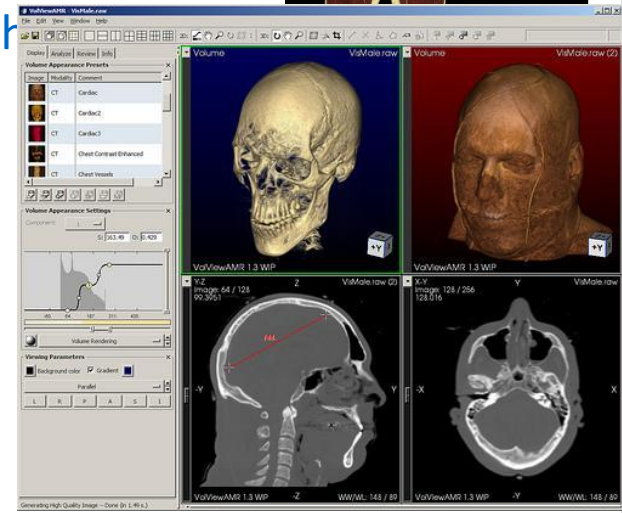
Cloud /HPC

kitware
Platforms

3D Slicer
ParaView
KWIVER
mstk
VTK
Pulse Physiology Engine
CMake
Resonant
tomviz
ITK

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 th Visible Human Project
  - Data CT/MR/Slice 1994/1995
  - Code (ITK) 1999
    - Cmake Release-1-0 branch created in 2001



kitware

How CMake Changes The Way We Build C++

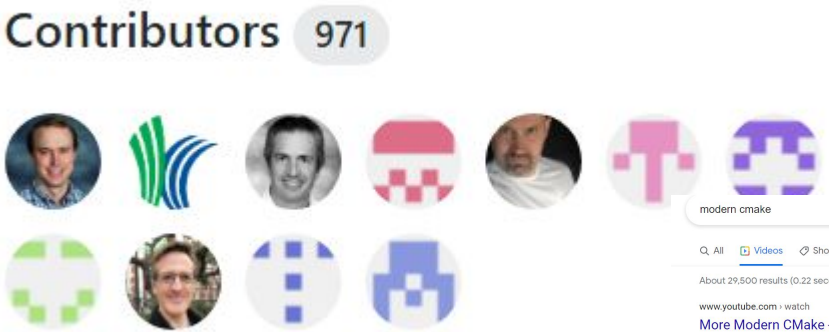
- Boost aims to give C++ a set of useful libraries like Java, Python, and C#
- CMake aims to give C++ compile portability like the compile once and run everywhere of Java, Python, and C#
  - Same build tool and files for all platforms
  - Easy to mix both large and small libraries

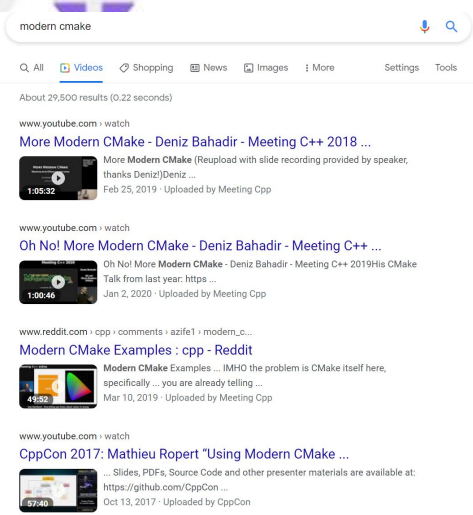


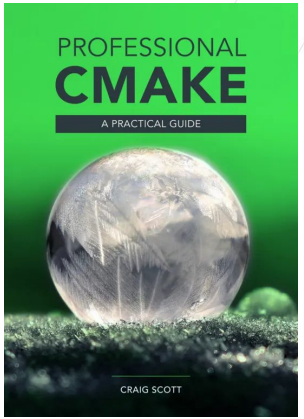


kitware

CMake is a Community Effort







kitware

Bloomberg

Engineering

- Improved XCOFF support in CMake
- Improved CMake's file-based API to include more information about installed components
- C++ 20 Modules! (in progress)

kitware

Unlikely, but cool contributors

Minecraft CMake Collaboration

Zack Galbreath, Kyle Edwards, Brad King, Robert Maynard and Bill Hoffman on November 4, 2020

Tags: CMake , Software Process

`CMakePresets.json` `CMakeUserPresets.json` VS Module support

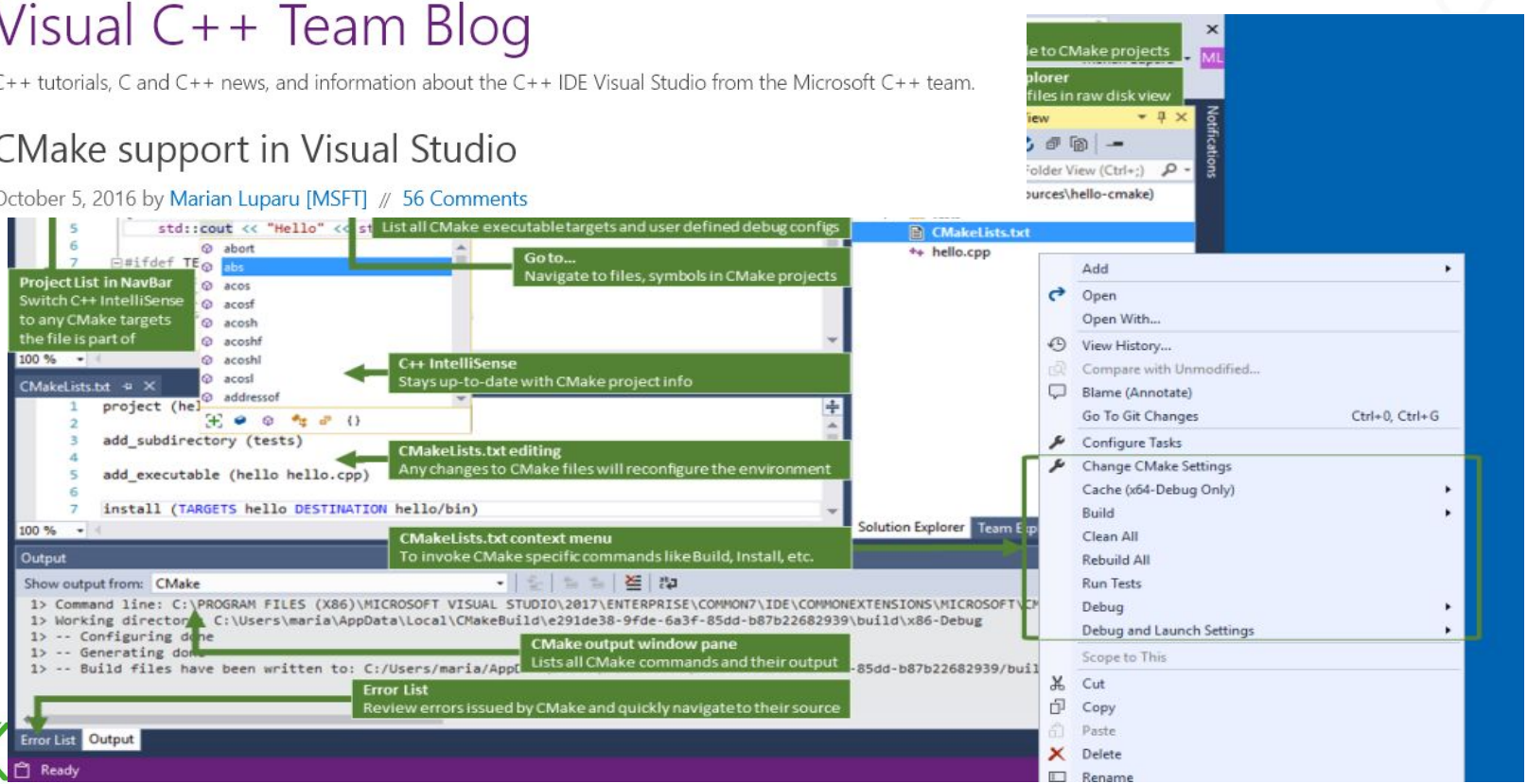
kitware

Visual C++ Team Blog

C++ tutorials, C and C++ news, and information about the C++ IDE Visual Studio from the Microsoft C++ team.

CMake support in Visual Studio

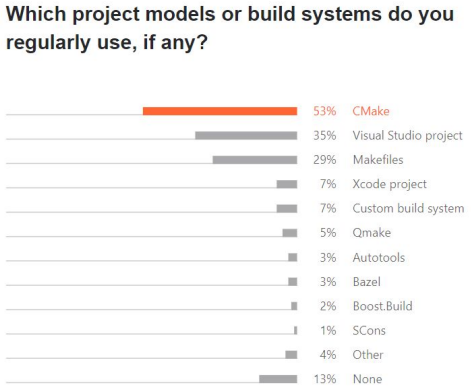
October 5, 2016 by Marian Luparu [MSFT] // 56 Comments



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
- New compiler support
  - clang
  - gcc versions

Jetbrains IDE- CMake is the most popular build tool at 53%



Up from up from 42% the previous year.

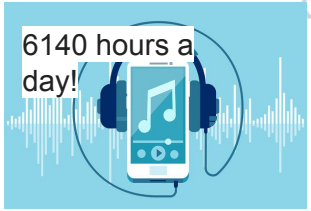
- Job openings requiring CMake experience, March, 2019 Indeed.com, 464 jobs, at Tesla Motors, DCS Corp, Mindsource, Quanergy, ...LinkedIn.com, 486 jobs, at Samsung, Johnson Controls, Apple, Uber, Toyota, Microsoft ...

kitware

CMake

5 million CMake downloads so far this year for total dl size greater than 100 TiB

13809 downloads per day, for a daily rate of 307.3GiB



The Qt Company Decides To Deprecate The Qbs Build System, Will Focus On CMake & QMake

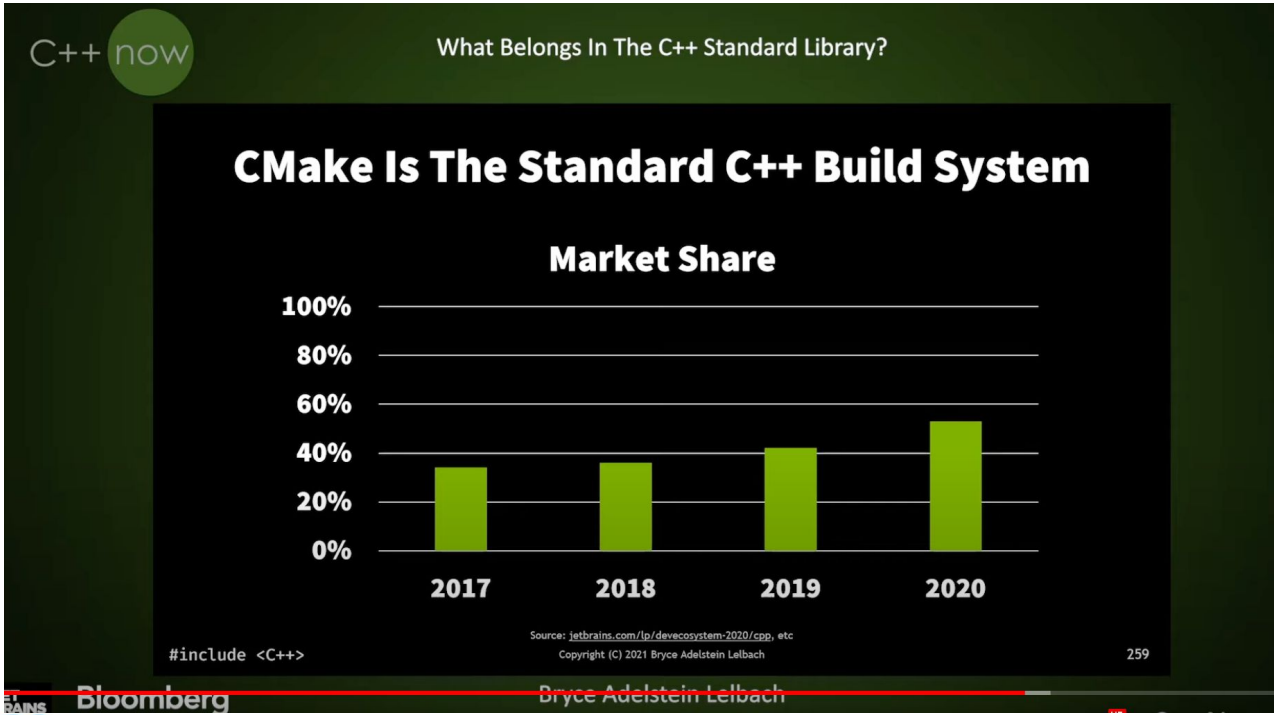
Written by Michael Larabel in Qt on 29 October 2018 at 08:23 AM EDT. 62 Comments

While Qt's Qbs build system was once planned as the default build system for Qt6 and shaping up to be the de facto successor to QMake, there is a change of course with The Qt Company now announcing they are deprecating this custom build system.

In recent months Qbs for Qt 6 began looking less certain and now The Qt Company has announced they are going to deprecate Qbs. From talking with their customers, they decided to focus on QMake and CMake.

kitware

"*you want a standard C++ build system, you got one it's called CMake, resistance is futile just use CMake*" - Bryce Adelstein C++ Now Jul 2021



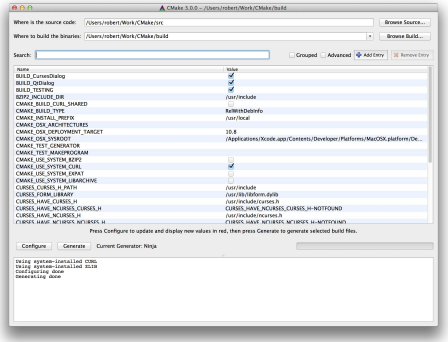
Why is CMake Popular

It does a lot with a little!

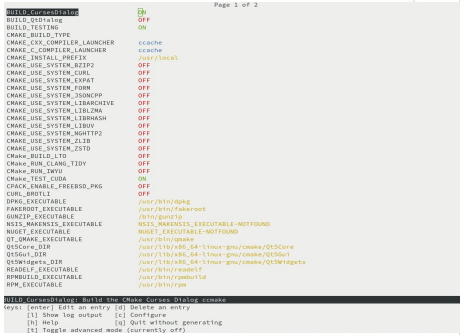
- add_library()
- add_executable()
- add_test()

A lot happens with those simple commands! (shared libraries, static libraries, many compilers/IDEs, and more)

CMake Workflow



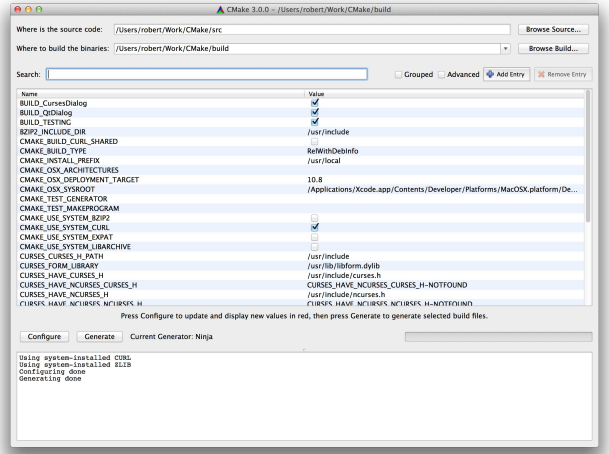

```
~/W/c/tutorial_build $ cmake ../src/Tests/Tutorial/Complete/
```

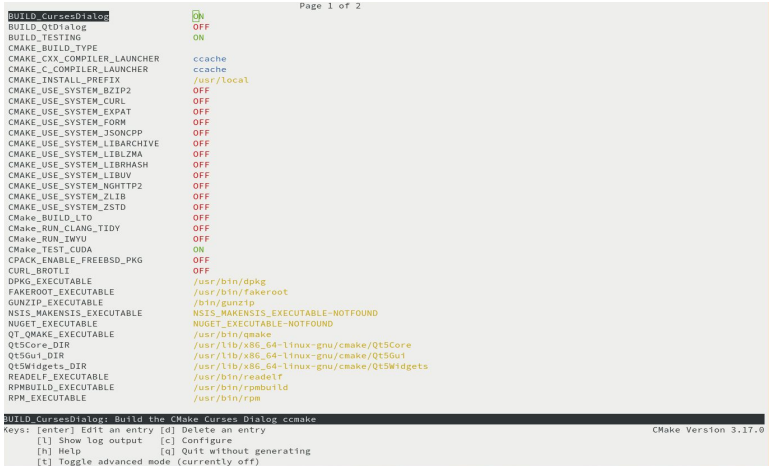


kitware

Running CMake

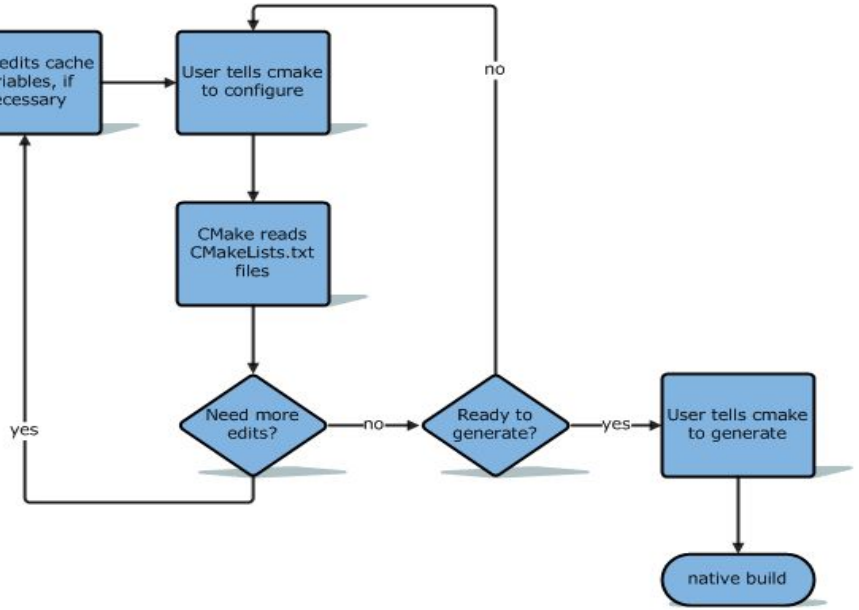
- cmake-gui (the Qt gui)
- ccmake (the terminal cli)
- cmake (non-interactive command line)





kitware

Running CMake



“Usage Requirements” aka Modern CMake

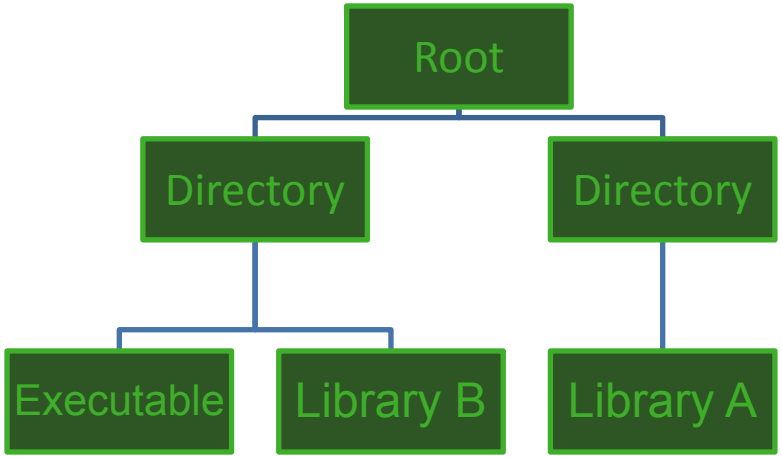
Modern style: target-centric

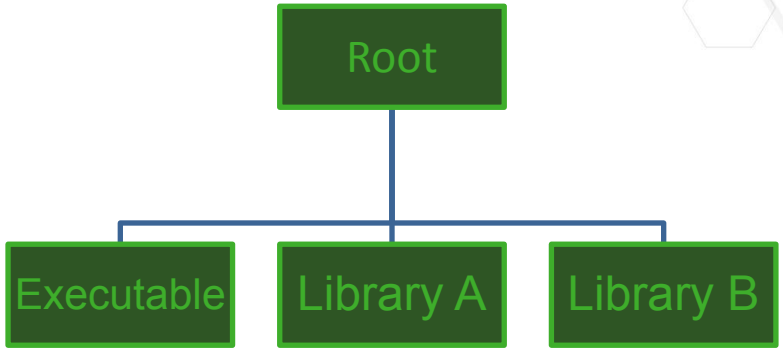
```
target_include_directories(example PUBLIC "inc")
```

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

Usage Requirements

```
target_link_libraries(trunk PUBLIC root)
target_link_libraries(leaf PUBLIC trunk)
```

```
/usr/bin/c++ -fPIC -shared -Wl,-soname,libleaf.so
             -o libleaf.so leaf.cxx.o libtrunk.so libroot.so
```

```
target_link_libraries(trunk PRIVATE root)
target_link_libraries(leaf PUBLIC trunk)
```

```
/usr/bin/c++ -fPIC -shared -Wl,-soname,libleaf.so
             -o libleaf.so leaf.cxx.o libtrunk.so
```

kitware

Unity / Jumbo Builds

-Control grouping size with CMAKE_UNITY_BUILD_BATCH_SIZE

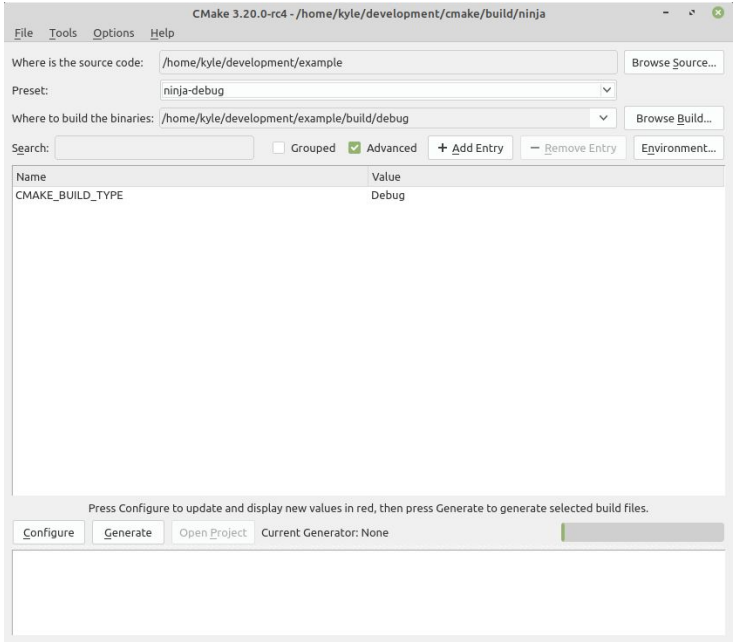
(of course cxx modules might reduce this utility)

Presets

- Allows common configuration flags (variables, build directory, generator, etc.) for a project to be stored in a JSON file for reuse
  - CMakePresets.json - version controlled, for sharing between users
  - CMakeUserPresets.json - not version controlled, for local machine-specific or user-specific use

Presets Example

```
{
  "version": 3,
  "configurePresets": [
    {
      "name": "ninja-debug",
      "generator": "Ninja",
      "binaryDir": "${sourceDir}/build/debug",
      "cacheVariables": {
        "CMAKE_BUILD_TYPE": "Debug"
      }
    }
  ]
}
```



kitware

Precompiled Headers

- Speed up compilation by creating partially processed header files
- Use those when compiling instead of repeatedly parsing header files

Precompiled Headers

```
add_library(leaf SHARED leaf.cxx)
target_precompile_headers(leaf
  PRIVATE
    <iostream>
    <vector>
    <unordered_map>
  INTERFACE
    "leaf.h")
```

- Multi Config Ninja
- CMAKE_<LANG>_COMPILER_LAUNCHER via ENV

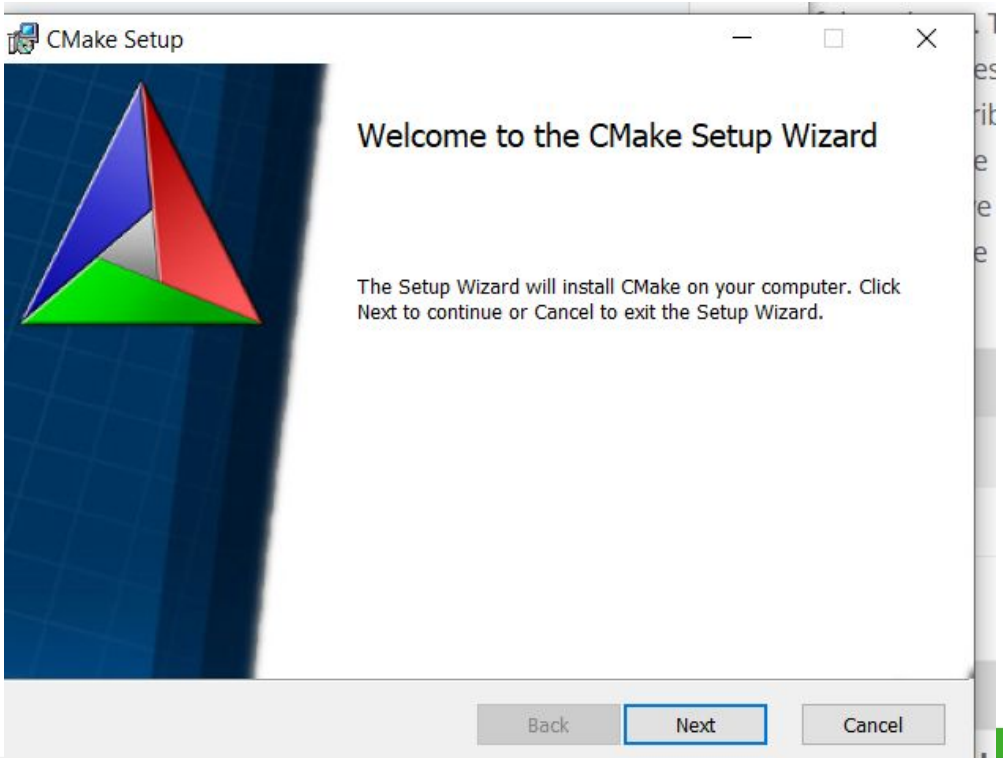
```
export CMAKE_CXX_COMPILER_LAUNCHER=ccache
cmake -G “Ninja Multi-Config” -S ./src -B ./build
...
cmake --build ./build --config Debug -j10 -v
...
/usr/bin/ccache gcc ...
...
ctest --build-config Debug -j10
```

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)
```

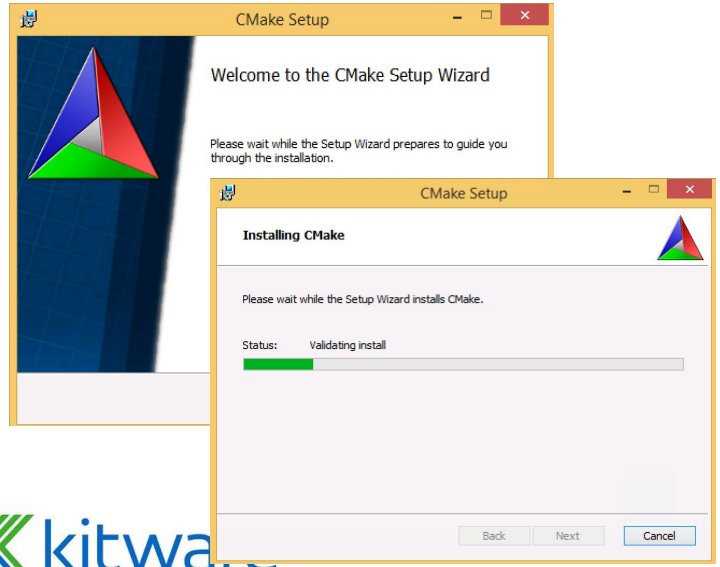
Packaging (dark art… or simple CMake command)

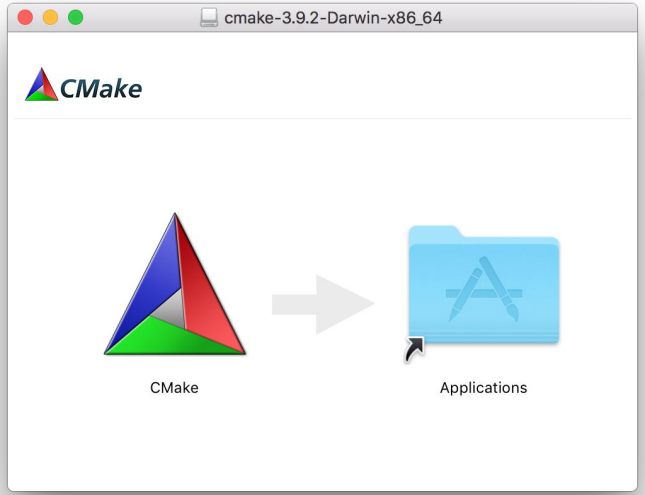


kitware

CPack

- CPack is bundled with CMake
- Creates professional platform specific installers





CPack Features

- Supports CMake-based and non-CMake-based projects
- Unix
  - TGZ and self-extracting TGZ (STGZ)
- Windows
  - WiX – MSI installers
  - NullSoft Scriptable Install System (NSIS / NSIS64)
- Mac OSX
  - DragNDrop
  - PackageMaker
- Deb
  - Debian packages
- RPM
  - RPM package manager

Using CPack

- On Windows install command line ZIP program, NSIS, and WiX
- Setup your project to work with cpack
  - Get `make install` to work
    - install(…)
    - make sure your executables work with relative paths and can work from any directory
  - Set cpack option variables if needed
  - `include(CPack)`

Testing with CMake

- Testing needs to be enabled by calling `include(CTest)` or `enable_testing()`

```
add_test(NAME testname
         COMMAND exename arg1 arg2 ...)
```

  - Executable should return 0 for a test that passes

- ctest – an executable that is distributed with cmake that can run tests in a project

Running CTest

- Run ctest at the top of a binary directory to run all tests

```
$ ctest
Test project /tmp/example/bin
    Start 1: case1
1/1 Test #1: case1 .............................   Passed    0.00 sec
    Start 2: case2
2/2 Test #2: case2 .............................   Passed    0.00 sec

100% tests passed, 0 tests failed out of 2

Total Test time (real) =   0.01 sec
```

Running CTest

- `-j` option allows you to run tests in parallel
- `-R` option allows you to choose a test
- `-VV` for more verbose output
- `--rerun-failed` to repeat failed tests

- `ctest --help` for more information

GoogleTest integration

```
include(GoogleTest)
add_executable(tests tests.cpp)
target_link_libraries(tests GTest::GTest)
```

- gtest_discover_tests: added in CMake 3.10.
  - CMake asks the test executable to list its tests. Finds new tests without rerunning CMake.

```
gtest_discover_tests(tests)
```

CTest and multi core tests

- PROCESSOR_AFFINITY - when supported ties processes to specific processors

```
set_tests_properties(myTest PROPERTIES
                     PROCESSOR_AFFINITY ON
                     PROCESSORS 4)
```

CDash https://my.cdash.org/

Scan Build — 1 build

Site	Build Name	Update Revision	Configure Error	Configure Warn	Build Error	Build Warn	Start Time
elysium-linux.kitware	Linux-clang-scanbuild	8c3071	0	0	0	0	10 hours ago

Nightly Expected — 96 of 108 builds

Site	Build Name	Update Revision	Configure Error	Configure Warn	Build Error	Build Warn	Test Not Run	Test Fail	Test Pass	Start Time
trinsic.kitware	vs14-64-ninja	8c3071	0	0	0	0	0	2^{+2}	479_{-2}	13 hours ago
trinsic.kitware	mingw64-make	8c3071	0	0	0	0	0	1^{+1}	449_{-1}	6 hours ago
hythloth.kitware	Linux-ninja-gcov	8c3071	0	0	0	0	0	1	519	7 hours ago
trinsic.kitware	vs14-64-ide-v90	8c3071	0	0	0	0	0	1^{+1}_{-1}	387_{-1}	8 hours ago
vesper.kitware	watcom	8c3071	0	0	0	0	0	1^{+1}	368_{-1}	8 hours ago
trinsic.kitware	vs9-64-ide	8c3071	0	0	0	0	0	1^{+1}	381_{-1}	9 hours ago
vesper.kitware	bcc55	8c3071	0	0	0	0	0	1^{+1}	370_{-1}	9 hours ago
dash3win7.kitware	vs14-64	8c3071	0	0	0	0	0	1^{+1}_{-1}	460^{+1}_{-1}	9 hours ago
dashsun1.kitware.com	Solaris-10-8.11_Oracle-12.3	8c3071	0	0	0	0	0	0	415	1 hour ago

kitware

Enter C++ Modules

CMake Devs- Be afraid, Be very afraid….

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() { }
```

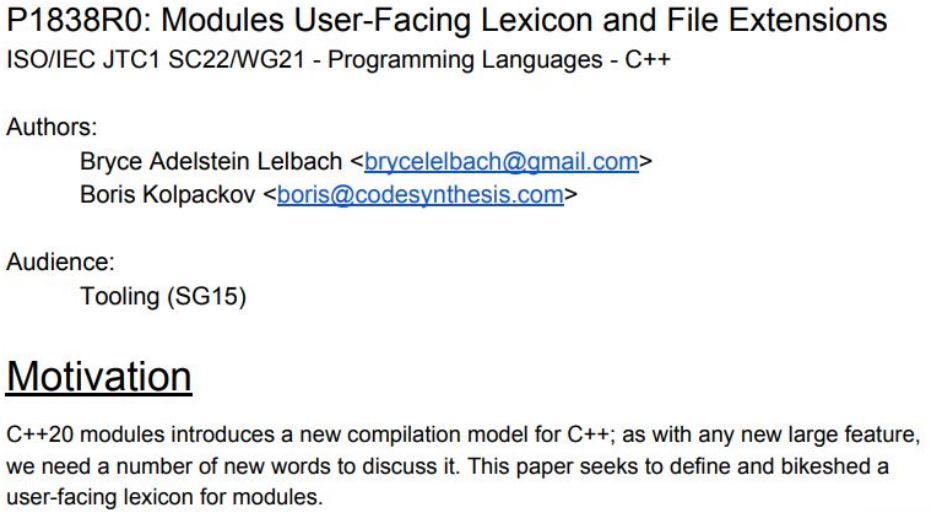
```
A.cpp:

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

Built Module Interface (BMI)

- MSVC
  - .ifc file
- G++
  - .gcm file
- 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

CMake has 16 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
- Patches made to ninja build tool now upstreamed
- The dynamic dependency collator inside CMake

Not going to do it alone

From Fortran experience modules require parsing Fortran code, C++ modules will require parsing C++

CMake needs help from compilers and standards to get this done

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
```

---

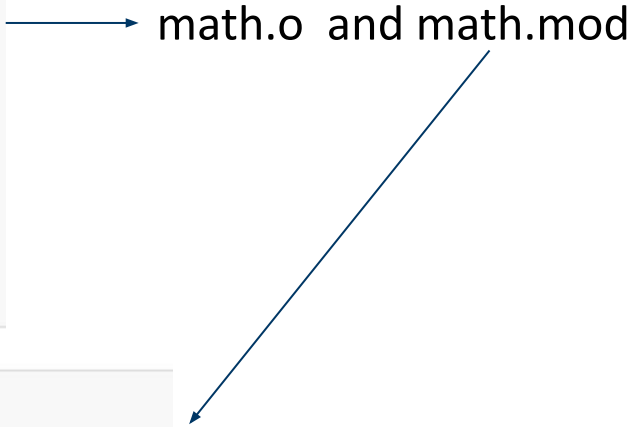
https://mathstuf.fedorapeople.org/fortran-modules/fortran-modules.html

kitware

How Fortran Modules Work

r

```
module math
contains
  function add(a,b)
    real :: add
    real, intent(in) :: a
    real, intent(in) :: b
    add = a + b
  end function
end module
```

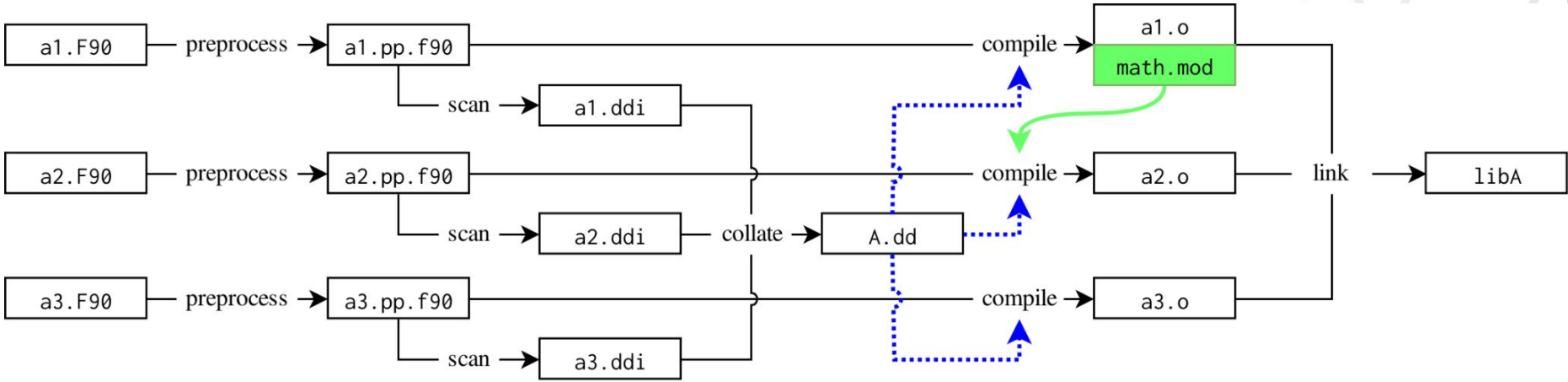



```
program main
  use math
  print *,'sum is',add(1., 2.)
end program
```

main.o needs math.mod

kitware

Build graph for single target

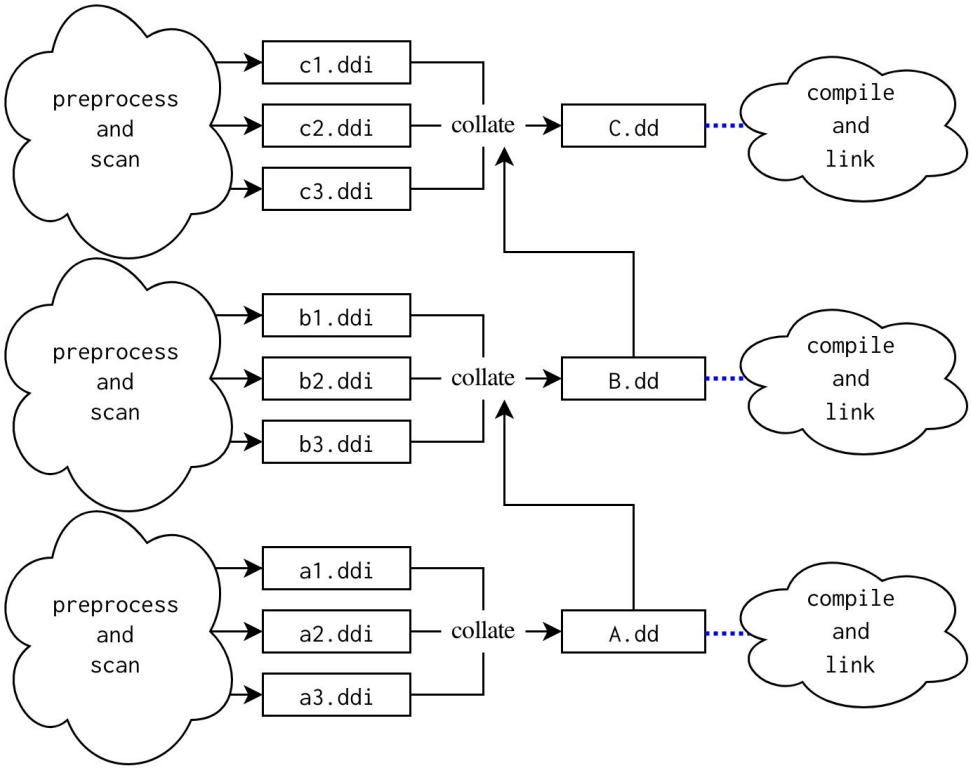


When `A.dd` is up-to-date and read, `ninja` updates its build graph and it learns:

- `a3.F90` and `a1.F90` may be compiled in parallel
- `a2.F90` must wait for the compilation of `a1.F90` for `math.mod` to created.
- If `a1.F90` changes, `a2.F90` may need recompiled, but `a3.F90` will never need to be.
- If `a1.F90` is modified and recompiled, if `math.mod` is not changed, then `a2.F90` is still considered up-to-date and not recompiled unnecessarily.
- If `a3.F90` is changed to gain a dependency on `math.mod`, the requirement on the `A.dd` generation means that the new dependency will be discovered and that the `a1.F90` must be checked for changes as well.

kitware

Build graph multiple targets



The `.dd` files for direct dependent targets are additional inputs to the target's `.dd` file. This allows modules contained within a source file in library `B` to be used in a source file in library `C` while still getting dependencies correct.

kitware

http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2021/p1689r4.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 P1689R4, 2021-06-14

Table of Contents

Document number	ISO/IEC/JTC1/SC22/WG21/P1689R4
Date	2021-06-14
Reply-to	Ben Boeckel, Brad King, ben.boeckel@kitware.com, brad.king@kitware.com
Audience	EWG (Evolution), SG15 (Tooling)

kitware

Where we are today

- Visual studio 2022 preview
  - module scanning support
- GCC - patch for named modules
  - https://github.com/mathstuf/gcc/tree/p1689r5
- CMake - module work in CMake master

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

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

back to A.cpp g++

```
c++   -std=gnu++20 -MD -MF A.cpp.o.ddi.d -E -x c++ A.cpp -MT A.cpp.o.ddi
-fmodules-ts -fdep-file=A.cpp.o.ddi -fdep-output=A.cpp.o -fdep-format=trtbd
-o A.cpp.o.ddi.i
```

```
{
"rules": [
{
"primary-output": "CMakeFiles/simple.dir/A.cpp.o",
"provides": [
{
"logical-name": "A"
}
],
"requires": [
{
"logical-name": "B"
}
]
}
],
"version": 0,
"revision": 0
}
```

kitware

back to B.cpp g++

```
c++   -std=gnu++20 -MD -MF A.cpp.o.ddi.d -E -x c++ A.cpp -MT A.cpp.o.ddi
-fmodules-ts -fdep-file=A.cpp.o.ddi -fdep-output=A.cpp.o -fdep-format=trtbd
-o A.cpp.o.ddi.i
```

```
{
"rules": [
{
"primary-output": "B.cpp.o",
"provides": [
{
"logical-name": "B"
}
],
"requires": [
]
}
],
"version": 0,
"revision": 0
}
```

kitware

Basic support for named modules is there for VS and g++(patched)

FILE_SET

Quick look at a new feature in 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 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

Install Header Only Libraries (before FILE SET)

```
set(Eigen_headers
  src/eigen.h
  src/vector.h
  src/matrix.h
  )
add_library(Eigen INTERFACE ${Eigen_headers})
target_include_directories(Eigen INTERFACE
  $<BUILD_INTERFACE:${CMAKE_CURRENT_SOURCE_DIR}/src>
  $<INSTALL_INTERFACE:include/Eigen>
)
install(TARGETS Eigen EXPORT eigenExport)
install(EXPORT eigenExport NAMESPACE Upstream::   DESTINATION lib/cmake/Eigen
)
install(FILES ${Eigen_headers}   DESTINATION include/Eigen
)
add_executable(exe1 exe1.cpp)
target_link_libraries(exe1 Eigen)
```

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 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
```

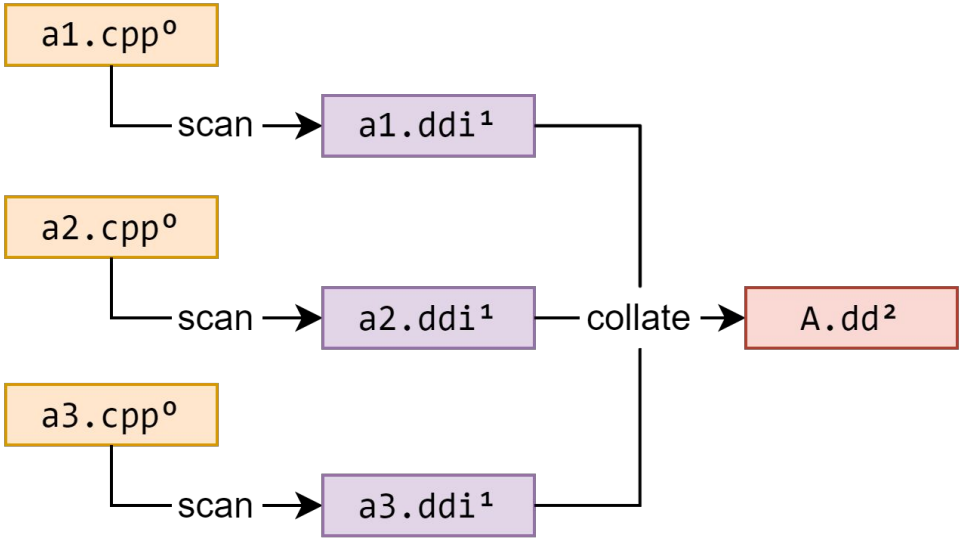
Compile A.cpp and B.cpp with CMake MSVC

```
$ ninja -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 ."
```

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 && :
```

Per target scanning



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

collate: cmake -E cmake_ninja_dyndep

kitware

Processes	Files produced
CMake	rules.ninja build.nina CXXDependInfo.json (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) TU.modmap (passed to compiler paths to BMI) CXXModules.json (cmake readable ver of CXX.dd)
CC compile per TU (in order based on dyndep)	.o .lib .exe, etc

kitware

Named Module Types

- "module unit" is a translation unit that contains a module-declaration. A "named module" is the collection of module units with the same module-name.
- A "module interface unit" is a module unit whose module-declaration starts with export-keyword; any other module unit is a "module implementation unit".
- A "module partition" is a module unit whose module-declaration contains module-partition.

http://eel.is/c++draft/module#def:module_partition

3 A *module partition* is a module unit whose *module-declaration* contains a *module-partition*. A named module shall not contain multiple module partitions with the same *module-partition*. All module partitions of a module that are module interface units shall be directly or indirectly exported by the primary module interface unit ([module.import]). No diagnostic is required for a violation of these rules.

[*Note 1*: Module partitions can be imported only by other module units in the same module. The division of a module into module units is not visible outside the module. — *end note*]

4 [*Example 1*:

Translation unit #1:

```
export module A;
export import :Foo;
export int baz();
```

Translation unit #2:

```
export module A:Foo;
import :Internals;
export int foo() { return 2 * (bar() + 1); }
```

Translation unit #3:

```
module A:Internals;
int bar();
```

Translation unit #4:

```
module A;
import :Internals;
int bar() { return baz() - 10; }
int baz() { return 30; }
```

Module `A` contains four translation units:

(4.1) — a primary module interface unit,

(4.2) — a module partition `A:Foo`, which is a module interface unit forming part of the interface of module `A`,

(4.3) — a module partition `A:Internals`, which does not contribute to the external interface of module `A`, and

(4.4) — a module implementation unit providing a definition of `bar` and `baz`, which cannot be imported because it does not have a partition name.

— *end example*]

kitware

STD standard example

```
add_library(std_module_example)
target_link_libraries(std_module_example t3lib)

target_sources(std_module_example
    PUBLIC
     FILE_SET cxx_modules TYPE CXX_MODULES FILES
     t1.cxx t2.cxx t3.cxx t4.cxx
)

add_executable(main main.cxx)
target_link_libraries(main std_module_example)
```

ninja build g++

```
$[6/14] Generating CXX dyndep file CMakeFiles/std_module_example.dir/CXX.dd
FAILED: CMakeFiles/std_module_example.dir/CXX.dd
"/Users/hoffman/Work/My Builds/cmake-ninja/bin/cmake" -E cmake_ninja_dyndep
--tdi=CMakeFiles/std_module_example.dir/CXXDependInfo.json --lang=CXX --modmapfmt=gcc
--dd=CMakeFiles/std_module_example.dir/CXX.dd @CMakeFiles/std_module_example.dir/CXX.dd.rsp
CMake Error: Output CMakeFiles/std_module_example.dir/t4.cxx.o is of type `CXX_MODULES` but does not provide
a module
```

STD standard example

```
add_library(std_module_example)
target_link_libraries(std_module_example t3lib)

target_sources(std_module_example
  PRIVATE
     t4.cxx  # does not export any C++ modules
  PUBLIC
     FILE_SET cxx_modules TYPE CXX_MODULES FILES
     t1.cxx t2.cxx t3.cxx
)

add_executable(main main.cxx)
target_link_libraries(main std_module_example)
```

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
- Bot 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.

kitware

ninja build g++

```
$ninja -v
[1/14] /home/bill/gcc-install/bin/c++   -std=gnu++20 -E -x c++ /home/bill/cxxm/std_example/t4.cxx -MT CMakeFiles/std_module_example.dir/t4.cxx.o.ddi -MD -MF
CMakeFiles/std_module_example.dir/t4.cxx.o.ddi.d -fmodules-ts -fdep-file=CMakeFiles/std_module_example.dir/t4.cxx.o.ddi -fdep-output=CMakeFiles/std_module_example.dir/t4.cxx.o -fdep-format=trtbd -o
CMakeFiles/std_module_example.dir/t4.cxx.o.ddi.i
[2/14] /home/bill/gcc-install/bin/c++   -std=gnu++20 -E -x c++ /home/bill/cxxm/std_example/t1.cxx -MT CMakeFiles/std_module_example.dir/t1.cxx.o.ddi -MD -MF
CMakeFiles/std_module_example.dir/t1.cxx.o.ddi.d -fmodules-ts -fdep-file=CMakeFiles/std_module_example.dir/t1.cxx.o.ddi -fdep-output=CMakeFiles/std_module_example.dir/t1.cxx.o -fdep-format=trtbd -o
CMakeFiles/std_module_example.dir/t1.cxx.o.ddi.i
[3/14] /home/bill/gcc-install/bin/c++   -std=gnu++20 -E -x c++ /home/bill/cxxm/std_example/t2.cxx -MT CMakeFiles/std_module_example.dir/t2.cxx.o.ddi -MD -MF
CMakeFiles/std_module_example.dir/t2.cxx.o.ddi.d -fmodules-ts -fdep-file=CMakeFiles/std_module_example.dir/t2.cxx.o.ddi -fdep-output=CMakeFiles/std_module_example.dir/t2.cxx.o -fdep-format=trtbd -o
CMakeFiles/std_module_example.dir/t2.cxx.o.ddi.i
[4/14] /home/bill/gcc-install/bin/c++   -std=gnu++20 -E -x c++ /home/bill/cxxm/std_example/main.cxx -MT CMakeFiles/main.dir/main.cxx.o.ddi -MD -MF CMakeFiles/main.dir/main.cxx.o.ddi.d -fmodules-ts
-fdep-file=CMakeFiles/main.dir/main.cxx.o.ddi -fdep-output=CMakeFiles/main.dir/main.cxx.o -fdep-format=trtbd -o CMakeFiles/main.dir/main.cxx.o.ddi.i
[5/14] /home/bill/gcc-install/bin/c++   -std=gnu++20 -E -x c++ /home/bill/cxxm/std_example/t3.cxx -MT CMakeFiles/std_module_example.dir/t3.cxx.o.ddi -MD -MF
CMakeFiles/std_module_example.dir/t3.cxx.o.ddi.d -fmodules-ts -fdep-file=CMakeFiles/std_module_example.dir/t3.cxx.o.ddi -fdep-output=CMakeFiles/std_module_example.dir/t3.cxx.o -fdep-format=trtbd -o
CMakeFiles/std_module_example.dir/t3.cxx.o.ddi.i
[6/14] /home/bill/cxxm/bencmake/ninja/bin/cmake -E cmake_ninja_dyndep --tdi=CMakeFiles/std_module_example.dir/CXXDependInfo.json --lang=CXX --modmapfmt=gcc
--dd=CMakeFiles/std_module_example.dir/CXX.dd @CMakeFiles/std_module_example.dir/CXX.dd.rsp
[7/14] /home/bill/gcc-install/bin/c++   -std=gnu++20 -MD -MT CMakeFiles/std_module_example.dir/t3.cxx.o -MF CMakeFiles/std_module_example.dir/t3.cxx.o.d -fmodules-ts
-fmodule-mapper=CMakeFiles/std_module_example.dir/t3.cxx.o.modmap -fdep-format=trtbd -x c++ -o CMakeFiles/std_module_example.dir/t3.cxx.o -c /home/bill/cxxm/std_example/t3.cxx
[8/14] /home/bill/gcc-install/bin/c++   -std=gnu++20 -MD -MT CMakeFiles/std_module_example.dir/t2.cxx.o -MF CMakeFiles/std_module_example.dir/t2.cxx.o.d -fmodules-ts
-fmodule-mapper=CMakeFiles/std_module_example.dir/t2.cxx.o.modmap -fdep-format=trtbd -x c++ -o CMakeFiles/std_module_example.dir/t2.cxx.o -c /home/bill/cxxm/std_example/t2.cxx
[9/14] /home/bill/gcc-install/bin/c++   -std=gnu++20 -MD -MT CMakeFiles/std_module_example.dir/t1.cxx.o -MF CMakeFiles/std_module_example.dir/t1.cxx.o.d -fmodules-ts
-fmodule-mapper=CMakeFiles/std_module_example.dir/t1.cxx.o.modmap -fdep-format=trtbd -x c++ -o CMakeFiles/std_module_example.dir/t1.cxx.o -c /home/bill/cxxm/std_example/t1.cxx
[10/14] /home/bill/gcc-install/bin/c++   -std=gnu++20 -MD -MT CMakeFiles/std_module_example.dir/t4.cxx.o -MF CMakeFiles/std_module_example.dir/t4.cxx.o.d -fmodules-ts
-fmodule-mapper=CMakeFiles/std_module_example.dir/t4.cxx.o.modmap -fdep-format=trtbd -x c++ -o CMakeFiles/std_module_example.dir/t4.cxx.o -c /home/bill/cxxm/std_example/t4.cxx
[11/14] : && /home/bill/cxxm/bencmake/ninja/bin/cmake -E rm -f libstd_module_example.a && /usr/bin/ar qc libstd_module_example.a  CMakeFiles/std_module_example.dir/t4.cxx.o
CMakeFiles/std_module_example.dir/t1.cxx.o CMakeFiles/std_module_example.dir/t2.cxx.o CMakeFiles/std_module_example.dir/t3.cxx.o && /usr/bin/ranlib libstd_module_example.a && :
[12/14] /home/bill/cxxm/bencmake/ninja/bin/cmake -E cmake_ninja_dyndep --tdi=CMakeFiles/main.dir/CXXDependInfo.json --lang=CXX --modmapfmt=gcc --dd=CMakeFiles/main.dir/CXX.dd
@CMakeFiles/main.dir/CXX.dd.rsp
[13/14] /home/bill/gcc-install/bin/c++   -std=gnu++20 -MD -MT CMakeFiles/main.dir/main.cxx.o -MF CMakeFiles/main.dir/main.cxx.o.d -fmodules-ts -fmodule-mapper=CMakeFiles/main.dir/main.cxx.o.modmap
-fdep-format=trtbd -x c++ -o CMakeFiles/main.dir/main.cxx.o -c /home/bill/cxxm/std_example/main.cxx
[14/14] : && /home/bill/gcc-install/bin/c++   CMakeFiles/main.dir/main.cxx.o -o main  libstd_module_example.a && :
```

kitware

ninja build (MSVC)

```
$ ninja
[1/14] Scanning C:\Users\hoffman\Work\cxxmodules\cxx_modules\std_example\t2.cxx for CXX dependencies
[2/14] Scanning C:\Users\hoffman\Work\cxxmodules\cxx_modules\std_example\t4.cxx for CXX dependencies
[3/14] Scanning C:\Users\hoffman\Work\cxxmodules\cxx_modules\std_example\t3.cxx for CXX dependencies
[4/14] Scanning C:\Users\hoffman\Work\cxxmodules\cxx_modules\std_example\t1.cxx for CXX dependencies
[5/14] Scanning C:\Users\hoffman\Work\cxxmodules\cxx_modules\std_example\main.cxx for CXX dependencies
[6/14] Generating CXX dyndep file CMakeFiles\std_module_example.dir\CXX.dd
warning: acknowledging an VS 17.3 toolchain bug; accepting until a new release which fixes it is available
warning: acknowledging an VS 17.3 toolchain bug; accepting until a new release which fixes it is available
warning: acknowledging an VS 17.3 toolchain bug; accepting until a new release which fixes it is available
warning: acknowledging an VS 17.3 toolchain bug; accepting until a new release which fixes it is available
[7/14] Building CXX object CMakeFiles\std_module_example.dir\t3.cxx.obj
[8/14] Building CXX object CMakeFiles\std_module_example.dir\t2.cxx.obj
[9/14] Building CXX object CMakeFiles\std_module_example.dir\t1.cxx.obj
[10/14] Building CXX object CMakeFiles\std_module_example.dir\t4.cxx.obj
[11/14] Linking CXX static library std_module_example.lib
[12/14] Generating CXX dyndep file CMakeFiles\main.dir\CXX.dd
warning: acknowledging an VS 17.3 toolchain bug; accepting until a new release which fixes it is available
[13/14] Building CXX object CMakeFiles\main.dir\main.cxx.obj
[14/14] Linking CXX executable main.exe
```

Split into two libraries

```
cmake_minimum_required(VERSION 3.23)
project(std_module_example CXX)
set(CMAKE_CXX_STANDARD 20)

add_library(t3lib)
target_sources(t3lib
  PUBLIC
     FILE_SET cxx_modules_internals
     TYPE CXX_MODULE FILES
       t3.cxx
)
```

```
add_library(std_module_example)
target_link_libraries(std_module_example t3lib)

target_sources(std_module_example
  PRIVATE
     t4.cxx
  PUBLIC
     FILE_SET cxx_modules TYPE CXX_MODULES FILES
     t1.cxx t2.cxx
)
add_executable(main main.cxx)
target_link_libraries(main std_module_example)
```

kitware

run ninja module split across libraries

```
$ ninja
[1/16] Scanning C:\Users\hoffman\Work\cxxmodules\std_example\t4.cxx for CXX dependencies
[2/16] Scanning C:\Users\hoffman\Work\cxxmodules\std_example\t3.cxx for CXX dependencies
[3/16] Scanning C:\Users\hoffman\Work\cxxmodules\std_example\t1.cxx for CXX dependencies
[4/16] Scanning C:\Users\hoffman\Work\cxxmodules\std_example\t2.cxx for CXX dependencies
[5/16] Scanning C:\Users\hoffman\Work\cxxmodules\std_example\main.cxx for CXX dependencies
[6/16] Generating CXX dyndep file CMakeFiles\t3lib.dir\CXX.dd
[7/16] Building CXX object CMakeFiles\t3lib.dir\t3.cxx.obj
[8/16] Linking CXX static library t3lib.lib
[9/16] Generating CXX dyndep file CMakeFiles\std_module_example.dir\CXX.dd
[10/16] Building CXX object CMakeFiles\std_module_example.dir\t2.cxx.obj
[11/16] Building CXX object CMakeFiles\std_module_example.dir\t1.cxx.obj
[12/16] Building CXX object CMakeFiles\std_module_example.dir\t4.cxx.obj
[13/16] Linking CXX static library std_module_example.lib
[14/16] Generating CXX dyndep file CMakeFiles\main.dir\CXX.dd
[15/16] Building CXX object CMakeFiles\main.dir\main.cxx.obj
[16/16] Linking CXX executable main.exe
```

cxx-modules-examples

- `main.cpp`: Executable consuming the module(s).
- `mymodule.cpp`: A module interface unit.
- `mymodule_impl.cpp`: A module implementation unit.
- `mymodule_part.cpp`: A module partition (re-exported by `mymodule.cpp`).
- `mymodule_part_impl.cpp`: A module implementation unit, implementing the partition.
- `mymodule_part_internal.cpp`: A module partition (internal).

main.cpp

```
// Module consumer.
import MyModule;

// Use functions exported by the module interface unit.
int main() {
  mod_f();
  mod_g();
  part_f();
  part_g();
  part_g_impl();
  return 0;
}
```

mymodule.cpp

```
// Module Interface Unit
export module MyModule;

// Import and re-export a module partition.
export import :part;

// Import an internal module partition.
import :part_internal;

// Export a function from this module.  Implement it here.
export void mod_f()
{
  // The implementation can use internal module partition
  // functions even though they are not exported.
  part_internal_f();
}

// Export a function from this module.  Implemented elsewhere.
export void mod_g();
```

mymodule_impl.cpp

```
// Module Implementation Unit
module MyModule;

// Implement a function declared in the module interface unit.
void mod_g() {}

// Implement a function declared in a module partition.
void part_g() {}
```

mymodule_part.cpp

```
// Module Partition
export module MyModule:part;

// Export a function from this module partition.  Implement it here.
export void part_f() {}

// Export functions from this module partition.  Implemented elsewhere.
export void part_g();
export void part_g_impl();
```

mymodule_part_impl.cpp

```
// Module Implementation Unit
// Implements functions from the module interface or module partition.
module MyModule;
// Implement a function declared in a module partition.
void part_g_impl() {}
```

mymodule_part_internal.cpp

```
// Module Partition (internal)
module MyModule:part_internal;

// Define a function usable elsewhere in the module implementation.
// This is not exported from the module.
void part_internal_f() {}
```

cmake code

```
add_library(with_named_modules)
target_sources(with_named_modules
    PRIVATE
        mymodule_impl.cpp
        mymodule_part_impl.cpp
        main.cpp
    PUBLIC
        FILE_SET cxx_modules TYPE CXX_MODULES FILES
            mymodule_part.cpp
            mymodule.cpp
            mymodule_part_internal.cpp)
```

install and export cmake code

```
install(TARGETS with_named_modules
    EXPORT with_named_modules
    RUNTIME
        DESTINATION "bin"
        COMPONENT "runtime"
    ARCHIVE
        DESTINATION "lib"
        COMPONENT "development"
    LIBRARY
        DESTINATION "lib"
        COMPONENT "runtime"
        NAMELINK_COMPONENT "development"
    FILE_SET cxx_modules
        DESTINATION "include/cxx/cxx-modules-examples"
        COMPONENT "development"
    FILE_SET cxx_modules_internals
        DESTINATION "include/cxx/cxx-modules-examples"
        COMPONENT "development"
    CXX_MODULES_BMI
        DESTINATION
"lib/${CMAKE_CXX_COMPILER_ID}-${CMAKE_CXX_COMPILER_VERSION}/$<CON
FIG>"
        COMPONENT "development")
```

```
export(
    EXPORT with_named_modules
    NAMESPACE NS::
    FILE
"${CMAKE_BINARY_DIR}/lib/cmake/cxx-modules-examples/with_named_modules-tar
gets.cmake"
    CXX_MODULES_DIRECTORY "with_named_modules-cxx-modules")
install(
    EXPORT with_named_modules
    NAMESPACE NS::
    DESTINATION "lib/cmake/cxx-modules-examples"
    FILE "with_named_modules-targets.cmake"
    CXX_MODULES_DIRECTORY "with_named_modules-cxx-modules"
    COMPONENT "development")
```

running install

```
$ cmake  --install-prefix=$HOME/cxxm/cxxmodexamp/inst .
$ ninja install
[0/1] Install the project...
-- Install configuration: ""
-- Installing: /home/bill/cxxm/cxxmodexamp/inst/lib/libwith_named_modules.a
-- Up-to-date: /home/bill/cxxm/cxxmodexamp/inst/include/cxx/cxx-modules-examples/mymodule_part.cpp
-- Up-to-date: /home/bill/cxxm/cxxmodexamp/inst/include/cxx/cxx-modules-examples/mymodule.cpp
-- Up-to-date: /home/bill/cxxm/cxxmodexamp/inst/include/cxx/cxx-modules-examples/mymodule_part_internal.cpp
-- Up-to-date: /home/bill/cxxm/cxxmodexamp/inst/lib/cmake/cxx-modules-examples/with_named_modules-targets.cmake
-- Up-to-date: /home/bill/cxxm/cxxmodexamp/inst/lib/cmake/cxx-modules-examples/with_named_modules-targets-noconfig.cmake
-- Up-to-date:
/home/bill/cxxm/cxxmodexamp/inst/lib/cmake/cxx-modules-examples/with_named_modules-cxx-modules/cxx-modules.cmake
-- Up-to-date:
/home/bill/cxxm/cxxmodexamp/inst/lib/cmake/cxx-modules-examples/with_named_modules-cxx-modules/cxx-modules-noconfig.cmak
e
-- Installing:
/home/bill/cxxm/cxxmodexamp/inst/lib/cmake/cxx-modules-examples/with_named_modules-cxx-modules/target-with_named_modules
-noconfig.cmake
-- Installing: /home/bill/cxxm/cxxmodexamp/inst/lib/GNU-12.0.1//MyModule-part.gcm
-- Installing: /home/bill/cxxm/cxxmodexamp/inst/lib/GNU-12.0.1//MyModule.gcm
-- Installing: /home/bill/cxxm/cxxmodexamp/inst/lib/GNU-12.0.1//MyModule-part_internal.gcm
```

```
$ ninja install
[0/1] Install the project...
-- Install configuration: "Debug"
-- Installing: C:/Users/hoffman/Work/cxxmodules/cxx-modules-examples/inst/lib/with_named_modules.lib
-- Installing: C:/Users/hoffman/Work/cxxmodules/cxx-modules-examples/inst/include/cxx/cxx-modules-examples/mymodule_part.cpp
-- Installing: C:/Users/hoffman/Work/cxxmodules/cxx-modules-examples/inst/include/cxx/cxx-modules-examples/mymodule.cpp
-- Installing: C:/Users/hoffman/Work/cxxmodules/cxx-modules-examples/inst/include/cxx/cxx-modules-examples/mymodule_part_internal.cpp
-- Installing:
C:/Users/hoffman/Work/cxxmodules/cxx-modules-examples/inst/lib/cmake/cxx-modules-examples/with_named_modules-targets.cmake
-- Installing:
C:/Users/hoffman/Work/cxxmodules/cxx-modules-examples/inst/lib/cmake/cxx-modules-examples/with_named_modules-targets-debug.cmake
-- Installing:
C:/Users/hoffman/Work/cxxmodules/cxx-modules-examples/inst/lib/cmake/cxx-modules-examples/with_named_modules-cxx-modules/cxx-modules.
cmake
-- Installing:
C:/Users/hoffman/Work/cxxmodules/cxx-modules-examples/inst/lib/cmake/cxx-modules-examples/with_named_modules-cxx-modules/cxx-modules-
Debug.cmake
-- Installing:
C:/Users/hoffman/Work/cxxmodules/cxx-modules-examples/inst/lib/cmake/cxx-modules-examples/with_named_modules-cxx-modules/target-with_
named_modules-Debug.cmake
-- Installing: C:/Users/hoffman/Work/cxxmodules/cxx-modules-examples/inst/lib/MSVC-19.32.31326.0/Debug/MyModule-part.ifc
-- Installing: C:/Users/hoffman/Work/cxxmodules/cxx-modules-examples/inst/lib/MSVC-19.32.31326.0/Debug/MyModule.ifc
-- Installing: C:/Users/hoffman/Work/cxxmodules/cxx-modules-examples/inst/lib/MSVC-19.32.31326.0/Debug/MyModule-part_internal.ifc
```

Kick the tires CMake C++20 Modules

kitware

CMake C++20 Modules

- Build CMake master
  - Read the docs: https://github.com/Kitware/CMake/blob/master/Help/dev/experimental.rst
- Get MSVC 2022 preview
- Build 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")
if(WIN32)
  include(msvc_modules.cmake)
else()
  include(gcc_modules.cmake)
endif()
set(CMAKE_CXX_STANDARD 20)
```

kitware

gcc on M1 mac

https://github.com/iains/gcc-darwin-arm64

patch -p1 <
https://github.com/gcc-mirror/gcc/commit/3075e510e3d29583f8886b95aff044c0474c84a5.patch

build gcc, set CXX and CC env vars before running CMake.

CMake C++ 20 Modules gcc

```
SET (CMAKE_EXPERIMENTAL_CXX_MODULE_CMAKE_API 3c375311-a3c9-4396-a187-3227ef642046)



set(CMAKE_EXPERIMENTAL_CXX_MODULE_DYNDEP 1)
string(CONCAT CMAKE_EXPERIMENTAL_CXX_SCANDEP_SOURCE
  "<CMAKE_CXX_COMPILER> <DEFINES> <INCLUDES> <FLAGS> <SOURCE>"
  " -MT <DYNDEP_FILE> -MD -MF <DEP_FILE>"
  " ${flags_to_scan_deps} -fdep-file=<DYNDEP_FILE> -fdep-output=<OBJECT>"
  )



set(CMAKE_EXPERIMENTAL_CXX_MODULE_MAP_FORMAT "gcc")
set(CMAKE_EXPERIMENTAL_CXX_MODULE_MAP_FLAG
  "${compiler_flags_for_module_map} -fmodule-mapper=<MODULE_MAP_FILE>")
```

kitware

CMake C++ 20 Modules msvc

```
SET (CMAKE_EXPERIMENTAL_CXX_MODULE_CMAKE_API 3c375311-a3c9-4396-a187-3227ef642046)



set(CMAKE_EXPERIMENTAL_CXX_MODULE_DYNDEP 1)
string(CONCAT CMAKE_EXPERIMENTAL_CXX_SCANDEP_SOURCE
  "<CMAKE_CXX_COMPILER> <DEFINES> <INCLUDES> <FLAGS> <SOURCE> -nologo -TP"
  " -showIncludes"
  " -scanDependencies <DYNDEP_FILE>"
  " -Fo<OBJECT>"
  )


set(CMAKE_EXPERIMENTAL_CXX_SCANDEP_DEPFILE_FORMAT "msvc")
set(CMAKE_EXPERIMENTAL_CXX_MODULE_MAP_FORMAT "msvc")
set(CMAKE_EXPERIMENTAL_CXX_MODULE_MAP_FLAG "@<MODULE_MAP_FILE>")
```

kitware

Simple Example

https://gitlab.kitware.com/bill-hoffman/cxx_modules

CMake C++20 Module tests

Add this to msvc or gcc rule file:

```
set(CMake_TEST_CXXModules_UUID "a246741c-d067-4019-a8fb-3d16b0c9d1d3")
 /path/to/built/cmake \
   -DCMake_TEST_MODULE_COMPILATION=named,shared,partitions,internal_partitions,export_bmi,install_bmi \
   -DCMake_TEST_MODULE_COMPILATION_RULES=/full/path/to/rules/file.cmake \  # MSVC or gcc
   -DCMake_TEST_HOST_CMAKE=ON \
   -S /path/to/cmake/source/tree \
   -B /path/to/build/tree
```

The test suite is available as `RunCMake.CXXModules` (pass to `ctest -R`to run) and the sources live in `Tests/RunCMake/CXXModules`; the `RunCMakeTest.cmake` script is where to start looking for what's going on in the test itself.

Tests/RunCMake/CXXModules/examples has some examples to look at

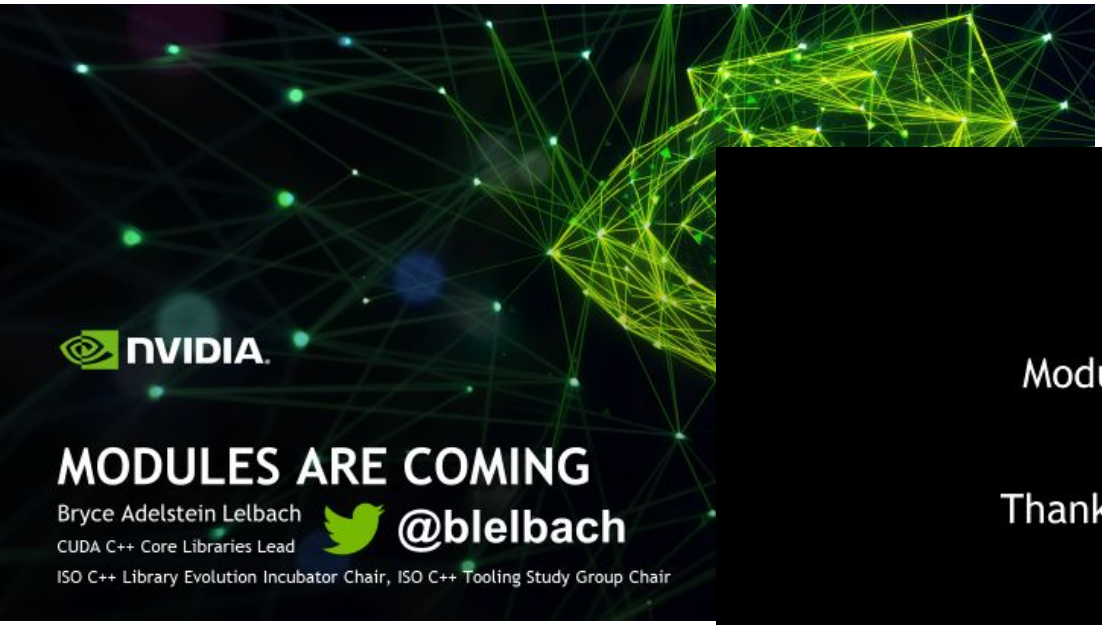
kitware

CMake C++Modules status

- Experimental code in CMake master
- MSVC modules with 2022 preview
- GCC working with patch
- Clang not does not yet provide p1689 depend scanning
  https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2021/p1689r4.html
- Header units stubbed out but not yet working due to lack of compiler support

kitware

Modules almost here?



kitware

Bryce at Meeting C++ 2019

Other New CMake Features

Some command line improvements

```
cmake --install-prefix path
over
cmake -DCMAKE_INSTALL_PREFIX=path
```

```
cmake --toolchain $file
over
cmake -DCMAKE_TOOLCHAIN_FILE=path
```

Debugging Find Calls [3.23+]

```
cmake --debug-find-pkg=XYZ ../src
```

```
Running with debug output on for the 'find' commands for package(s) XYZ.
CMake Debug Log at CMakeLists.txt:4 (find_package):
  find_package considered the following paths for FindXYZ.cmake:
    ...
```

Debugging Find Calls [3.23+]

```
cmake --debug-find-var=mylib ../src
```

```
Running with debug output on for the variable(s) mylib.
  find_library called with the following settings:
      ...
  find_library considered the following locations:
      ...
  The item was not found.
```


kitware

DEPFILE in all generators

Th DEPFILE

New in version 3.7.

Specify a depfile which holds dependencies for the custom command. It is usually emitted by the custom command itself. This keyword may only be used if the generator supports it, as detailed below.

The expected format, compatible with what is generated by `gcc` with the option `-M`, is independent of the generator or platform.

The formal syntax, as specified using BNF notation with the regular extensions, is the following:

```
depfile        ::=  rule*
rule           ::=  targets (`:` (separator dependencies?)?)? eol
targets        ::=  target (separator target)* separator*
target         ::=  pathname
dependencies   ::=  dependency (separator dependency)* separator*
dependency     ::=  pathname
separator      ::=  (space | line_continue)+
line_continue  ::=  '\' eol
space          ::=  ' ' | '\t'
pathname       ::=  character+
character      ::=  std_character | dollar | hash | whitespace
std_character  ::=  <any character except '$', '#' or ' '>
dollar         ::=  '$$'
hash           ::=  '\#'
whitespace     ::=  '\ '
eol            ::=  '\r'? '\n'
```

New in version 3.21.

```
install(RUNTIME_DEPENDENCY_SET <set-name>
        [[LIBRARY|RUNTIME|FRAMEWORK]
         [DESTINATION <dir>]
         [PERMISSIONS permissions...]
         [CONFIGURATIONS [Debug|Release|...]]
         [COMPONENT <component>]
         [NAMELINK_COMPONENT <component>]
         [OPTIONAL] [EXCLUDE_FROM_ALL]
        ] [...]
        [PRE_INCLUDE_REGEXES regexes...]
        [PRE_EXCLUDE_REGEXES regexes...]
        [POST_INCLUDE_REGEXES regexes...]
        [POST_EXCLUDE_REGEXES regexes...]
        [POST_INCLUDE_FILES files...]
        [POST_EXCLUDE_FILES files...]
        [DIRECTORIES directories...]
        )
```

Installs a runtime dependency set previously created by one or more install(TARGETS) or install(IMPORTED_RUNTIME_ARTIFACTS) commands. The dependencies of targets belonging to a runtime dependency set are installed in the `RUNTIME` destination and component on DLL platforms, and in the `LIBRARY` destination and component on non-DLL platforms. macOS frameworks are installed in the `FRAMEWORK` destination and component. Targets built within the build tree will never be installed as runtime dependencies, nor will their own dependencies, unless the targets themselves are installed with install(TARGETS).

The generated install script calls **`file(GET_RUNTIME_DEPENDENCIES)`** on the build-tree files to calculate the runtime dependencies. The build-tree executable files are passed as the `EXECUTABLES` argument, the build-tree shared libraries as the `LIBRARIES` argument, and the build-tree modules as the `MODULES` argument. On macOS, if one of the executables is a **`MACOSX_BUNDLE`**, that executable is passed as the `BUNDLE_EXECUTABLE` argument. At most one such bundle executable may be in the runtime dependency set on macOS. The **`MACOSX_BUNDLE`** property has no effect on other platforms. Note that **`file(GET_RUNTIME_DEPENDENCIES)`** only supports collecting the runtime dependencies for Windows, Linux and macOS platforms, so `install(RUNTIME_DEPENDENCY_SET)` has the same limitation.

The following sub-arguments are forwarded through as the corresponding arguments to **`file(GET_RUNTIME_DEPENDENCIES)`** (for those that provide a non-empty list of directories, regular expressions or files). They all support **`generator expressions`**.

Few more CMake Features

* CMAKE_BUILD_TYPE and CMAKE_CONFIGURATION_TYPES environment variables to set defaults for the corresponding cache entries.

* Support for SYSTEM include directories with MSVC -external:I

* CTest gained an ENVIRONMENT_MODIFICATION test property to modify the environment for individual tests.

* HIP language support (though there are some open portability issues that we have no funding to address).

* CUDA language support improved in several areas (see 3.23. notes)

* VS .NET SDK-style C# project files (contributed, not yet mature).

* VS NuGet package restoration with cmake --build

CMake Guides

cmake.org/cmake/help/latest/index.html

- cmake-toolchains(7)
- cmake-variables(7)
- cpack-generators(7)

Guides

- CMake Tutorial
- User Interaction Guide
- Using Dependencies Guide
- Importing and Exporting Guide
- IDE Integration Guide

kitware

Testing Doc Code

```
Specify the C++ Standard
------------------------

Next let's add some C++11 features to our project by replacing ``atof`` with
``std::stod`` in ``tutorial.cxx``.  At the same time, remove
``#include <cstdlib>``.

.. literalinclude:: Step2/tutorial.cxx
  :language: c++
  :start-after: // convert input to double
  :end-before: // calculate square root

We will need to explicitly state in the CMake
correct flags. The easiest way to enable suppo
in CMake is by using the :variable:`CMAKE_CXX_
tutorial, set the :variable:`CMAKE_CXX_STANDAR
``CMakeLists.txt`` file to 11 and :variable:`C
True. Make sure to add the ``CMAKE_CXX_STANDAR
to ``add_executable``.

.. literalinclude:: Step2/CMakeLists.txt
  :language: cmake
  :end-before: # configure a header file to pa
```

```
#include <string>

#include "TutorialConfig.h"

int main(int argc, char* argv[])
{
  if (argc < 2) {
    // report version
    std::cout << argv[0] << " Version " << Tutorial_VERSION_MAJOR << "."
              << Tutorial_VERSION_MINOR << std::endl;
    std::cout << "Usage: " << argv[0] << " number" << std::endl;
    return 1;
  }

  // convert input to double
  const double inputValue = std::stod(argv[1]);

  // calculate square root
  const double outputValue = sqrt(inputValue);
  std::cout << "The square root of " << inputValue << " is " << outputValue
            << std::endl;
  return 0;
}
```

Specify the C++ Standard

Next let's add some C++11 features to our project by replacing `atof` with `std::stod` in `tutorial.cxx`. At the same time, remove `#include <cstdlib>`.

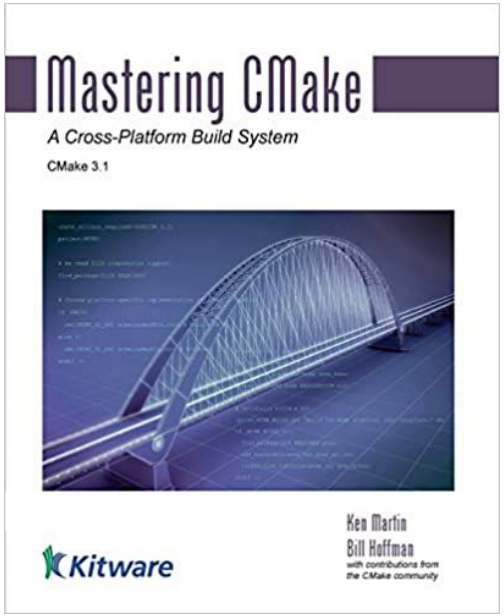
```
const double inputValue = std::stod(argv[1]);
```

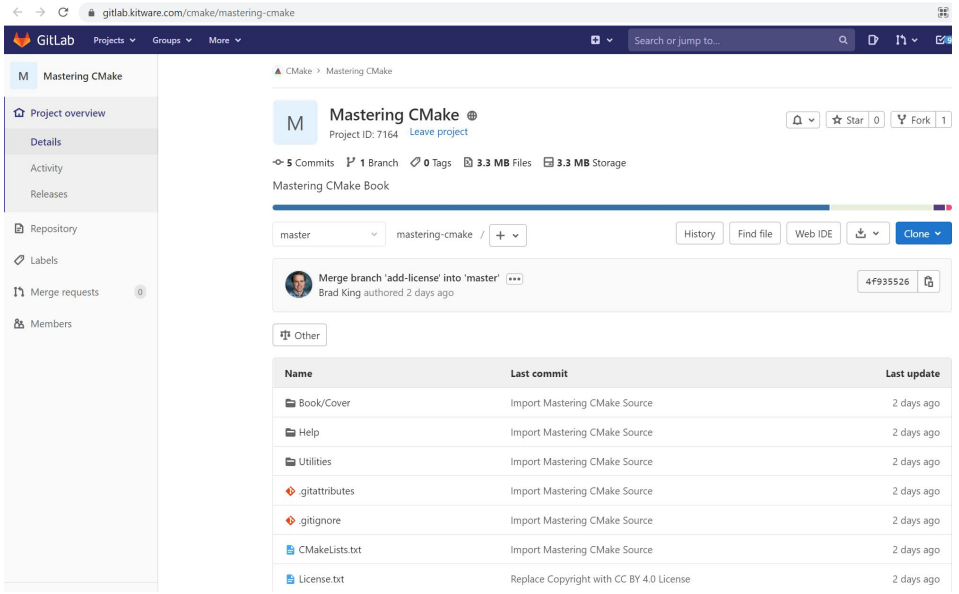
We will need to explicitly state in the CMake code that it should use the correct flags. The easiest way to enable support for a specific C++ standard in CMake is by using the `CMAKE_CXX_STANDARD` variable. For this tutorial, set the `CMAKE_CXX_STANDARD` variable in the `CMakeLists.txt` file to 11 and `CMAKE_CXX_STANDARD_REQUIRED` to True. Make sure to add the `CMAKE_CXX_STANDARD` declarations above the call to `add_executable`.

```
cmake_minimum_required(VERSION 3.10)

# set the project name and version
project(Tutorial VERSION 1.0)

# specify the C++ standard
set(CMAKE_CXX_STANDARD 11)
set(CMAKE_CXX_STANDARD_REQUIRED True)
```

kitware

Mastering CMake Now Open Source!

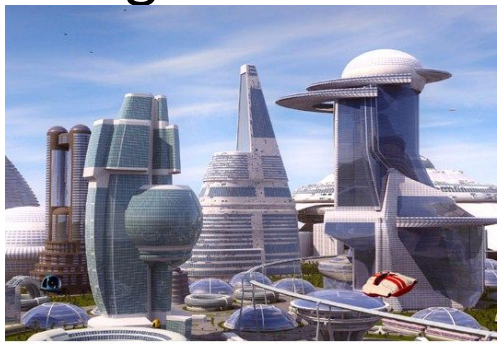




kitware

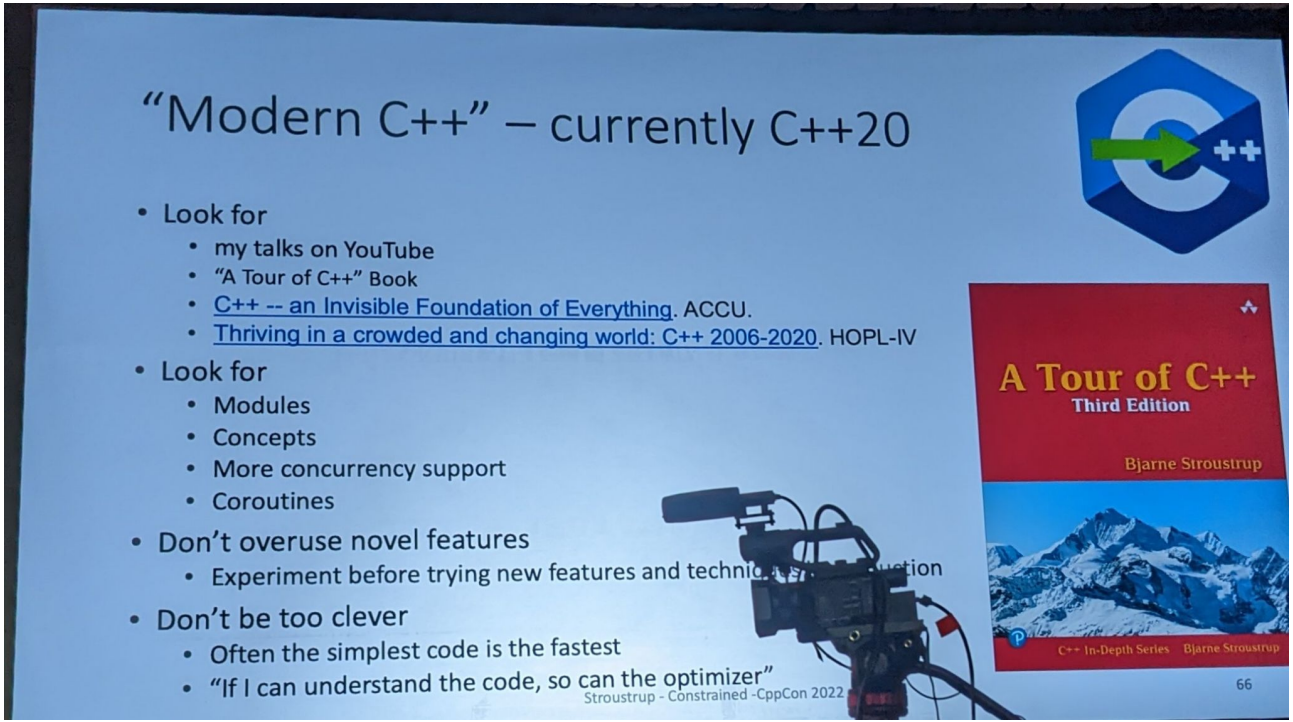
Future I would like to see for C++/CMake

- All C++ compilers provide build system interfaces to collect C++20 module dependency information.
- A cross platform standard for the information found in cmake config files



kitware

We all want Modules and package managers



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