# SDK Tutorial

# Overview

In this tutorial, you will learn how to use the Marlin Multicast SDK to send and receive information using the Marlin Network.

## Prerequisites

This tutorial assumes familiarity with using the command line. You will need a C++ compiler (supporting C++17), CMake (atleast 3.13 or higher) and Make installed on your computer.

## Contents

You will be guided through the following parts:



## Code samples

To make the tutorial easier to follow or if you get stuck somewhere, you can find code samples for each part here.

# Part 1: Setting up a basic C++ project using CMake and Make for building

## Step 1 - Create a working directory

Open a terminal, navigate to a suitable directory and run the following commands:

```
$ mkdir tutorial
$ cd tutorial
```

This will create an empty directory called `tutorial` inside which we'll be working.

## Step 2 - Add a minimal C++ program

Create a new file in `tutorial` called `main.cpp` and fill it with the following piece of code:

```
int main() {
    return 0;
}
```

The above is a small C++ program with a basic `main` function that doesn't do anything (for now).

## Step 3 - Add a CMakeLists.txt file

Create a new file in `tutorial` called `CMakeLists.txt`.

Set the minimum CMake version required to 3.13. While we don't need such a high CMake version right now, we'll need it in later parts of the tutorial.

```
cmake_minimum_required(VERSION 3.13 FATAL_ERROR)
```

Set project details, primarily the language to let CMake know that this is a C++ project.

```
project(tutorial VERSION 0.0.1 LANGUAGES CXX)
```

Add a new executable target with our main program above.

```
add_executable(tutorial
    main.cpp
)
```

Once you have followed the above instructions, you should have a CMakeLists.txt that looks like this:

```
cmake_minimum_required(VERSION 3.13 FATAL_ERROR)
project(tutorial VERSION 0.0.1 LANGUAGES CXX)

add_executable(tutorial
    main.cpp
)
```

## Step 4 - Build

Create a new build directory in `tutorial` for running any build commands. It is a good practice to follow so that any files produced as a result of building do not pollute the original project.

```
$ mkdir build
$ cd build
```

Run CMake to generate a Makefile.

```
$ cmake .. -DCMAKE_BUILD_TYPE=Release
```

Run Make to build the executable.

```
$ make
```

Check if everything above worked without errors.

## Step 5 - Run executable

There should be an executable named `tutorial` in the `build` directory. It does nothing for now, but we can still run it using

```
./tutorial
```

## Conclusion

In this part, we learnt how to create a new C++ project that uses CMake and Make for building. In the next part we will learn how to add the Marlin Multicast SDK to the project using CMake.

# Part 2: Adding the Marlin Multicast SDK to the project

## Step 1 - Add CMake module to find the SDK and download if not present

Create a folder called `cmake` in the `tutorial` directory by running the folling command from inside the `build` directory.

```
$ mkdir ../cmake
```

Create a file called `marlin-multicastsdk.cmake` inside the `cmake` folder with the following contents.

```
find_package(marlinMulticastSDK QUIET)
if(NOT marlinMulticastSDK_FOUND)
    message("-- marlinMulticastSDK not found. Using internal marlinMulticastSDK.")
    include(FetchContent)
    FetchContent_Declare(marlinMulticastSDK
        GIT_REPOSITORY https://gitlab.com/marlinprotocol/marlin.cpp.git
        GIT_TAG master
    )

    # Check if population has already been performed
    FetchContent_GetProperties(marlinMulticastSDK)
    string(TOLOWER "marlinMulticastSDK" lcName)
    if(NOT ${lcName}_POPULATED)
        # Fetch the content using previously declared details
        FetchContent_Populate(marlinMulticastSDK)

        # Bring the populated content into the build
        add_subdirectory(${${lcName}_SOURCE_DIR} ${${lcName}_BINARY_DIR} EXCLUDE_FROM_ALL)
    endif()
else()
    message("-- marlinMulticastSDK found. Using system marlinMulticastSDK.")
```

```
endif()
```

This snippet first checks if the SDK is installed already in the system or is downloaded already. If not, it uses the CMake FetchContent module to download the SDK from version control.

## Step 2 - Include the SDK in the build

Add the following line to `CMakeLists.txt`:

```
include("${CMAKE_CURRENT_LIST_DIR}/cmake/marlin-multicastsdk.cmake")
```

This will include the above module in the CMake build process.

## Step 3 - Link the SDK

To use the SDK in our code, we must first link it to our original executable.

```
target_link_libraries(tutorial PUBLIC marlin::multicastsdk)
```

Once you have followed the above instructions, you should have a CMakeLists.txt that looks like this:

```
cmake_minimum_required(VERSION 3.13 FATAL_ERROR)
project(tutorial VERSION 0.0.1 LANGUAGES CXX)

add_executable(tutorial
    main.cpp
)

# marlinMulticastSDK
include("${CMAKE_CURRENT_LIST_DIR}/cmake/marlin-multicastsdk.cmake")
target_link_libraries(tutorial PUBLIC marlin::multicastsdk)
```

## Step 4 - Build and run executable

Build using

```
$ cmake .. -DCMAKE_BUILD_TYPE=Release
$ make
```

and run using

```
$ ./tutorial
```

Building should have worked and the executable should run (and still do nothing for now).

## Conclusion

In this part, we learnt how to add the Marlin Multicast SDK to the project using CMake. In the next part we will learn how to use the Marlin Multicast SDK to connect to the Marlin Network.

# Part 3: Connecting to the Marlin Network

## Step 1 - Download and run a Goldfish node

For the purposes of this tutorial, we will use an executable called Goldfish that emulates the Marlin Network locally.

The executable is available here for Linux and here for macOS.

After downloading the executable to the `build` folder, open a new console and run it using

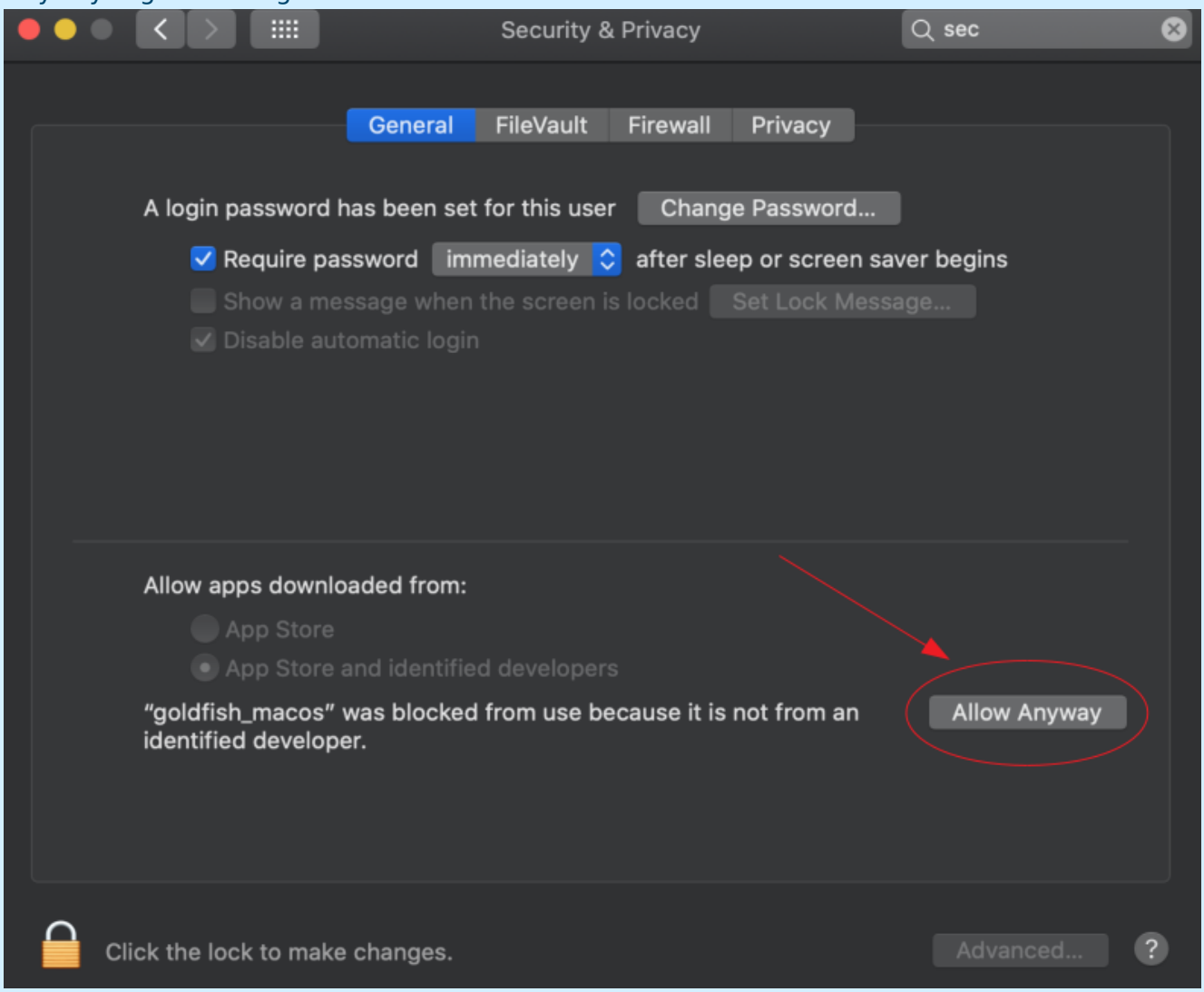
```
$ ./goldfish  # You might have given it a different name during download
```

You might have to change permissions on the file to make it executable using

```
$ chmod +x ./goldfish  # Add sudo in the beginning if necessary
```

You might get a verification error on macOS 10.15 that looks something like this:

To fix this, navigate to System Preferences -> Security & Privacy -> General and click "Allow Anyway" against the goldfish executable:




settings_allow

Image not found or type unknown

Check that it prints the IP and ports that it is using in the console.

## Step 2 - Set up a multicast client

Include the header for the multicast client and the relevant namespaces using

```
#include <marlin/multicast/DefaultMulticastClient.hpp>


using namespace marlin::multicast;
using namespace marlin::net;
```

Before we create the client, we need to create a delegate that is notified of important events by the client (e.g. new messages). See the reference documentation for further information on the events that can be listened to.

```
class MulticastDelegate {
public:
    void did_recv_message(
        DefaultMulticastClient<MulticastDelegate> &client,
        Buffer &&message,
        Buffer &&witness,
        std::string &channel,
        uint64_t message_id
    ) {}

    void did_subscribe(
        DefaultMulticastClient<MulticastDelegate> &client,
        std::string &channel
    ) {
        SPDLOG_INFO("Did subscribe to channel {}", channel);
    }

    void did_unsubscribe(
        DefaultMulticastClient<MulticastDelegate> &client,
        std::string &channel
    ) {
        SPDLOG_INFO("Did unsubscribe from channel {}", channel);
    }
};
```

Currently, the delegate ignores all messages and prints subscription/unsubscription logs.

Inside the main function, create a keypair that is used by the client for identification.

```
    uint8_t static_sk1[crypto_box_SECRETKEYBYTES];
    uint8_t static_pk1[crypto_box_PUBLICKEYBYTES];
    crypto_box_keypair(static_pk1, static_sk1);
```

Inside the main function, initialize a multicast client and delegate. The client can be customized by providing options during initialization. For further information on the available options, please refer

to the reference documentation.

```
    MulticastDelegate del;
    DefaultMulticastClientOptions clop1 {
        static_sk1,  // Secret key
        {"goldfish"},  // Pubsub channels
        "127.0.0.1:9002"  // Beacon address from the goldfish node
    };

    DefaultMulticastClient<MulticastDelegate> cl1(clop1);
    cl1.delegate = &del;
```

## Step 3 - Run the event loop

The Marlin Multicast SDK is fully asynchronous and uses event loops to acheive this.

Run the event loop using

```
return DefaultMulticastClient<MulticastDelegate>::run_event_loop();
```

Once you have followed the above instructions, you should have a `main.cpp` that looks like this:

```
#include <marlin/multicast/DefaultMulticastClient.hpp>



using namespace marlin::multicast;
using namespace marlin::net;

class MulticastDelegate {
public:
    void did_recv_message(
        DefaultMulticastClient<MulticastDelegate> &client,
        Buffer &&message,
        Buffer &&witness,
        std::string &channel,
        uint64_t message_id
    ) {}

    void did_subscribe(
```

```
        DefaultMulticastClient<MulticastDelegate> &client,
        std::string &channel
    ) {
        SPDLOG_INFO("Did subscribe to channel {}", channel);
    }

    void did_unsubscribe(
        DefaultMulticastClient<MulticastDelegate> &client,
        std::string &channel
    ) {
        SPDLOG_INFO("Did unsubscribe from channel {}", channel);
    }
};

int main() {
    uint8_t static_sk1[crypto_box_SECRETKEYBYTES];
    uint8_t static_pk1[crypto_box_PUBLICKEYBYTES];
    crypto_box_keypair(static_pk1, static_sk1);

    MulticastDelegate del;

    DefaultMulticastClientOptions clop1 {
        static_sk1,
        {"goldfish"},
        "127.0.0.1:9002"
    };

    DefaultMulticastClient<MulticastDelegate> cl1(clop1);
    cl1.delegate = &del;

    return DefaultMulticastClient<MulticastDelegate>::run_event_loop();
}
```

This sets up a multicast client that connects to the Marlin Network on `127.0.0.1:9002` can send and receive on the `goldfish` channel. The client automatically talks to the Goldfish node we ran above and sets up PubSub on the given channel. The client calls the corresponding functions in the delegate to notify it of any significant events.

## Step 4 - Build and run executable

Build using

```
$ cmake .. -DCMAKE_BUILD_TYPE=Release
$ make
```

and run using

```
$ ./tutorial
```

Building should have worked and the executable should run. You should see subscription related logs printed here indicating a successful connection between the client and the Marlin Network.

## Conclusion

In this part, we learnt how to use the Marlin Multicast SDK to connect to the Marlin Network. In the next part we will learn how to use the Marlin Multicast SDK to send and receive messages using the Marlin Network.

# Part 4: Sending and receiving messages using the Marlin Network

## Step 1 - Run two clients

Create second key pair.

```
    uint8_t static_sk2[crypto_box_SECRETKEYBYTES];
    uint8_t static_pk2[crypto_box_PUBLICKEYBYTES];
    crypto_box_keypair(static_pk2, static_sk2);
```

Add a second client inside `main.cpp`.

```
    DefaultMulticastClientOptions clop2 {
        static_sk2,
        {"goldfish"},
        "127.0.0.1:7002",
        "127.0.0.1:7000"
    };
    DefaultMulticastClient<MulticastDelegate> cl2(clop2);
    cl2.delegate = &del;
```

This time, we also set options that changes the ports used by the client since the default ports are taken up by the first client.

## Step 2 - Handle new messages

Handle new messages inside the `did_recv_message` function in the delegate. For now, we simply log the messages to the console.

```
        SPDLOG_INFO(
```

```
            "Did recv message: {}",
            std::string(message.data(), message.data() + message.size())
        );
```

## Step 3 - Send messages on some event

In real world scenarios, new messages are usually triggered by external events like timers, other messages, user actions, etc. For the purpose of this tutorial, we take advantage of the client calling `did_subscribe` to trigger a new message. Add the following to `did_subscribe`

```
client.ps.send_message_on_channel(channel, "Hello!", 6);
```

Once you have followed the above instructions, you should have a `main.cpp` that looks like this:

```
#include <marlin/multicast/DefaultMulticastClient.hpp>



using namespace marlin::multicast;
using namespace marlin::net;

class MulticastDelegate {
public:
    void did_recv_message(
        DefaultMulticastClient<MulticastDelegate> &client,
        Buffer &&message,
        Buffer &&witness,
        std::string &channel,
        uint64_t message_id
    ) {
        SPDLOG_INFO(
            "Did recv message: {}",
            std::string(message.data(), message.data() + message.size())
        );
    }

    void did_subscribe(
        DefaultMulticastClient<MulticastDelegate> &client,
        std::string &channel
    ) {
```

```
        SPDLOG_INFO("Did subscribe to channel {}", channel);
        client.ps.send_message_on_channel(channel, "Hello!", 6);
    }

    void did_unsubscribe(
        DefaultMulticastClient<MulticastDelegate> &client,
        std::string &channel
    ) {
        SPDLOG_INFO("Did unsubscribe from channel {}", channel);
    }
};

int main() {
    uint8_t static_sk1[crypto_box_SECRETKEYBYTES];
    uint8_t static_pk1[crypto_box_PUBLICKEYBYTES];
    crypto_box_keypair(static_pk1, static_sk1);

    uint8_t static_sk2[crypto_box_SECRETKEYBYTES];
    uint8_t static_pk2[crypto_box_PUBLICKEYBYTES];
    crypto_box_keypair(static_pk2, static_sk2);

    MulticastDelegate del;

    DefaultMulticastClientOptions clop2 {
        static_sk2,
        {"goldfish"},
        "127.0.0.1:9002",
        "127.0.0.1:7002",
        "127.0.0.1:7000"
    };
    DefaultMulticastClient<MulticastDelegate> cl2(clop2);
    cl2.delegate = &del;

    DefaultMulticastClientOptions clop1 {
        static_sk1,
        {"goldfish"},
        "127.0.0.1:9002"
    };
    DefaultMulticastClient<MulticastDelegate> cl1(clop1);
    cl1.delegate = &del;
```

```
    return DefaultMulticastClient<MulticastDelegate>::run_event_loop();
}
```

## Step 4 - Build and run executable

Build using

```
$ cmake .. -DCMAKE_BUILD_TYPE=Release
$ make
```

and run using

```
$ ./tutorial
```

Building should have worked and the executable should run. You should see subscription related logs printed here indicating a successful connection between both the clients and the Marlin Network. You should also see message logs indicating new messages sent and received using the Marlin Network both here and in the Goldfish node.

## Conclusion

In this last part, we learnt how to use the Marlin Multicast SDK to send and receive messages using the Marlin Network.