Building Dune with CMake - Frequently Asked Questions

Dominic Kempf* and Christoph Grüninger♠

June 29, 2017

*Interdisziplinäres Zentrum für Wissenschaftliches Rechnen, Universität Heidelberg,
Im Neuenheimer Feld 368, D-69120 Heidelberg, Germany
♠Institut für Wasser- und Umweltsystemmodellierung, Universität Stuttgart,
Pfaffenwaldring 61, D-70569 Stuttgart, Germany

Contents

This document collects some frequently asked questions about building Dune with the new CMake build system. It is nowhere near complete and does not intend to be a technical documentation of the build system. Instead, it tries to sum up the essentials of the build system to users that do not (want to) care about the build system.

1 What is CMake anyway?

CMake...

- is an open source buildsystem tool developed at KITware.
- offers a one-tool-solution to all building tasks, like configuring, building, linking, testing and packaging.
- is a build system generator: It supports a set of backends called "generators"
- is portable
- is controlled by ONE rather simple language

Dune got support for CMake in version 2.3 alongside the old Autotools build system. It got the default in the 2.4 release. After that release, the Autotools build system will be removed from the master branch.
You can install cmake through your favorite package manager or downloading source code from www.cmake.org. The minimum required version to build Dune with CMake is 2.8.6.

2 How do I use Dune with CMake?

As with the Autotools, the build process is controlled by the script `dunecontrol`, located in `dune-common/bin`. There is a compatibility layer that will translate all the configure flags from your opts file into the corresponding CMake flags. While this is a great tool to determine how to do the transition, in the long run you should switch to a CMake-only approach.
`dunecontrol` will pickup the variable `CMAKE_FLAGS` from your opts file and use it as command line options for any call to CMake. There, you can define variables for the configure process with CMakes `-D` option; just as with the C pre-processor.
The most important part of the configure flags is to tell the build system where to look for external libraries. You can either use the variable `CMAKE_PREFIX_PATH` for that or set the variable given in the documentation of the corresponding find module (see below for details).

3 What files in a dune module belong to the CMake build system?

Every directory in a project contains a file called `CMakeLists.txt`, which is written in the CMake language. You can think of these as a distributed configure script. Upon configure, the top-level `CMakeLists.txt` is executed. Whenever an `add_subdirectory` command is encountered, the `CMakeLists.txt` of that sub-directory is executed. The top-level `CMakeLists.txt` is special, because it sets up the entire Dune module correctly. You should not delete the auto-generated parts of it.
Additionally, a Dune module can export some cmake modules. A cmake module is a file that contains one or more build system macros meant for downstream use. If a module provides modules, they can be found in the subfolder `cmake/modules`. The module

```
dune-foo/cmake/modules/DuneFooMacros.cmake
```

is special: Its contents are always executed when configuring the module `dune-foo` or any other Dune module, that requires or suggests the module `dune-foo`. This is the perfect place to put your checks for external packages, see section 6 below.
The file `config.h.cmake` defines a template for the section of `config.h`, that is generated by the module.
The file `stamp-regenerate-config-h` also belongs to the CMake build system. You can trigger regeneration of `config.h` by touching it.

4 How do I port an existing module?

There is multiple approaches to this:

- First, check section 8 and decide whether such simple approach is sufficient for your project.
- There is the python script `dune-common/bin/am2cmake.py`, which automatically generates `CMakeLists.txt`'s from its `Makefile.am` counterparts. While this works fine, for many modules the resulting files look very autotoolish and are often too complicated.
- Copy the top-level `CMakeLists.txt` file from a freshly generated Dune module (you still need to adjust the project name to have it match the module name) and write all other `CMakeLists.txt` by hand. This sounds cumbersome, but its actually not very much work if you read below advices.

In order to write your own `CMakeLists.txt` files you should be aware of the following basic commands of the CMake language.

- `add_subdirectory(dir)` immediately executes the `CMakeLists.txt` in the `dir` subfolder.
- `add_executable(target src1 [, src2 ..])` adds an executable named `target`. Note that, unlike in autotools, target names have to be unique throughout the entire project. The given sources are the `*.cc` files that are used to determine the dependencies of the target. Configuring the targets with the correct flags and linked libraries is described in section 5.
- `add_test(testname execname [args..])` registers a test. If `execname` matches a target name within the project, it is automatically replaced with the corresponding executable, but any executable may be given.
- `install(FILES files DESTINATION dest)` [1] Use to define the install location of a list of headers.

For a detailed reference, use:

```
cmake --help-command <command>
```

If your module requires any other packages than the dune modules listed in your `dune.module` file, you should also use the command `find_package` in the module `dune-foo/cmake/modules/DuneFooMacros.cmake` (as mentioned in section 3). How to do this with external packages not yet supported by Dune is covered in section 6.

[1] Note that CMake uses positional arguments for some commands and named arguments for more complicated ones. If you ever happen to write your own macro, try going for named arguments using the module `CMakeParseArguments`

5 How do I modify the flags and linked libraries of a given target?

Again, there are multiple ways to do this. The Dune build system offers macros to make this task as easy as possible. For each external module, there is a macro `add_dune_*_flags(targets)`. Those macros should cover most flags. Example usage:

```
add_executable(foo foo.cc)
add_dune_umfpack_flags(foo)
add_dune_mpi_flags(foo)
```

There is also the macro `add_dune_all_flags(targets)`, which uses the same flag registry mechansim then the simplfied build system in section 8.
If you want to fully control the configuration of the targets, you can do so. Build system entities such as targets, directories and tests do have so called properties in CMake. You can access and modify those properties via the commands `get_property` and `set_property`. You can for example use those to modify a targets `COMPILE_DEFINITIONS` or `INCLUDE_DIRECTORIES` property:

```
add_executable(foo foo.cc)
set_property(TARGET foo APPEND PROPERTY COMPILE_DEFINITIONS <somedefinition>)
set_property(TARGET foo APPEND PROPERTY INCLUDE_DIRECTORIES <somepath>)
```

For a full list of properties, check the manual:

```
cmake --help-property-list
```

Manually linking libraries can be done through the `target_link_libraries` command instead of manually tweaking properties.

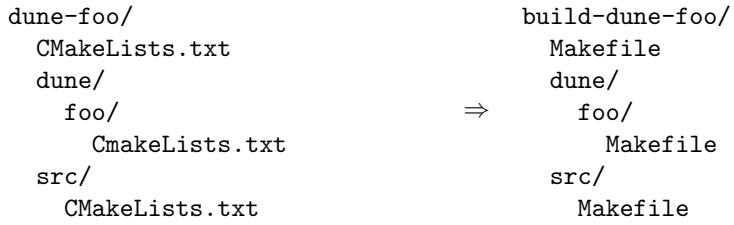
6 How do I link against external libraries, that are not checked for by Dune?

While there might be many solutions that make your application work, there is only one clean solution to this: You have to provide a find module for the package. A find module is a CMake module that follows a specific naming scheme: For an external package called `SomePackage` it is called `FindSomePackage.cmake`. Note that CMake treats package names case sensitive. If CMake encounters a `find_package(SomePackage)` line, it searches its module include paths for this find module. A good read to get started writing a find module is *How to find Libraries* in the CMake wiki.
Depending on how common your external package is, you may not even need to write the find module on your own. You can have a look at the list of find modules shipped by CMake [2] or simply search the internet for the module name and profit from other open-source project's work.
It is considered good style to also provide a macro `add_dune_somepackage_flags(targets)`.

7 What is an out-of-source build?

An out-of-source build does leave the version-controlled source tree untouched and puts all files that are generated by the build process into a different directory – the build directory. The build directory does mirror your source tree's structure as seen in the following figure:

[2] Linux distributions may put them at `/usr/share/cmake-<version>/modules` or `/usr/share/cmake/modules`



Using the `Unix Makefiles` generator, your Makefiles are generated in the build tree, so that is where you have to call `make`. There are multiple advantages with this approach, such as a clear separation between version controlled and generated files and you can have multiple out-of-source builds with different configurations at the same time.
Out-of-source builds are the default with CMake. In-source builds are strongly discouraged. By default, a subfolder `build-cmake` is generated within each dune module and is used as a build directory. You can customize this folder through the `--builddir` option of `dunecontrol`. Give an absolute path to the `--builddir` option, you will get something like this:

```
build/
  dune-common/
    Makefile
  dune-foo/
    Makefile
```

8 What is the new simplified build system and how do I use it?

Dune offers a simplified build system, where all flags are added to all targets and all libraries are linked to all targets. You can enable the feature with:

```
dune_enable_all_packages(INCLUDE_DIRS [include_dirs]
                         COMPILE_DEFINITIONS [compile_definitions]
                         MODULE_LIBRARIES [libraries]
                         [VERBOSE] [APPEND]
                        )
```

This will modify all targets in the directory of the `CMakeLists.txt`, where you put this, and also in all sub-directories. The compile flags for all found external packages are added to those targets and the target is linked against all found external libraries. The `VERBOSE` option will prompt those flags upon configure. This is especially useful for application modules.
To use this while using custom external packages, you have to register your flags. Check the module

```
dune-common/cmake/modules/DuneEnableAllPackages.cmake
```

Some special care has to be given, if your module does build one or more library which targets within the module do link against. Carefully read the in-module documentation of above module in that case.

9 How do I change my compiler and compiler flags?

In general, there are multiple ways to do this:

- Setting the CMake variables `CMAKE_{C,CXX}_COMPILER` from the opts file

- Setting those variables within the project with the `set` command
- Setting the environment variables `CC`, `CXX` etc.

The first option is the recommended way. Whenever you change your compiler, you should delete all build directories. For some CMake versions, there is a known CMake bug, that requires you to give an absolute path to your compiler, but Dune will issue a warning, if you violate that.
You can modify your default compiler flags by setting the variables `CMAKE_{C,CXX}_FLAGS` in your opts file. Note, you can define build-type specific flags with `CMAKE_{C,CXX}_FLAGS_{DEBUG,RELEASE}`. You can switch the build type with `CMAKE_BUILD_TYPE={Release,Debug}`.

10 How should I handle ini and grid files in an out-of-source-build setup?

Such files are under version control, but they are needed in the build directory. The module

```
dune-common/cmake/modules/DuneSymlinkOrCopy.cmake
```

delivers macros for that purpose.
The simplest way to solve the problem is to add `-DDUNE_SYMLINK_TO_SOURCE_TREE=1` to your opts file. This will execute `dune_symlink_to_source_tree()` to your top-level `CMakeLists.txt`. This will add a symlink `src_dir` to all subdirectories of the build directory, which points to the corresponding directory of the source tree. This will only work on platforms that support symlinking. For other (more portable) solutions, check the documentation of above module.

11 How do I use CMake with IDEs?

As already said, CMake is merely a build system generator with multiple backends (called a generator). Using IDEs requires a different generator. Check `cmake --help` for a list of generators. You can then add the `-G` to the `CMAKE_FLAGS` in your opts file, like:

```
-G'Eclipse␣CDT4␣-␣Unix␣Makefiles'
```

Note that the generator name has to match character by character, including case and spaces.

12 I usually modify my CXXFLAGS upon calling make. How can I do this in CMake?

This violates the CMake philosophy and there is no clean solution to achieve it. The CMake-ish solution would be to have for each configuration one out-of-source build. We have nevertheless implemented a workaround. It can be enable by setting `ALLOW_CXXFLAGS_OVERWRITE=ON` in your opts file. You can then type:

```
make CXXFLAGS="<your␣flags>" <target>
```

Furthermore any C pre-processor variable of the form `-DVAR=<value>` can be overloaded on the command line and the grid type can be set via `GRIDTYPE="<grid␣type>"`.
Note this only works with generators that are based on Makefiles and several Unix tools like bash must be available.

13 How do I run the test suite from CMake?

The built-in target to run the tests is called `test` instead of Autotools' `check`. It is a mere wrapper around CMake's own testing tool CTest. You can check `ctest --help` for a lot of useful options, such as choosing the set of tests to be run by matching regular expressions or showing the output of failed tests.
Although this is not the CMake-ish way, `make test` also builds the tests before executing them. This behavior will change in the near future.

14 Where can I get a list of all configuration options?

There are two ways. Either use `ccmake` which is a graphical user interface and shows you all variables and their values after a configure run. You can edit these values, too. Or add `-LH` to the CMake call, then CMake will print all variables and their values after configuration. There is nothing comparable to Autotools' `configure --help`.

15 Can I disable an external dependency?

To disable an external dependency `Foo`, add

```
-DCMAKE_DISABLE_FIND_PACKAGE_Foo=TRUE
```

to your opts file. The name of the dependency is case sensitive but there is no canonical naming scheme; it can be `FOO`, `Foo`, or even something else. See the output of configure to get the right name.
Make sure to not use cached configure results by deleting the cache file or the build directory, cf. 18.

16 How do I switch between parallel and sequential builds?

Dune builds with CMake are parallel if and only if MPI is found. To have a sequential build despite an installed MPI library, you have to explicitly disable the corresponding find module by setting

```
-DCMAKE_DISABLE_FIND_PACKAGE_MPI=TRUE
```

in the `CMAKE_FLAGS` of your opts file.

17 Why is it not possible anymore to do make headercheck?

The headercheck feature has been disabled by default. You can enable it by setting the CMake variable `ENABLE_HEADERCHECK=1` through your opts file. This step has been necessary, because of the large amount of additional file the headercheck adds to the build directory. A better implementation has not been found yet, because it simply does not fit the CMake philosophy.

18 How do I troubleshoot?

CMake caches aggressively which makes it bad at recognizing changed configurations.

To trigger a fresh run of configure, you can delete the `CMakeCache.txt` file from the build directory and maybe save some compilation time afterward.
Whenever you experience any problems, your first step should be to delete all build directories. Nice trick:

```
dunecontrol exec rm -rf build-cmake
```

This will remove all build directories from all DUNE modules.
Later on you can get an error log from the file `CMakeError.log` in the `CMakeFiles` sub-directory of your build directory. This is what you should send to the mailing list alongside the description of your setup and efforts to help us help you.

19 Where can I get help?

The CMake manual is available on the command line:

- `cmake --help-command-list`
- `cmake --help-command <command>`
- `cmake --help-property-list`
- `cmake --help-property <property>`
- `cmake --help-module-list`
- `cmake --help-module <module>`

To get help on which variables are picked up by CMake, there is a CMake wiki page collecting them. Of course, there is also Google, StackOverflow and the CMake Mailing list (archive). For problems specific to DUNE's build system, ask on our mailing lists.