

Install third-party libraries

[中文](#)

The template provides two ways of third-party library installation. You can mix them to install libraries if you like.

⚠ Please make sure the network, especially inside terminal, is accessible to github (for instance, in the "steam++ toolbox" app you can choose to speed up github access). If you're using a VPN, please enable "tun mode" or something alike.

Use vcpkg

Enable vcpkg

Edit `CMakeLists.txt` (located at the root of this repository folder), add a line `run_vcpkg()` between `include(cpp_novice_fetch_project_options)` and `project(cpp_novice LANGUAGES CXX)`. That is:

```
1 cmake_minimum_required(VERSION 3.25)
2
3 list(APPEND CMAKE_MODULE_PATH "${CMAKE_CURRENT_SOURCE_DIR}/cmake")
4 include(fix_msvc)
5 include(enable_cpp_module NO_POLICY_SCOPE)
6 include(cpp_novice_fetch_project_options)
7
8 run_vcpkg()
9 project(cpp_novice LANGUAGES CXX)
```

By adding this line of code, we enable cmake to install and use vcpkg automatically, so all we need to do next is to specify libraries required and link libraries to our programs.

Search libraries

First, we open the website [Browse public vcpkg packages](#) to search libraries we need in order to see whether they exist and what their exact names are.

Below I'll use `fmt` and `range-v3` libraries as the example.

Specify libraries required

Edit `vcpkg.json` (located at the root of this repository file) by simply adding the library names into the `dependencies` array.

For example, here's how we add `fmt` and `range-v3` library:

```

1 {
2   "$schema": "https://raw.githubusercontent.com/microsoft/vcpkg-
   tool/main/docs/vcpkg.schema.json",
3   "dependencies": ["fmt", "range-v3"]
4 }

```

Clear the CMake cache and reconfigure CMake in your IDE in some way (for example, you might delete the build or out folder and restart the software). If you're lucky, cmake will automatically install vcpkg for you, and use it to install `dependencies` you specified.

Link libraries to our programs

After reopening your IDE (or reconfigure cmake), cmake should output messages somewhere like the following example, which inform you on how to use the installed libraries:

```

1 [cmake] The package fmt provides CMake targets:
2 [cmake]
3 [cmake]     find_package(fmt CONFIG REQUIRED)
4 [cmake]     target_link_libraries(main PRIVATE fmt::fmt)
5 [cmake]
6 [cmake]     # Or use the header-only version
7 [cmake]     find_package(fmt CONFIG REQUIRED)
8 [cmake]     target_link_libraries(main PRIVATE fmt::fmt-header-only)
9 [cmake]
10 [cmake] range-v3 provides CMake targets:
11 [cmake]
12 [cmake]     # this is heuristically generated, and may not be correct
13 [cmake]     find_package(range-v3 CONFIG REQUIRED)
14 [cmake]     target_link_libraries(main PRIVATE range-v3::meta range-v3::concepts range-
   v3::range-v3)
15 [cmake]
16 [cmake] -- Running vcpkg install - done

```

Although the instruction is already simple, I simplified this in the template even further. Just edit the `add_program_options` function in `CMakeLists.txt` by:

- adding library names showed in `find_package(<name> CONFIG REQUIRED)` into `DEPENDENCIES` section.
- adding target names showed in `target_link_libraries(main PRIVATE <name>)` into `LIBRARIES` section.

For example, here's how we add `fmt` and `range-v3` library:

```

1 add_program_options(
2   DEPENDENCIES
3   fmt
4   range-v3
5
6   LIBRARIES
7   fmt::fmt

```

```

8     range-v3::meta
9     range-v3::concepts
10    range-v3::range-v3
11
12    INCLUDES
13    include
14 )

```

Reopen your IDE (or reconfigure cmake in your IDE somehow) again.

Done.

Use conan

Install conan

Install conan somehow. For example, you can download it from [the official website](#).

Enable conan

Edit `CMakeLists.txt` (located at the root of this repository file), add a line `run_conan()` between `include(cpp_novice_fetch_project_options)` and `project(cpp_novice LANGUAGES CXX)`. That is:

```

1  cmake_minimum_required(VERSION 3.25)
2
3  list(APPEND CMAKE_MODULE_PATH "${CMAKE_CURRENT_SOURCE_DIR}/cmake")
4  include(fix_msvc)
5  include(enable_cpp_module NO_POLICY_SCOPE)
6  include(cpp_novice_fetch_project_options)
7
8  run_conan()
9  project(cpp_novice LANGUAGES CXX)

```

By adding this line of code, we enable cmake to install and use vckpg automatically, so all we need to do next is to specify libraries required and link libraries to our programs.

Search libraries

First, we open the website [JFrog ConanCenter](#) to search libraries we need in order to see whether they exist, what their exact names are and what their latest version are.

Below I'll use `fmt/10.2.1` and `range-v3/0.12.0` libraries as the example.

Specify libraries required

Edit `conanfile.txt` (located at the root of this repository file) by simply adding the `<library_name>/<version>` into the `[requires]` section.

For example, here's how we add `fmt/10.2.1` and `range-v3/0.12.0` library:

```
1 [layout]
2 cmake_layout
3
4 [requires]
5 fmt/10.2.1
6 range-v3/0.12.0
7
8 [generators]
9 CMakeDeps
```

Clear the CMake cache and reconfigure CMake in your IDE in some way (for example, you might delete the build or out folder and restart the software). If you're lucky, cmake will automatically use conan to install `[requires]` you specified.

Link libraries to our programs

After reopening your IDE (or reconfigure cmake), cmake should output messages somewhere like the following example, which inform you on how to use the installed libraries:

```
1 [cmake] conanfile.txt: CMakeDeps necessary find_package() and targets for your
  CMakeLists.txt
2 [cmake]      find_package(fmt)
3 [cmake]      find_package(range-v3)
4 [cmake]      target_link_libraries(... fmt::fmt range-v3::range-v3)
5 [cmake] conanfile.txt: Generating aggregated env files
6 [cmake] conanfile.txt: Generated aggregated env files: ['conanbuild.sh',
  'conanrun.sh']
7 [cmake] Install finished successfully
```

Although the instruction is already simple, I simplified this in the template even further. Just edit the `add_program_options` function in `CMakeLists.txt` by:

- adding library names showed in `find_package(<name>)` into `DEPENDENCIES` section.
- adding target names showed in `target_link_libraries(... <name>)` into `LIBRARIES` section.

For example, here's how we add `fmt` and `range-v3` library:

```
1 add_program_options(
2     DEPENDENCIES
3     fmt
4     range-v3
5
6     LIBRARIES
7     fmt::fmt
8     range-v3::range-v3
9
10    INCLUDES
11    include
12 )
```

Reopen your IDE (or reconfigure cmake in your IDE somehow) again.

Done.

Use CMake's built-in support for certain third-party libraries

⚠ It is recommended to learn CMake through [Modern CMake for C++](#).

The previously mentioned tools, vcpkg and conan, are used to download (compile,) and install third-party libraries, and then provide CMake, which manages C++ projects, with a `Find<PackageName>.cmake`, `<package_name>-config.cmake`, or `<PackageName>Config.cmake` file, so that CMake can successfully locate and use these third-party libraries.

CMake itself also supports some third-party libraries (such as [FindOpenMP.cmake](#) and [FindQt.cmake](#)), so as long as you install these libraries into the system path, you can use them in CMake:

```
1  add_program_options(  
2      DEPENDENCIES  
3      OpenMP  
4  
5      LIBRARIES  
6      OpenMP::OpenMP_CXX  
7  
8      INCLUDES  
9      include  
10 )
```

Clear the CMake cache and reconfigure CMake in your IDE in some way (for example, you might delete the build or out folder and restart the software).

Done.