A CMake build system for deal.II

Matthias Maier

Institute of Applied Mathematics
Heidelberg University

Fourth deal.II Users' and Developers' Workshop
College Station, TX, USA

Aug 21, 2013

Part I

A build system for deal.II

Introducing CMake

An overview of build systems

What is a build system, anyway?

A build system should

- configure and set up external dependencies

  `MPI / BLAS / LAPACK / etc.`

- build and install the library and executables
- provide configuration for external projects
- run tests
- (other demands)

Over 150 known “build tools” listed on Wikipedia. . .

An overview of build systems

What is a build system, anyway?

A build system should

- configure and set up external dependencies

  `MPI / BLAS / LAPACK / etc.`

- build and install the library and executables
- provide configuration for external projects
- run tests
- (other demands)

Over 150 known "build tools" listed on Wikipedia. . .

An overview of build systems

(The usual rant)

cake

MakeMake

qmake

Cabal

Shake

SCons

GNU Build Tools

*.Jam

CMake

MSBuild

SBuild

GYP

xmake

Ant

Waf

tup

imake

tmake

An overview of build systems

(The usual rant)

cake

MakeMake

qmake

Cabal

Shake

SCons

GNU Build Tools

*.Jam

CMake

MSBuild

SBuild

GYP

xmake

Ant

Waf

tup

imake

tmake

The rest, aka
the abyss

The land of the
Make-like tools

An overview of build systems

(The usual rant)

`cake`

`MakeMake`

`SCons`

`GNU Build Tools`

`*.Jam`

`CMake`

`GYP`

`xmake`

`Ant`

`Waf`

`tup`

`imake`

`tmake`

The rest, aka
the abyss

An overview of build systems

(The usual rant)

SCons

GNU Build Tools

CMake

GYP

Waf

An overview of build systems

(The usual rant)

GNU Build Tools

CMake

An overview of build systems

(The usual rant)

An overview of build systems

(The usual rant)

`CMake`

Introducing CMake

- *CMake [is a] cross-platform, open-source build system, [it is] a familiy of tools designed to build, test and package software.*
- Started 1999 as a cross-platform build system for the *Insight Segmentation and Registration Toolkit* (ITK)
- Maintained by Kitware; 3-clause BSD license
- References: KDE, Blender, LLVM & Clang, ROOT, Reference BLAS, Reference LAPACK

- Good cross-platform support
- CMake is a flexible Makefile generator
- Fast, scales well (the cache. . .)
- Has a simple and expressive (but sometimes awkward) scripting language

Introducing CMake

- *CMake [is a] cross-platform, open-source build system, [it is] a familiy of tools designed to build, test and package software.*
- Started 1999 as a cross-platform build system for the *Insight Segmentation and Registration Toolkit* (ITK)
- Maintained by Kitware; 3-clause BSD license
- References: KDE, Blender, LLVM & Clang, ROOT, Reference BLAS, Reference LAPACK

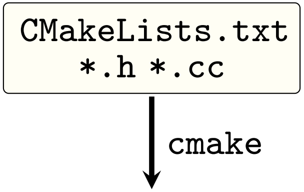
- Good cross-platform support
- CMake is a flexible Makefile generator
- Fast, scales well (the cache. . .)
- Has a simple and expressive (but sometimes awkward) scripting language

Introducing CMake

```
CMakeLists.txt
  *.h *.cc
```

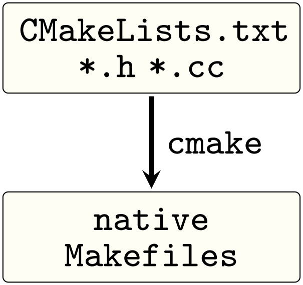
- platform independent

Introducing CMake



- platform independent

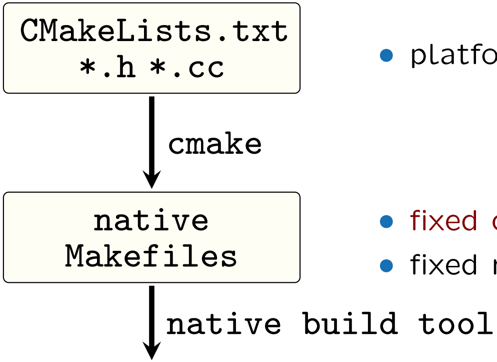
Introducing CMake



- platform independent

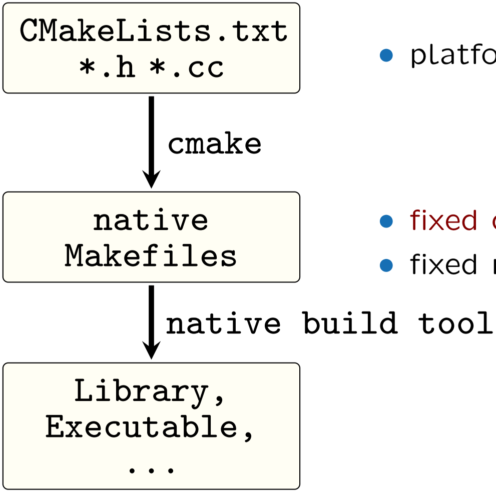
- fixed configuration
- fixed native build tool/compiler

Introducing CMake



- platform independent
- fixed configuration
- fixed native build tool/compiler

Introducing CMake

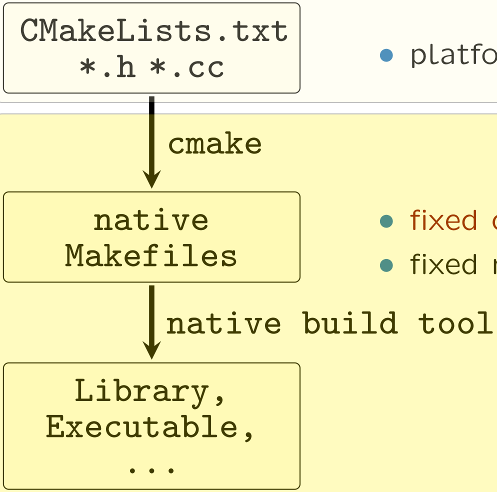


- platform independent
- fixed configuration
- fixed native build tool/compiler

Introducing CMake

Developers

- platform independent



- fixed configuration
- fixed native build tool/compiler

Users

Introducing CMake

`CMakeCache.txt` - the CMake cache

- Variables are held in a global table (private scope for subfolders and functions)
- Upon invocation of cmake:
  - Read in all cached variables out of `CMakeCache.txt` and populate the global table
  - `SET(foobar ...)` manipulates the table
  - `SET(foobar ... CACHE)` sets variable in cache and table iff not already present in cache.

This allows for efficient caching of configuration and search results.

For dealII: Initial 17.6 sec, subsequent 0.690 sec.

Introducing CMake

`CMakeCache.txt` - the CMake cache

- Variables are held in a global table (private scope for subfolders and functions)
- Upon invocation of cmake:
  - Read in all cached variables out of `CMakeCache.txt` and populate the global table
  - `SET(foobar ...)` manipulates the table
  - `SET(foobar ... CACHE)` sets variable in cache and table iff not already present in cache.

This allows for efficient caching of configuration and search results.

For dealII: Initial 17.6 sec, subsequent 0.690 sec.

Introducing CMake

`CMakeCache.txt` - the CMake cache

- Variables are held in a global table (private scope for subfolders and functions)
- Upon invocation of cmake:
  - Read in all cached variables out of `CMakeCache.txt` and populate the global table
  - `SET(foobar ...)` manipulates the table
  - `SET(foobar ... CACHE)` sets variable in cache and table iff not already present in cache.

This allows for efficient caching of configuration and search results.

For dealII: Initial 17.6 sec, subsequent 0.690 sec.

Introducing CMake

`CMakeCache.txt` - the CMake cache

- Variables are held in a global table (private scope for subfolders and functions)
- Upon invocation of cmake:
  - Read in all cached variables out of `CMakeCache.txt` and populate the global table
  - `SET(foobar ...)` manipulates the table
  - `SET(foobar ... CACHE)` sets variable in cache and table iff not already present in cache.

This allows for efficient caching of configuration and search results.

For dealII: Initial 17.6 sec, subsequent 0.690 sec.

A build system for deal.II

A build system for deal.II

Design goals

- Cross-platform support with a minimal set of dependencies:
  - CMake and a native build tool (no Perl, Python)
  - a native C++-compiler
- Flexible support of external dependencies
  - compiled by user
  - compiled and installed by user
  - packaged by a distribution
- Be as easy to use as possible (autodetection)
- . . . but try to support uncommon use cases
- Provide a possibility to easily override any configuration
- Support out-of source builds

A build system for deal.II

External libraries (insanity can't be far away. . .)

I want to link an external library `foobar`:

- `-L<..> -lfoobar` is not enough
- full transitive link interface might be needed
  (static archives, C++ code with templates)

And how to get the transitive link interface?

- There are solutions for this. . .
  (libtool's la-files, pkg-config, package provides config)
- . . . but none is guaranteed to be available.

So, do it (mostly) by hand. . .

A build system for deal.II

External libraries (insanity can't be far away. . .)

I want to link an external library `foobar`:

- `-L<..> -lfoobar` is not enough
- full transitive link interface might be needed (static archives, C++ code with templates)

And how to get the transitive link interface?

- There are solutions for this. . . (libtool's la-files, pkg-config, package provides config)
- . . . but none is guaranteed to be available.

So, do it (mostly) by hand. . .

A build system for deal.II

External libraries (insanity can't be far away. . .)

I want to link an external library **`foobar`**:

- **`-L<..> -lfoobar`** is not enough
- full transitive link interface might be needed (static archives, C++ code with templates)

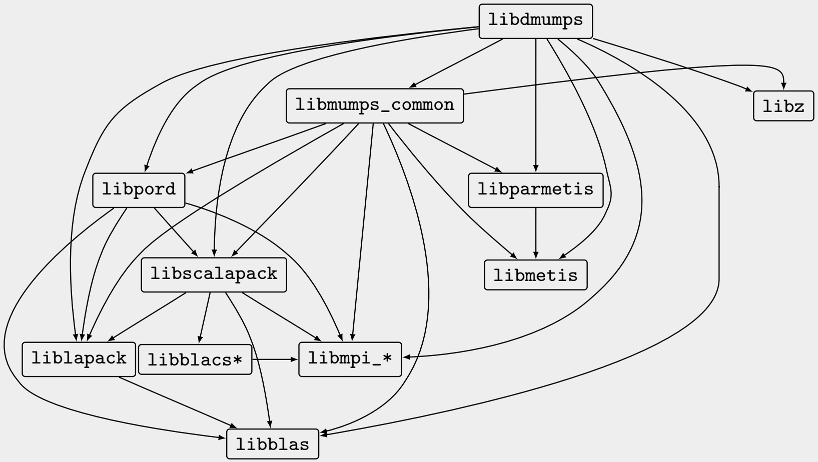
And how to get the transitive link interface?

- There are solutions for this. . . (libtool's la-files, pkg-config, package provides config)
- . . . but none is guaranteed to be available.

So, do it (mostly) by hand. . .

A build system for deal.II

An example:



Now, try this with Trilinos or PETSc.

A build system for deal.II

A quick overview:

- External dependencies are controlled with `DEAL_II_WITH_*` variables
- Options to override behaviour
  - `DEAL_II_ALLOW_AUTODETECTION`
  - `DEAL_II_ALLOW_BUNDLED`
  - `DEAL_II_FORCE_*`
- `deal.II/doc/development/Config.sample`

Some statistics:

External dependencies:	1841 SLOC (3495 total)
Platform and Compiler checks:	683 SLOC (1749 total)
Complete build system:	4658 SLOC (9548 total)

A build system for deal.II

A quick overview:

- External dependencies are controlled with `DEAL_II_WITH_*` variables
- Options to override behaviour
  - `DEAL_II_ALLOW_AUTODETECTION`
  - `DEAL_II_ALLOW_BUNDLED`
  - `DEAL_II_FORCE_*`
- `deal.II/doc/development/Config.sample`

Some statistics:

External dependencies:	1841 SLOC (3495 total)
Platform and Compiler checks:	683 SLOC (1749 total)
Complete build system:	4658 SLOC (9548 total)

Demo

Demo

Configure and let's see what happens:

```
deal.II % ls
CMakeLists.txt cmake include source [...]

deal.II % mkdir build
deal.II % cd build

build % cmake ..
```

Demo

```
build % cmake ..
-- This is CMake 2.8.11.1
[...]
-- Configuring done. Proceed to target definitions now.
###
#
#  deal.II configuration:
#       CMAKE_BUILD_TYPE:        DebugRelease
#       BUILD_SHARED_LIBS:       ON
#       CMAKE_INSTALL_PREFIX:    /usr/local
#       CMAKE_SOURCE_DIR:        /tmp/deal.II (Version 8.1.pre)
#       CMAKE_BINARY_DIR:        /tmp/deal.II/build
[...]
#  Configured Features:
#         DEAL_II_WITH_BOOST set up with external dependencies
#       ( DEAL_II_WITH_METIS = OFF )
[...]
#  Component configuration:
#       ( DEAL_II_COMPONENT_DOCUMENTATION = OFF )
[...]
#
###
-- Configuring done
-- Generating done
-- Build files have been written to: /tmp/deal.II/build
```

Demo

```
build % cmake ..
-- This is CMake 2.8.11.1
[...]
-- Configuring done. Proceed to target definitions now.
###
#
#  deal.II configuration:
#        CMAKE_BUILD_TYPE:          DebugRelease
#        BUILD_SHARED_LIBS:         ON
#        CMAKE_INSTALL_PREFIX:      /usr/local
#        CMAKE_SOURCE_DIR:          /tmp/deal.II (Version 8.1.pre)
#        CMAKE_BINARY_DIR:          /tmp/deal.II/build
[...]
#  Configured Features:
#          DEAL_II_WITH_BOOST set up with external dependencies
#        ( DEAL_II_WITH_METIS = OFF )
[...]
#  Component configuration:
#        ( DEAL_II_COMPONENT_DOCUMENTATION = OFF )
[...]
#
###
-- Configuring done
-- Generating done
-- Build files have been written to: /tmp/deal.II/build
```

Demo

```
build % cmake ..
-- This is CMake 2.8.11.1
[...]
-- Configuring done. Proceed to target definitions now.
###
#
#  deal.II configuration:
#        CMAKE_BUILD_TYPE:        DebugRelease
#        BUILD_SHARED_LIBS:       ON
#        CMAKE_INSTALL_PREFIX:    /usr/local
#        CMAKE_SOURCE_DIR:        /tmp/deal.II (Version 8.1.pre)
#        CMAKE_BINARY_DIR:        /tmp/deal.II/build
[...]
#  Configured Features:
#        DEAL_II_WITH_BOOST set up with external dependencies
#        ( DEAL_II_WITH_METIS = OFF )
[...]
#  Component configuration:
#        ( DEAL_II_COMPONENT_DOCUMENTATION = OFF )
[...]
#
###
-- Configuring done
-- Generating done
-- Build files have been written to: /tmp/deal.II/build
```

Demo

So let's correct for this:

```
build % cmake -DCMAKE_INSTALL_PREFIX=$PWD .
```

```
build % cmake -DDEAL_II_WITH_METIS=ON \
              -DMETIS_DIR=/tmp/metis .
```

Remark

- In many cases the `DEAL_II_` prefix can be omitted on the command line and will be automatically expanded (for `_WITH_`, `_ALLOW_`, `_FORCE_`)
- Alternatively, use the graphical editor!

```
build % ccmake .
```

Demo

So let's correct for this:

```
build % cmake -DCMAKE_INSTALL_PREFIX=$PWD .
```

```
build % cmake -DDEAL_II_WITH_METIS=ON \
              -DMETIS_DIR=/tmp/metis .
```

Remark

- In many cases the `DEAL_II_` prefix can be omitted on the command line and will be automatically expanded (for `_WITH_`, `_ALLOW_`, `_FORCE_`)
- Alternatively, use the graphical editor!

```
build % ccmake .
```

Demo

And what if the `METIS_DIR` is accidently omitted?

```
build % cmake -DDEAL_II_WITH_METIS=ON .
[...]
CMake Error at cmake/macros/macro_configure_feature.cmake:116 (MESSAGE)

  Could not find the metis library!

  Could not find a sufficient modern metis installation: Version 5.x
  required!

  Please ensure that a suitable metis library is installed on your
  computer.

  If the library is not at a default location, either provide some
  hints for autodetection,

      $ METIS_DIR="..." cmake <...>
      $ cmake -DMETIS_DIR="..." <...>

  or set the relevant variables by hand in ccmake.
[...]
build %
```

Demo

Finally, build and install the library:

```
build % make -j8
build % make install
```

Tip:

Try **ninja**!

```
build % cmake -GNinja ..
build % ninja
build % ninja install
```

Remark:

And if you just don't care about out-of-source builds:

```
deal.II % cmake .
deal.II % make -j8 install
```

Demo

Finally, build and install the library:

```
build % make -j8
build % make install
```

Tip:

Try **ninja**!

```
build % cmake -GNinja ..
build % ninja
build % ninja install
```

Remark:

And if you just don't care about out-of-source builds:

```
deal.II % cmake .
deal.II % make -j8 install
```

Demo

Finally, build and install the library:

```
build % make -j8
build % make install
```

Tip:

Try **`ninja`**!

```
build % cmake -GNinja ..
build % ninja
build % ninja install
```

Remark:

And if you just don't care about out-of-source builds:

```
deal.II % cmake .
deal.II % make -j8 install
```

Part II

Using CMake in user projects

Hello Deallog

Hello World

Just a **`hello_world.cc`** file:

Project:

```
project % ls
CMakeLists.txt hello_world.cc
```

CMakeLists.txt

```
ADD_EXECUTABLE(hello_world
  hello_world.cc
  )
```

Hello Deallog

A **`hello_deallog.cc`** file with dependency on deal.II:

CMakeLists.txt

```
CMAKE_MINIMUM_REQUIRED(VERSION 2.8.8)
PROJECT(hello_world CXX)

FIND_PACKAGE(deal.II 8.0 REQUIRED)

ADD_EXECUTABLE(hello_deallog
  hello_deallog.cc
  )
DEAL_II_SETUP_TARGET(hello_deallog)
```

`DEAL_II_SETUP_TARGET` will

- add include directories
- add compile definitions
- add linker flags and the link interface

Hello Deallog

A **hello_deallog.cc** file with dependency on deal.II:

CMakeLists.txt

```
CMAKE_MINIMUM_REQUIRED(VERSION 2.8.8)
PROJECT(hello_world CXX)

FIND_PACKAGE(deal.II 8.0 REQUIRED)

ADD_EXECUTABLE(hello_deallog
  hello_deallog.cc
  )
DEAL_II_SETUP_TARGET(hello_deallog)
```

DEAL_II_SETUP_TARGET will

- add include directories
- add compile definitions
- add linker flags and the link interface

Hello Deallog

A **hello_deallog.cc** file with dependency on deal.II:

CMakeLists.txt

```
CMAKE_MINIMUM_REQUIRED(VERSION 2.8.8)
PROJECT(hello_world CXX)

FIND_PACKAGE(deal.II 8.0 REQUIRED)

ADD_EXECUTABLE(hello_deallog
  hello_deallog.cc
  )
DEAL_II_SETUP_TARGET(hello_deallog)
```

DEAL_II_SETUP_TARGET will

- add include directories
- add compile definitions
- add linker flags and the link interface

Hello Deallog

And with build type handling and compiler flags:

CMakeLists.txt

```
CMAKE_MINIMUM_REQUIRED(VERSION 2.8.8)

FIND_PACKAGE(deal.II 8.0 REQUIRED)
DEAL_II_INITIALIZE_CACHED_VARIABLES()
PROJECT(hello_world CXX)

ADD_EXECUTABLE(hello_deallog
  hello_deallog.cc
  )
DEAL_II_SETUP_TARGET(hello_deallog)
```

DEAL_II_INITIALIZE_CACHED_VARIABLES will

- setup CMAKE_BUILD_TYPE (Debug, Release)
- setup CMAKE_CXX_FLAGS (*_DEBUG, *_RELEASE)

Hello Deallog

And with build type handling and compiler flags:

CMakeLists.txt

```
CMAKE_MINIMUM_REQUIRED(VERSION 2.8.8)

FIND_PACKAGE(deal.II 8.0 REQUIRED)
DEAL_II_INITIALIZE_CACHED_VARIABLES()
PROJECT(hello_world CXX)

ADD_EXECUTABLE(hello_deallog
  hello_deallog.cc
  )
DEAL_II_SETUP_TARGET(hello_deallog)
```

`DEAL_II_INITIALIZE_CACHED_VARIABLES` will

- setup `CMAKE_BUILD_TYPE` (`Debug`, `Release`)
- setup `CMAKE_CXX_FLAGS` (`*_DEBUG`, `*_RELEASE`)

Hello Deallog

Dependency on an external library:

```
# If there is a FindFOO.cmake module:
FIND_PACKAGE(foo 8.0 REQUIRED)
INCLUDE_DIRECTORIES(${FOO_INCLUDE_DIRS})

ADD_EXECUTABLE(hello_deallog hello_deallog.cc)
DEAL_II_SETUP_TARGET(hello_deallog)
TARGET_LINK_LIBRARIES(hello_deallog
   ${FOO_LIBRARIES}
   )
```

```
# And with pkg-config:
FIND_PACKAGE(PkgConfig)
PKG_CHECK_MODULES(FOO REQUIRED foo)
# Rest as above (Better write a FindFOO.cmake)
```

Hello Deallog

Dependency on an external library:

```
# If there is a FindFOO.cmake module:
FIND_PACKAGE(foo 8.0 REQUIRED)
INCLUDE_DIRECTORIES(${FOO_INCLUDE_DIRS})

ADD_EXECUTABLE(hello_deallog hello_deallog.cc)
DEAL_II_SETUP_TARGET(hello_deallog)
TARGET_LINK_LIBRARIES(hello_deallog
  ${FOO_LIBRARIES}
  )
```

```
# And with pkg-config:
FIND_PACKAGE(PkgConfig)
PKG_CHECK_MODULES(FOO REQUIRED foo)
# Rest as above (Better write a FindFOO.cmake)
```

Hello Deallog

Dependency on an external library:

```
# If there is a FindFOO.cmake module:
FIND_PACKAGE(foo 8.0 REQUIRED)
INCLUDE_DIRECTORIES(${FOO_INCLUDE_DIRS})

ADD_EXECUTABLE(hello_deallog hello_deallog.cc)
DEAL_II_SETUP_TARGET(hello_deallog)
TARGET_LINK_LIBRARIES(hello_deallog
  ${FOO_LIBRARIES}
  )
```

```
# And with pkg-config:
FIND_PACKAGE(PkgConfig)
PKG_CHECK_MODULES(FOO REQUIRED foo)
# Rest as above (Better write a FindFOO.cmake)
```

Hello Deallog

Declare compile definitions:

```
...
ADD_DEFINITIONS(-DFOO -DNAME="Matthias␣Maier")

...
ADD_EXECUTABLE(hello_deallog hello_deallog.cc)
```

Or the modern way to do this:

```
...
ADD_EXECUTABLE(hello_deallog hello_deallog.cc)
SET_PROPERTY(TARGET hello_deallog
  APPEND PROPERTY COMPILE_DEFINITIONS
  FOO
  NAME="Matthias␣Maier"
  )
```

Hello Deallog

Declare compile definitions:

```
...
ADD_DEFINITIONS(-DFOO -DNAME="Matthias␣Maier")

...
ADD_EXECUTABLE(hello_deallog hello_deallog.cc)
```

Or the modern way to do this:

```
...
ADD_EXECUTABLE(hello_deallog hello_deallog.cc)
SET_PROPERTY(TARGET hello_deallog
  APPEND PROPERTY COMPILE_DEFINITIONS
  FOO
  NAME="Matthias␣Maier"
  )
```

Advanced example

Advanced example

An advanced example:

- A support library **`Kiyo`** providing wrappers for third party libraries
- A library **`Grendel`** that provides multiscale data structures and algorithms
- An executable **`Cynder`**

Project structure:

```
project % ls

cynder/{*.h|*.cc}
grendel/{*.h|*.cc}
kiyo/{*.h|*.cc}
```

Advanced example

The top level `CMakeLists.txt` file:

./CMakeLists.txt

```
CMAKE_MINIMUM_REQUIRED(VERSION 2.8.8)

FIND_PACKAGE(deal.II 8.0 REQUIRED
  HINTS ${DEAL_II_DIR} $ENV{DEAL_II_DIR}
  )

DEAL_II_INITIALIZE_CACHED_VARIABLES()

PROJECT(cynder CXX)

ADD_SUBDIRECTORY(kiyo)
ADD_SUBDIRECTORY(grendel)
ADD_SUBDIRECTORY(cynder)
```

Advanced example

The top level **`CMakeLists.txt`** file:

./CMakeLists.txt

```
CMAKE_MINIMUM_REQUIRED(VERSION 2.8.8)

FIND_PACKAGE(deal.II 8.0 REQUIRED
  HINTS ${DEAL_II_DIR} $ENV{DEAL_II_DIR}
  )

DEAL_II_INITIALIZE_CACHED_VARIABLES()

PROJECT(cynder CXX)

ADD_SUBDIRECTORY(kiyo)
ADD_SUBDIRECTORY(grendel)
ADD_SUBDIRECTORY(cynder)
```

Advanced example

The top level **`CMakeLists.txt`** file:

./CMakeLists.txt

```
CMAKE_MINIMUM_REQUIRED(VERSION 2.8.8)

FIND_PACKAGE(deal.II 8.0 REQUIRED
  HINTS ${DEAL_II_DIR} $ENV{DEAL_II_DIR}
  )

DEAL_II_INITIALIZE_CACHED_VARIABLES()

PROJECT(cynder CXX)

ADD_SUBDIRECTORY(kiyo)
ADD_SUBDIRECTORY(grendel)
ADD_SUBDIRECTORY(cynder)
```

Advanced example

For the support library **Kiyo**:

kiyo/CMakeLists.txt

```
FIND_PACKAGE(quantim REQUIRED)

INCLUDE_DIRECTORIES(
  ${QUANTIM_INCLUDE_DIRS}
  )

ADD_LIBRARY(kiyo SHARED
  boundary_values.cc
  coefficients.cc
  ...
  )
DEAL_II_SETUP_TARGET(kiyo)
TARGET_LINK_LIBRARIES(kiyo ${QUANTIM_LIBRARIES}
```

Advanced example

For the support library `Kiyo`:

kiyo/CMakeLists.txt

```
FIND_PACKAGE(quantim REQUIRED)

INCLUDE_DIRECTORIES(
  ${QUANTIM_INCLUDE_DIRS}
  )

ADD_LIBRARY(kiyo SHARED
  boundary_values.cc
  coefficients.cc
  ...
  )
DEAL_II_SETUP_TARGET(kiyo)
TARGET_LINK_LIBRARIES(kiyo ${QUANTIM_LIBRARIES}
```

Advanced example

Grendel that provides data structures and algorithms:

grendel/CMakeLists.txt

```
INCLUDE_DIRECTORIES(
  ${CMAKE_SOURCE_DIR}/kiyo
  )

ADD_LIBRARY(grendel SHARED
  averaged_tensor.cc
  diffusion_problem.cc
  ...
  )
DEAL_II_SETUP_TARGET(grendel)
```

Advanced example

Grendel that provides data structures and algorithms:

grendel/CMakeLists.txt

```
INCLUDE_DIRECTORIES(
  ${CMAKE_SOURCE_DIR}/kiyo
  )

ADD_LIBRARY(grendel SHARED
  averaged_tensor.cc
  diffusion_problem.cc
  ...
  )
DEAL_II_SETUP_TARGET(grendel)
```

Advanced example

Finally, the executable **Cynder**:

cynder/CMakeLists.txt

```
SET(DIM 2 CACHE STRING "The␣dimension")
INCLUDE_DIRECTORIES(
  ${CMAKE_SOURCE_DIR}/grendel
  ${CMAKE_SOURCE_DIR}/kiyo
  )
ADD_EXECUTABLE(cynder
  cynder.cc
  loop.cc
  )
DEAL_II_SETUP_TARGET(cynder)
TARGET_LINK_LIBRARIES(cynder grendel kiyo)
SET_PROPERTY(TARGET cynder APPEND
  PROPERTY COMPILE_DEFINITIONS DIM=${DIM}
  )
```

Advanced example

Finally, the executable **Cynder**:

cynder/CMakeLists.txt

```
SET(DIM 2 CACHE STRING "The␣dimension")
INCLUDE_DIRECTORIES(
  ${CMAKE_SOURCE_DIR}/grendel
  ${CMAKE_SOURCE_DIR}/kiyo
  )
ADD_EXECUTABLE(cynder
  cynder.cc
  loop.cc
  )
DEAL_II_SETUP_TARGET(cynder)
TARGET_LINK_LIBRARIES(cynder grendel kiyo)
SET_PROPERTY(TARGET cynder APPEND
  PROPERTY COMPILE_DEFINITIONS DIM=${DIM}
  )
```

Advanced example

A run folder and a **make run**-target for **Cynder**:

./CMakeLists.txt

```
# Copy folder run to build directory:
FILE(COPY run DESTINATION ${CMAKE_BINARY_DIR})
```

cynder/CMakeLists.txt

```
ADD_EXECUTABLE(cynder cynder.cc loop.cc)
SET_PROPERTY(TARGET cynder
  PROPERTY RUNTIME_OUTPUT_DIRECTORY
  ${CMAKE_BINARY_DIR}/run
  )
ADD_CUSTOM_TARGET(run
  COMMAND cynder
  WORKING_DIRECTORY ${CMAKE_BINARY_DIR}/run
  )
```

Advanced example

A run folder and a **make run**-target for **Cynder**:

./CMakeLists.txt

```
# Copy folder run to build directory:
FILE(COPY run DESTINATION ${CMAKE_BINARY_DIR})
```

cynder/CMakeLists.txt

```
ADD_EXECUTABLE(cynder cynder.cc loop.cc)
SET_PROPERTY(TARGET cynder
  PROPERTY RUNTIME_OUTPUT_DIRECTORY
  ${CMAKE_BINARY_DIR}/run
  )
ADD_CUSTOM_TARGET(run
  COMMAND cynder
  WORKING_DIRECTORY ${CMAKE_BINARY_DIR}/run
  )
```

Advanced example

A run folder and a **`make run`**-target for **`Cynder`**:

./CMakeLists.txt

```
# Copy folder run to build directory:
FILE(COPY run DESTINATION ${CMAKE_BINARY_DIR})
```

cynder/CMakeLists.txt

```
ADD_EXECUTABLE(cynder cynder.cc loop.cc)
SET_PROPERTY(TARGET cynder
  PROPERTY RUNTIME_OUTPUT_DIRECTORY
  ${CMAKE_BINARY_DIR}/run
  )
ADD_CUSTOM_TARGET(run
  COMMAND cynder
  WORKING_DIRECTORY ${CMAKE_BINARY_DIR}/run
  )
```

Advanced example

Install the project:

cynder/CMakeLists.txt

```
INSTALL(TARGETS cynder DESTINATION bin)
```

grendel/CMakeLists.txt

```
INSTALL(TARGETS grendel DESTINATION lib)
```

kiyo/CMakeLists.txt

```
INSTALL(TARGETS kiyo DESTINATION lib)
```

./CMakeLists.txt

```
INSTALL(DIRECTORY run/ DESTINATION run)
```

And beyond that...

Thank you for your attention!