TACC

Help! I want to write CMake myself!

Victor Eijkhout

Fall 2023

Justification

CMake is a portable build system that is becoming a *de facto* standard for C++ package management.
If you publish software, it's becoming expected that you deliver a CMake configuration.

Table of contents

Make your CMake configuration

What are we talking here?

TACC

You have a code that you want to distribute in source form for easy installation.
You decide to use CMake for portability.
You think that using CMake might make life easier.
$\Rightarrow$ To do: write the `CMakeLists.txt` file.

The CMakeLists file

```
cmake_minimum_required( VERSION 3.12 )
project( myproject VERSION 1.0 )
```

- Which cmake version is needed for this file?
  (CMake has undergone quite some evolution!)
- Give a name to your project.
- Maybe pick a language.
  C and C++ available by default, or:

```
enable_language(Fortran)
```


Target philosophy

- Declare a target: something that needs to be built, and specify what is needed for it

  ```
  add_executable( myprogram program.cxx )
  ```

  Use of macros:

  ```
  add_executable( ${PROJECT_NAME} program.cxx )
  ```

- Do things with the target, for instance state where it is to be installed:

  ```
  install( TARGETS myprogram DESTINATION . )
  ```

  relative to the prefix location.

Example: single source

Build an executable from a single source file:

```
cmake_minimum_required( VERSION 3.12 )
project( singleprogram VERSION 1.0 )

add_executable( program program.cxx )
install( TARGETS program DESTINATION . )
```


Use of a library

First a library that goes into the executable:

```
add_library( auxlib aux.cxx aux.h )
target_link_libraries( program PRIVATE auxlib )
```

Example: library during build

Full configuration for an executable that uses a library:

```
cmake_minimum_required( VERSION 3.12 )
project( cmakeprogram VERSION 1.0 )

add_executable( program program.cxx )
add_library( auxlib
             aux.cxx aux.h )
target_link_libraries( program PRIVATE auxlib )
install( TARGETS program DESTINATION . )
```

Library shared by default; see later.

Shared and static libraries

In the configuration file:

```
add_library( auxlib STATIC aux.cxx aux.h )
# or
add_library( auxlib SHARED aux.cxx aux.h )
```

(default shared if left out), or by adding a runtime flag

```
cmake -D BUILD_SHARED_LIBS=TRUE
```

Build both by having two lines, one for shared, one for static.
Related: the `-fPIC` compile option is set by `CMAKE_POSITION_INDEPENDENT_CODE`:

```
cmake -D CMAKE_POSITION_INDEPENDENT_CODE=ON
```

Release a library

To have the library released too, use **`PUBLIC`**.
Add the library target to the **`install`** command.

Example: released library

```
cmake_minimum_required( VERSION 3.12 )
project( cmakeprogram VERSION 1.0 )

add_executable( program program.cxx )
target_include_directories(
       program PRIVATE lib )
add_library(
       auxlib STATIC
       lib/aux.cxx lib/aux.h )
target_link_libraries( program PUBLIC auxlib )
install( TARGETS program DESTINATION bin )
install( TARGETS auxlib DESTINATION lib )
install( FILES lib/aux.h DESTINATION include )
```

Note the separate destination directories.

We are getting realistic

The previous setup was messy
Better handle the library through a recursive cmake
and make the usual `lib` `include` `bin` setup

Recursive setup, main directory

Declare that there is a directory to do recursive make:

```
1  cmake_minimum_required( VERSION 3.12 )
2  project( cmakeprogram VERSION 1.0 )
3
4  add_executable( program program.cxx )
5  add_subdirectory( lib )
6  target_include_directories(
7          program PRIVATE lib )
8  target_link_libraries( program PUBLIC auxlib )
9  install( TARGETS program DESTINATION bin )
```

(Note that the name of the library comes from the subdirectory)

Recursive setup, subdirectory

Installs into `lib` and `include`

```
1  cmake_minimum_required( VERSION 3.12 )
2
3  add_library(
4      auxlib STATIC
5      aux.cxx aux.h )
6  install( TARGETS auxlib DESTINATION lib )
7  install( FILES aux.h DESTINATION include )
```


External libraries

- Use `LD_LIBRARY_PATH`, or
- use `rpath`.

(Apple note: forced to use second option)

```
set_target_properties(
    ${PROGRAM_NAME} PROPERTIES
    BUILD_RPATH   "${CATCH2_LIBRARY_DIRS};${FMTLIB_LIBRARY_DIRS}"
    INSTALL_RPATH "${CATCH2_LIBRARY_DIRS};${FMTLIB_LIBRARY_DIRS}"
    )
```

Using other packages

Package finding

Package dependent:

- Sometimes through `pkg-config`:
  find the `.pc` file
- Sometimes through a *`Find`*`....` module
  see CMake documentation

Catch2

```
find_package( PkgConfig REQUIRED )
pkg_check_modules( CATCH2 REQUIRED catch2-with-main )
target_include_directories(
        ${PROGRAM_NAME} PUBLIC
        ${CATCH2_INCLUDE_DIRS}
        )
target_link_directories(
        ${PROGRAM_NAME} PUBLIC
        ${CATCH2_LIBRARY_DIRS}
        )
target_link_libraries(
        ${PROGRAM_NAME} PUBLIC
        ${CATCH2_LIBRARIES}
        )
```


Cxxopts

Header-only:

```
find_package( PkgConfig REQUIRED )
pkg_check_modules( OPTS REQUIRED cxxopts )
target_include_directories(
        ${PROGRAM_NAME} PUBLIC
        ${OPTS_INCLUDE_DIRS}
        )
```

Eigen

Header-only:

```
cmake_minimum_required( VERSION 3.12 )
project( eigentest )

find_package( PkgConfig REQUIRED )
pkg_check_modules( EIGEN REQUIRED eigen3 )

add_executable( eigentest eigentest.cxx )
target_include_directories(
        myprogram PUBLIC
        ${EIGEN_INCLUDE_DIRS})
```

Fmtlib

```
find_package( PkgConfig REQUIRED )
pkg_check_modules( FMTLIB REQUIRED fmt )
target_include_directories(
        ${PROGRAM_NAME} PUBLIC ${FMTLIB_INCLUDE_DIRS}
        )
target_link_directories(
        ${PROGRAM_NAME} PUBLIC ${FMTLIB_LIBRARY_DIRS}
        )
target_link_libraries(
        ${PROGRAM_NAME} PUBLIC ${FMTLIB_LIBRARIES}
        )
```

Range-v3

Has its own module:

```
find_package( range-v3 REQUIRED )
target_link_libraries(
    ${PROGRAM_NAME} PUBLIC range-v3::range-v3 )
```

MPI

MPI from C

MPI has a module:

```
find_package( MPI )
target_include_directories(
        ${PROJECT_NAME} PUBLIC
        ${MPI_C_INCLUDE_DIRS} )
target_link_libraries(
        ${PROJECT_NAME} PUBLIC
        ${MPI_C_LIBRARIES} )
```

MPI from C++

```
find_package( MPI )
target_include_directories(
        ${PROJECT_NAME} PUBLIC
        ${MPI_CXX_INCLUDE_DIRS} )
target_link_libraries(
        ${PROJECT_NAME} PUBLIC
        ${MPI_CXX_LIBRARIES} )
```


MPI from Fortran90

```
find_package( MPI )
target_include_directories(
        ${PROJECT_NAME} PUBLIC
        ${MPI_INCLUDE_DIRS} )
target_link_directories(
        ${PROJECT_NAME} PUBLIC
        ${MPI_LIBRARY_DIRS} )
target_link_libraries(
        ${PROJECT_NAME} PUBLIC
        ${MPI_Fortran_LIBRARIES} )
```

MPI from Fortran2008

```
if( MPI_Fortran_HAVE_F08_MODULE )
else()
  message( FATAL_ERROR "No f08 module for this MPI" )
endif()
```

MPL

```
find_package( mpl REQUIRED )
target_include_directories(
        ${PROJECT_NAME} PUBLIC
        ${CMAKE_CURRENT_SOURCE_DIR}
        mpl::mpl )
target_link_libraries(
        ${PROJECT_NAME} PUBLIC
        mpl::mpl )
```

OpenMP

OpenMP from C

```
find_package(OpenMP)
target_link_libraries(
    ${PROJECT_NAME}
    PUBLIC OpenMP::OpenMP_C )
```


OpenMP from C++

```
find_package(OpenMP)
if(OpenMP_CXX_FOUND)
else()
        message( FATAL_ERROR "Could not find OpenMP" )
endif()
target_link_libraries(
    ${PROJECT_NAME}
    PUBLIC OpenMP::OpenMP_CXX )
```

OpenMP from Fortran

```
enable_language(Fortran)
find_package(OpenMP)
target_link_libraries(
    ${PROJECT_NAME}
    PUBLIC OpenMP::OpenMP_Fortran )
```


TBB

TBB

```
find_package(TBB REQUIRED)
target_link_libraries( ${PROJECT_NAME} PUBLIC TBB::tbb)
```

Other

Kokkos

```
find_package(Kokkos REQUIRED)
target_link_libraries(myTarget Kokkos::kokkos)
```

Either set **CMAKE_PREFIX_PATH** or add

```
-DKokkos_ROOT=<Kokkos Install Directory>/lib64/cmake/Kokkos
```

Maybe:

```
-DCMAKE_CXX_COMPILER=<Kokkos Install Directory>/bin/
  nvcc_wrapper
```

See https://kokkos.org/kokkos-core-wiki/ProgrammingGuide/Compiling.html

Data packages

C:

```
find_package( PkgConfig REQUIRED )
pkg_check_modules( NETCDF REQUIRED netcdf )

target_include_directories(
        ${PROJECTNAME} PUBLIC
        ${NETCDF_INCLUDE_DIRS} )
target_link_libraries(
        ${PROJECTNAME} PUBLIC
        ${NETCDF_LIBRARIES} )
target_link_directories(
        ${PROJECTNAME} PUBLIC
        ${NETCDF_LIBRARY_DIRS} )
target_link_libraries(
        ${PROJECTNAME} PUBLIC netcdf )
```

Hdf5, Fortran

```
find_package( PkgConfig REQUIRED )
pkg_check_modules( NETCDFF REQUIRED netcdf-fortran )
pkg_check_modules( NETCDF REQUIRED netcdf )

target_include_directories(
       ${PROJECTNAME} PUBLIC
       ${NETCDFF_INCLUDE_DIRS}
       )
target_link_libraries(
       ${PROJECTNAME} PUBLIC
       ${NETCDFF_LIBRARIES} ${NETCDF_LIBRARIES}
       )
target_link_directories(
       ${PROJECTNAME} PUBLIC
       ${NETCDFF_LIBRARY_DIRS} ${NETCDF_LIBRARY_DIRS}
       )
target_link_libraries(
       ${PROJECTNAME} PUBLIC netcdf )
```

HighFive

Third party C++ interface to hdf5

```
find_package( HighFive REQUIRED )
target_link_libraries( ${PROJECTNAME} HighFive)
```