# Installing prerequisite software

An alternative to running the book recipes in a container is to install the dependencies directly on the host operating system. For this, we have assembled a minimal toolstack that can be used as a basic starting point for all of our recipes. You will have to install the following:

1. CMake
2. Language-specific tools, that is, the compilers
3. Build automation tools
4. Python

We will also detail how to install the additional dependencies required by some of the recipes.

# Getting CMake

CMake 3.5 is the minimum required version of CMake for this book. Only a few, specific recipes and examples that demonstrate useful features that were introduced after version 3.5 will require a more recent version of CMake. The introduction to every recipe features an info box, pointing out where the code is available, which examples are given, and the minimum version of CMake required. The info boxes will look like the following box:

The code for this recipe is available at `https://github.com/dev-cafe/cmake-cookbook/tree/v1.0/chapter-03/recipe-10`, and includes a C example. The recipe is valid with CMake version 3.5 (and higher) and has been tested on GNU/Linux, macOS, and Windows.

Some, if not most, of the recipes will still be valid with older versions of CMake. However, we have made no attempts to test this assumption, since we consider CMake 3.5 to be the default on most systems and distributions. We also consider upgrading to later versions of CMake to be a straightforward step.

CMake can be installed in a number of different ways. Downloading and extracting the binary distribution maintained by Kitware will work across all platforms. The download page is at `https://cmake.org/download/`.

Most GNU/Linux distributions have CMake available in their package managers. However, on some distributions, the packaged version can be rather old, so downloading the binary maintained by Kitware is still the preferred option. The following commands will download and install CMake 3.5.2 under `$HOME/Deps/cmake` (adjust this path to your preference), from the version packaged by CMake:

```
$ cmake_version="3.5.2"
$ target_path=$HOME/Deps/cmake/${cmake_version}
$
cmake_url="https://cmake.org/files/v${cmake_version%.*}/cmake-${cmake_versi
on}-Linux-x86_64.tar.gz"
$ mkdir -p "${target_path}"
$ curl -Ls "${cmake_url}" | tar -xz -C "${target_path}" --strip-
components=1
$ export PATH=$HOME/Deps/cmake/${cmake_version}/bin${PATH:+:$PATH}
$ cmake --version
```

Homebrew for macOS reliably ships the latest version of CMake:

```
$ brew upgrade cmake
```

On Windows, you can use Visual Studio 2017, which provides CMake support. The installation of Visual Studio 2017 is documented in `Chapter 13`, *Alternative Generators and Cross-compilation*, Recipe 1, *Building a CMake project using Visual Studio 2017*.

Alternatively, you can download the MSYS2 installer from `https://www.msys2.org`, follow the instructions given therein to update the list of packages, and then install CMake using the package manager, `pacman`. The following code assumes that we are building the 64-bit version:

```
$ pacman -S mingw64/mingw-w64-x86_64-cmake
```

For the 32-bit version, use the following (though we will only refer to 64-bit versions in future, for the sake of brevity):

```
$ pacman -S mingw64/mingw-w64-i686-cmake
```

Another nice feature of MSYS2 is that it provides a terminal on Windows that feels and behaves like a terminal on a Unix-like operating system, providing a useful development environment.

# Compilers

We will need compilers for C++, C, and Fortran. These should be fairly recent, as we require support for recent language standards in most of the recipes. CMake offers very good support for many compilers, from both commercial and non-commercial vendors. To keep the recipes consistently cross-platform and as operating system independent as possible, we have worked with open source compilers:

- On GNU/Linux, the GNU Compiler Collection (GCC) is the obvious choice. It is free and available for all distributions. For example, on Ubuntu, you can install the compilers as follows:

```
$ sudo apt-get install g++ gcc gfortran
```

- Clang, in the LLVM family, is also a good choice for C++ and C:

```
$ sudo apt-get install clang clang++ gfortran
```

- On macOS, the LLVM compilers shipped with XCode will work for C++ and C. We have used the Fortran compiler from GCC in our macOS testing. This has to be installed separately, using the package manager. For example, the command for Homebrew is as follows:

```
$ brew install gcc
```

- On Windows, you can use Visual Studio for the C++ and C recipes. Alternatively, you can use the MSYS2 installer and install the entire toolchain, including a C++, C, and Fortran compiler, with the following single command in an MSYS2 environment (for the 64-bit version):

```
$ pacman -S mingw64/mingw-w64-x86_64-toolchain
```

# Build-automation tools

These build-automation tools will provide the infrastructure for building and linking the projects presented in the recipes. What you will end up installing and using strongly depends on your operating system and your taste:

- On GNU/Linux, GNU Make will most likely be installed automatically, when installing the compilers.
- On macOS, XCode will provide GNU Make.

- On Windows, Visual Studio will provide you with the complete infrastructure. In the MSYS2 environment, GNU Make is installed as a part of the `mingw64/mingw-w64-x86_64-toolchain` package, which we installed previously.

For maximum portability, we have made the recipes as agnostic about these system-dependent details as possible. A clear advantage of this approach is that configuring, building, and linking are native to each platform and each set of compilers.

The Ninja program is a different build-automation tool that works on GNU/Linux, macOS, and Windows. Ninja is a new build tool, with a focus on speed, especially for incremental rebuilds. Prepackaged binaries for GNU/Linux, macOS, and Windows can be found on the project's GitHub repository at `https://github.com/ninja-build/ninja/releases`.

Using CMake and Ninja with Fortran projects requires some care. CMake 3.7.2 or later is required, along with the version of Ninja maintained by Kitware, available at `https://github.com/Kitware/ninja/releases`.

On GNU/Linux, you can install Ninja with the following series of commands:

```
$ mkdir -p ninja
$
ninja_url="https://github.com/Kitware/ninja/releases/download/v1.8.2.g3bbbe
.kitware.dyndep-1.jobserver-1/ninja-1.8.2.g3bbbe.kitware.dyndep-1.jobserver
-1_x86_64-linux-gnu.tar.gz"
$ curl -Ls ${ninja_url} | tar -xz -C ninja --strip-components=1
$ export PATH=$HOME/Deps/ninja${PATH:+:$PATH}
```

On Windows, using the MSYS2 environment (assuming the 64-bit version), executing the command:

```
$ pacman -S mingw64/mingw-w64-x86_64-ninja
```

We recommend reading the essay at `http://www.aosabook.org/en/posa/ninja.html` for an enlightening discussion of Ninja's history and design choices.

# Python

This book is about CMake, but some of the recipes, along with the whole infrastructure powering testing, need Python. Thus, first and foremost, you will need a working installation of Python: the interpreter, header files, and libraries. The end of life for Python 2.7 was announced for 2020, and we will thus use Python 3.5.