Cross-Compiling the C++ Example Project

Pieter P

The Greeter Library

For this example, we'll create a very simple library function that just takes a name and an output stream as arguments, and that prints a greeting message to this stream. It's basically a "Hello, World!" example, but as a library for demonstration purposes.

The structure of the library will be as follows:

```
greeter
    ├── CMakeLists.txt
    ├── include
    │   └── greeter
    │       └── greeter.hpp
    ├── src
    │   └── greeter.cpp
    └── test
        ├── CMakeLists.txt
        └── greeter.test.cpp
```

As is very common for C++ libraries, the function prototypes/declarations will be in the header file `greeter.hpp`. The implementations for these functions are in the implementation file `greeter.cpp`.
The `CMakeLists.txt` file in the `greeter` directory specifies how the library should be compiled, and where to find the headers. Additionally, there's a `test` folder with unit tests in `greeter.test.cpp`. The `CMakeLists.txt` file in this folder specifies how to compile and link the tests executable.

```hpp
#pragma once

#include <iosfwd>  // std::ostream
#include <string>  // std::string

namespace greeter {

/**
 * @brief   Function that greets a given person.
 *
 * @param   name
 *          The name of the person to greet.
 * @param   os
 *          The output stream to print the greetings to.
 */
void sayHello(const std::string &name, std::ostream &os);

}  // namespace greeter
```

```cpp
#include <greeter/greeter.hpp>
#include <iostream>

namespace greeter {

void sayHello(const std::string &name, std::ostream &os) {
    os << "Hello, " << name << "!" << std::endl;
}

}  // namespace greeter
```

```
add_library(greeter
    src/greeter.cpp
)

target_include_directories(greeter
    PUBLIC
        $<INSTALL_INTERFACE:include>
        $<BUILD_INTERFACE:${CMAKE_CURRENT_SOURCE_DIR}/include>
    PRIVATE
        src
)

add_subdirectory(test)
```

```
# Add a new executable with the name "hello-world" that is compiled from the
# source file "hello-world.cpp".
add_executable(hello-world
    hello-world.cpp
)

# The "hello-world" program requires the "greeter" library.
# The target_link_libraries command ensures that all compiler options such as
# include paths are set correctly, and that the executable is linked with the
# library as well.
target_link_libraries(hello-world
    greeter
)
```

```cpp
#include <greeter/greeter.hpp>  // Our own custom library

#include <iostream>  // std::cout, std::cin
#include <string>    // std::getline

int main(int argc, char *argv[]) {
    std::string name;
    if (argc > 1) {      // If the user passed arguments to our program
        name = argv[1];  // The name is the first argument
    } else {             // If not, ask the user for his name
        std::cout << "Please enter your name: ";
        std::getline(std::cin, name);
    }
    greeter::sayHello(name, std::cout);  // Greet the user
}
```

```
# Add a new executable with the name "hello-world" that is compiled from the
# source file "hello-world.cpp".
add_executable(hello-world
    hello-world.cpp
)

# The "hello-world" program requires the "greeter" library.
# The target_link_libraries command ensures that all compiler options such as
# include paths are set correctly, and that the executable is linked with the
# library as well.
target_link_libraries(hello-world
    greeter
)
```