Quick Start

FruityMesh can be built using GCC and is mainly written in C, with some small C11 bits. Setting up the development environment has become pretty straightforward.

You have three possibilities:

- Flashing the precompiled firmware will take you about **5 minutes**.
- If you want to **compile it yourself** you need to setup the Toolchain with VsCode & Compile FruityMesh. This will take you a few more minutes until everything is ready to go.
- We also offer a **simulator** that allows you to do a lot of the development work on your computer instead of on the hardware itself. You can set up mesh networks of hundreds of nodes at the same time while being able to control and debug them all. Take a look at CherrySim for building and usage instructions.

Flashing Precompiled Firmware

The precompiled firmware can be flashed in three easy steps.

Download nRF5 Command Line Tools

The nRF Command Line Tools are handy for flashing the compiled firmware on a device. You can download them here for the operating system of your choice and install them:

https://www.nordicsemi.com/Software-and-tools/Development-Tools/nRF-Command-Line-Tools

The tools come with a bundled installation of the Segger Utilities, which are necessary for flashing and debugging nRF Hardware.

You should add the **bin/** folder of the installation to your operating system PATH variable to make development easier.

Flash SoftDevice & Application

Plug in your nrf52 development board via USB, then open a command prompt in the FruityMesh folder and type in the following command for flashing:

```
C:/<path_to_nrfjprog>/nrfjprog.exe --chiperase --program binary/fruitymesh_nrf52_s132.hex --reset
```

Your output should look like this:

```
Parsing hex file.
Erasing user available code and UICR flash areas.
Applying system reset.
Checking that the area to write is not protected.
Programming device.
```

If you have multiple boards attached, you will be prompted for the id of your board that should be flashed.

Next Steps

You're now ready to test FruityMesh. Continue by reading the Get Started section.

Compile FruityMesh

FruityMesh is built using the GCC compiler toolchain together with CMake and GNU make. The installation process varies a little for each operating system.

Installing the Toolchain

GNU ARM Embedded Toolchain

Download Version 4.9 of the GNU ARM Embedded Toolchain. This includes the compiler gcc, linker, and multiple utilities like objcopy, addr2line and gdb - the GNU debugger:

https://launchpad.net/gcc-arm-embedded/4.9/4.9-2015-q3-update/

Unzip the downloaded package into a folder of your choice.

Do not choose a newer version, as this might fail to compile.

Installing CMake

Download and install the latest Version of CMake:

https://cmake.org/download/

Tools Installation (Windows Only)

Next, if you are working on windows, you have to download and install some unix utilities, the GNU ARM Eclipse Build Tools. This includes *make*, *sh*, *rm* and some others. Version 2.4 has been tested successfully and the executable installer can be downloaded from here:

https://github.com/gnu-mcu-eclipse/windows-build-tools/releases/tag/v2.4b

Because most makefiles will also use the *mkdir* command and maybe some others, you should also download the GNU Coreutils that provide a set of UNIX commands in binary for Windows. Copy at least the `mkdir.exe` and `md5.exe` to the `bin/` folder of your GNU ARM Build Tools installation. You must also download the dependencies and copy these to the same folder (Two `.dll` files).

Binaries:

http://gnuwin32.sourceforge.net/downlinks/coreutils-bin-zip.php

Dependencies:

http://gnuwin32.sourceforge.net/downlinks/coreutils-dep-zip.php

Finally, you should add the bin folders of the gnu buildtools and the `bin` folder of `gcc-arm-embedded` to your PATH variable under System Environment. This will make sure that you can access these tools from anywhere.

Building FruityMesh

Now, you have a few options on how to build FruityMesh. We recommend using VsCode for building and coding with FruityMesh as it has a really nice CMake integration and can be set up in a short time.

Option 1: Using VsCode

If you do not have VsCode installed, get it for your platform from:

https://code.visualstudio.com/

Next, you must install some extensions:

- CMake Tools: https://marketplace.visualstudio.com/items?itemName=ms-vscode.cmake-tools
- C/C++ Tools: https://marketplace.visualstudio.com/items?itemName=ms-vscode.cpptools

Configure the Project

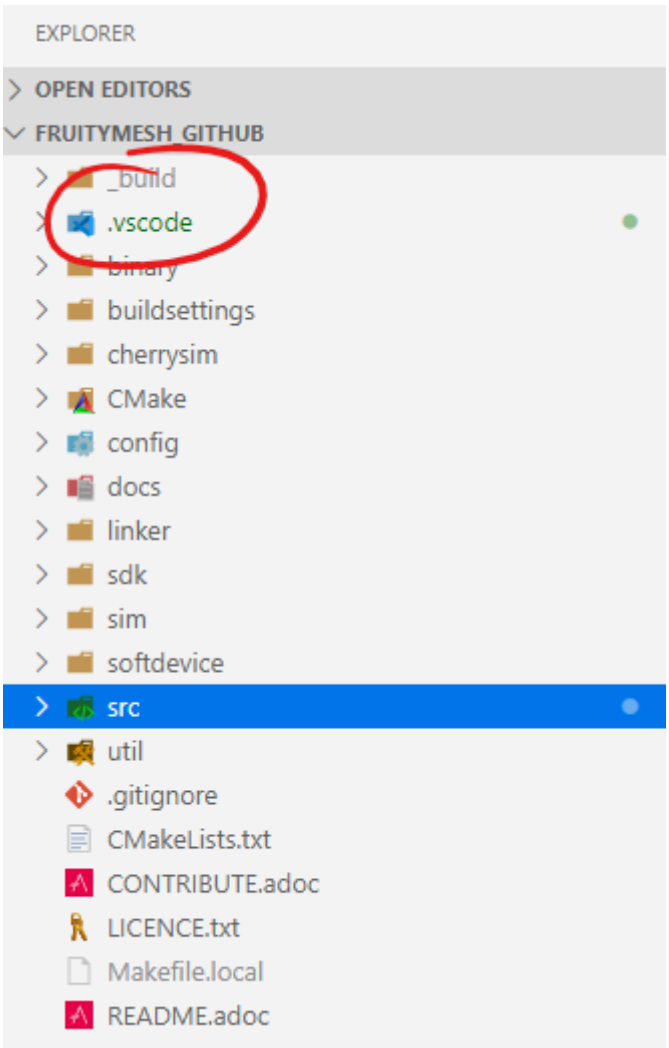
In order to configure the project, you should create the file `<fruitymesh>/.vscode/settings.json`. Create the folder if it does not yet exist and make sure to replace the path with the correct path that points to your GCC installation. (In case of problems see Troubleshooting)

settings.json

C++

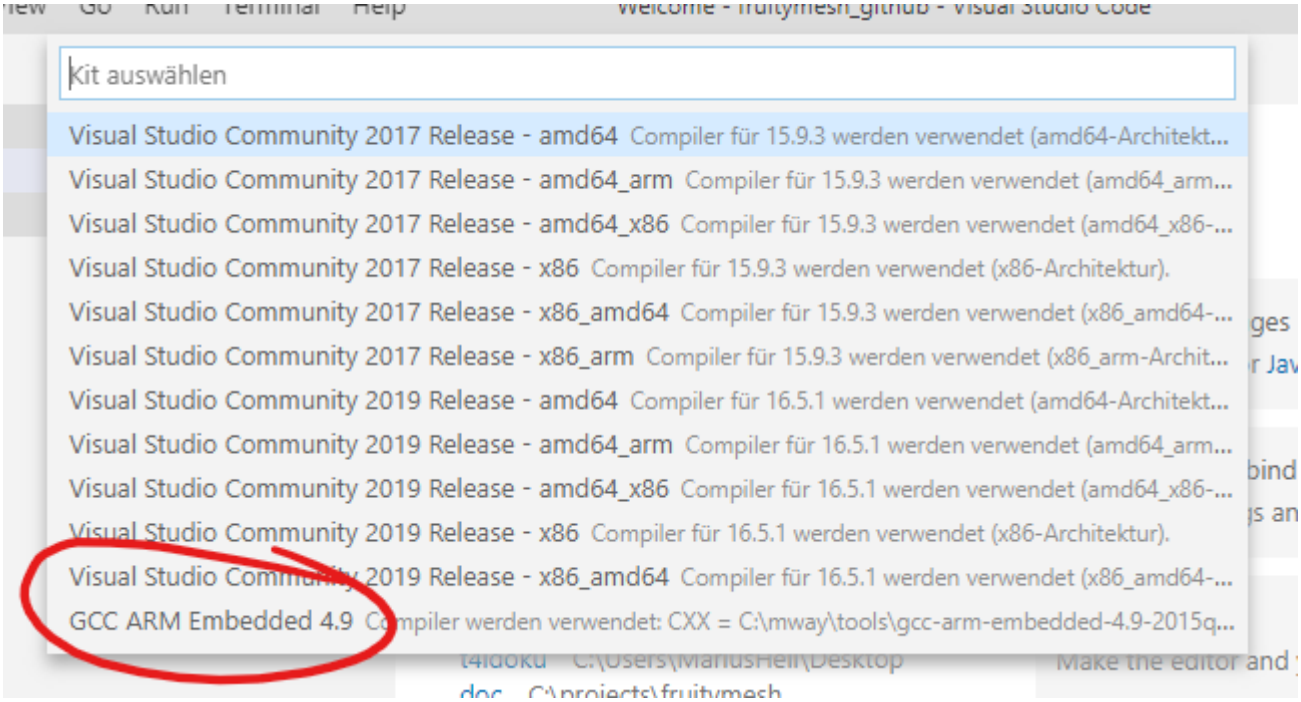
```
{
    "cmake.configureSettings":{
        "GCC_PATH":"C:/<your_path>/gcc-arm-embedded-4.9-2015q3",
        "BUILD_FEATURESETS":"ON"
    },
    "cmake.buildDirectory": "${workspaceFolder}/_build/vscode/",
    "cmake.configureOnOpen": true,
    "cmake.generator":"Unix Makefiles"
}
```

Now, if not already opened, you should open VsCode and use `File ⇒ Open Folder` to open the FruityMesh project folder. This should look similar to the following screenshot, with the `.vscode` folder as part of the repository.

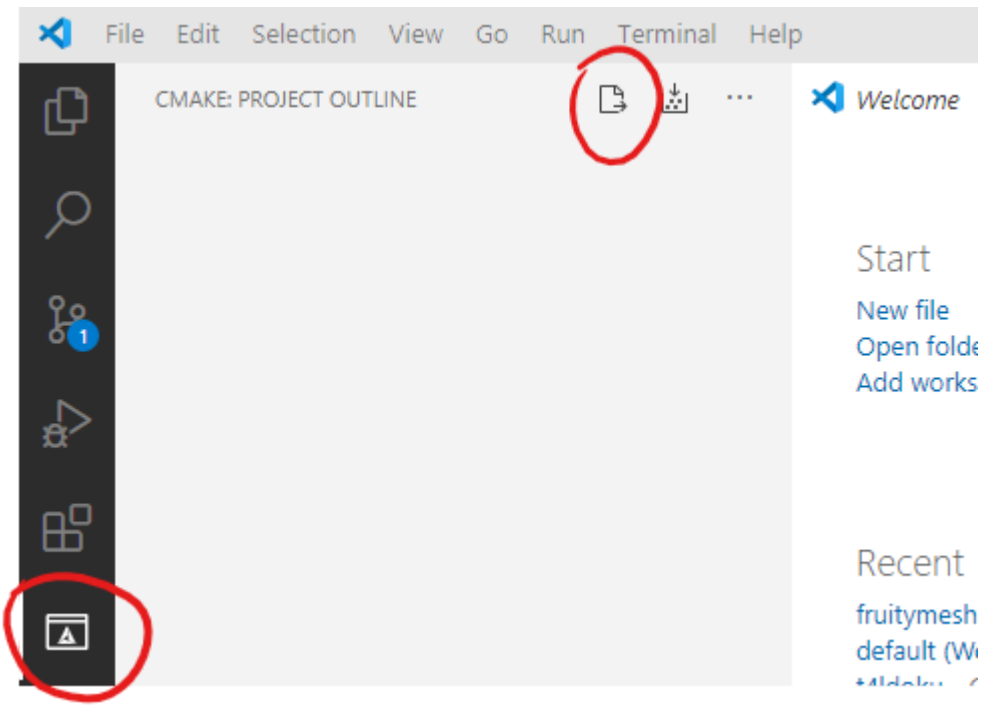


> If there are popups that ask you to allow Intellisense to be configured or if you want to use the compiler_commands.json file for configuring IntelliSense, click yes to have better indexing and code navigation support. This is mentioned up front as these popups might disappear fast. Don't worry, they will pop up again at some time.

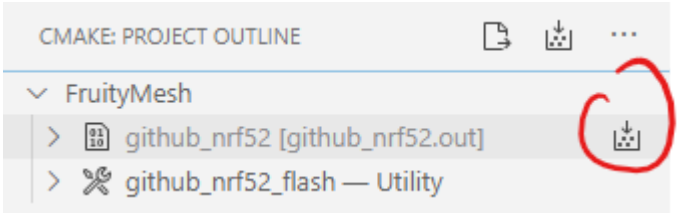
First, you have to select the Kit by clicking on "No Kit Selected" in the bottom bar of VsCode. You need to choose the installed GCC ARM Embedded 4.9 toolchain. If it does not show up in that list, make sure that you have added the gcc arm embedded directory to your path.



Next, switch to the CMake Panel on the left side and click "Configure".



This should automatically trigger the CMake configuration and load all available featuresets



You are now able to build the binary targets by clicking on the build button next to the featureset. There are also a number of Utility targets. If you right click a Utility target and choose "Run Utility", it will first build the target and then flash the application and SoftDevice on any attached development board.

For more information about VsCode, how to setup Debugging or for some Troubleshooting, make sure to also read the VsCode Setup page.

You can now continue with the Get Started section.

Option 2: Manual CMake Project Configuration

If you want to build FruityMesh on the command line, use the following instructions. Further explanation can be found under Building With CMake

Open a command line in `<fruitymesh>/_build/commandline` and execute the following command within that directory. Make sure to replace the path to GCC to fit your installation (use forward slashes "/"):

```
cmake "../../" -DBUILD_FEATURESETS=ON -DGCC_PATH="C:/<yourpath>/gcc-arm-none-eabi-4_9" -G "Unix Makefiles"
```

> The GCC_PATH must be specified using forward slashes "/", not backward slashes "\", otherwhise cmake will complain about `Invalid character escape '\m'` . You have to delete all files in your build directory before executing the command again to solve the issue! In case of other errors, make sure to have a look at Building With CMake

Next, from the same directory, execute the following command:

```
cmake --build . --target github_nrf52
```

Any other Featureset may be used as well as a target.

If you installed the nRF Command Line Tools and properly configured the `PATH` , you can now type:

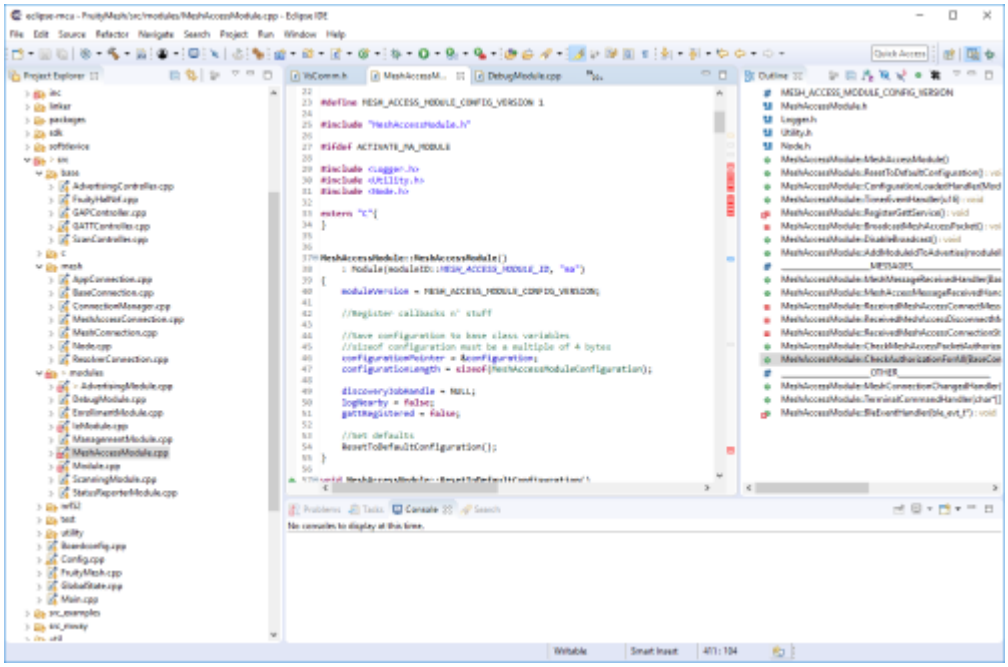
```
`cmake --build . --target github_nrf52_flash`
```

to flash the firmware on a device. This also works with any other Featureset by appending "_flash" to the name.

> The safest and most straight forward setup for flashing is to only have a single board connected to the computer while flashing.

You can now continue with the Get Started section.

Option 3: Using Eclipse For Development (Not recommended)



Eclipse is a good development IDE but its CMake integration is a bit outdated. The following setup will work, but you might experience issues with the code indexer. You should create a directory next to the fruitymesh directory, that you can call e.g. `fruitymesh_eclipse` . This is necessary as eclipse will otherwise not properly display the sourcecode directory in the project.

To generate the project settings, open a commandline in the `fruitymesh_eclipse` folder and execute:

```
cmake ../fruitymesh -DBUILD_FEATURESETS=ON -DGCC_PATH="C:/<yourpath>/gcc-arm-none-eabi-4_9" -G "Eclipse CDT4 - Unix Makefiles"
```

After starting Eclipse, all you have to do is to import the generated FruityMesh project. You can then develop, flash and debug in a comfortable way.

Get Started

Now, let's see how we can use FruityMesh. The precompiled firmware and the standard project settings are configured so that all devices immediately connect to each other. Start by plugging in your first development kit.

Open Serial Terminal & Connect

FruityMesh offers a Terminal to interact with the firmware. On Windows, PuTTY (http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html) is the best tool for this job. The screen utility can be used on macOS or Linux. You have to connect to UART using the following settings:

- **Serial line to connect to:** COMX (see blow)
- **Connection Type:** Serial
- **Speed:** 1000000
- **Data bits:** 8
- **Stop Bits:** 1
- **Parity:** None
- **Flow control:** RTS/CTS (Hardware)

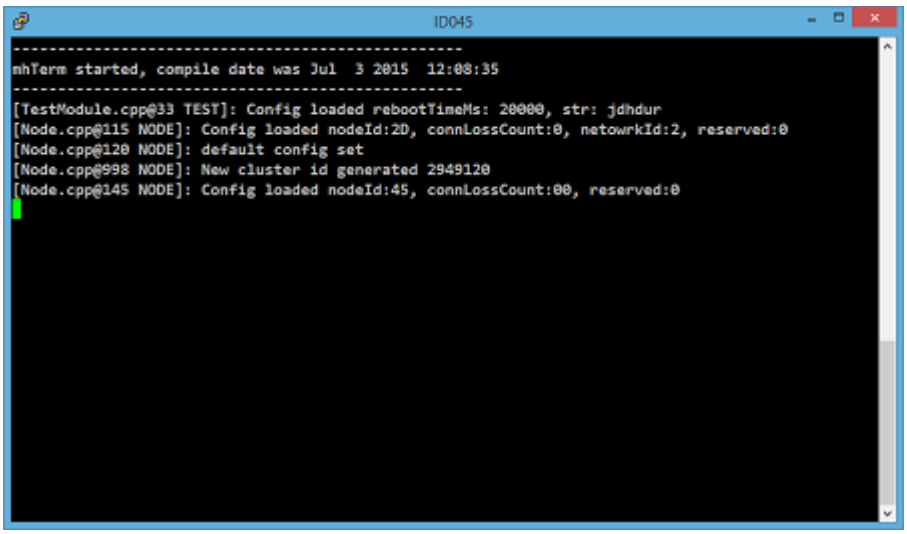
OSX users: To find out which serial port to open, you can list all devices under `/dev/cu.` and pick the one that says usbmodem.

On Windows you can find the correct COM port to connect to by opening the device manager and then under (COM & LPT) you should see a JLink entry with the COM number at the end (e.g. COM3). All the serial settings can be found in PuTTY under Connection/Serial.

You can also use the Segger RTT viewer to connect to the terminal of the node. Just open the viewer and select the correct settings while the debugger is connected to your computer.

Reset Development Kit

Once your terminal is connected to the serial port, press the reset button on the Development Kit and the Terminal should provide you with some output similar to this:



If you don't get output immediately it will sometimes help to disconnect the Devkit from USB for a short time or try to write something. This is an issue of the Segger Debugger chipset that bridges the UART.

Try Some Commands

You may now enter a number of commands to trigger actions. Here are some important ones:

- **status:** displays the status of the node and its connections
- **reset:** performs a system reset
- **data:** sends data through the mesh that other nodes then output to the terminal

Connect Second Development Kit

Next, flash and connect another node to the network and you should observe that they connect to each other after a short amount of time. You'll see that the LEDs will switch from blinking red to a single green pattern.

- If you enter the command **action 0 io led on**, both nodes should switch their led to white (all LEDs on). After you enter **action 0 io led off**, it will go back to connection signaling mode.
- Now, connect with another terminal to the second node and enter **data** in the command prompt and observe how the data is sent to the other node and outputted on the other terminal.
- You can add as many nodes as you like to the network and see how it reacts. If you remove a node, the network will try to repair this connection. You can observe the size change of the cluster by entering **status** from time to time.

> Two nodes will only connect to each other once they have been enrolled in the same network. The Github configuration will automatically have all nodes enrolled in the same network after flashing. If you do not want this, take a look at the UICR configuration.

GitHub Featureset

The default Featureset that is compiled for the github release is called github_nrf52. This featureset uses some default values from `Conf::LoadDefaults()` in `Config.cpp` and sets some other default values in `SetFeaturesetConfiguration_github_nrf52()` in `github_nrf52.cpp`. These defaults are useful to get you started quickly. To get your nodes into production you should however make use of the UICR to store a separate node key for each of your nodes. Also, nodes are typically enrolled by the user so they should not automatically connect to the same network after flashing. Also take a look at our Enrollment Module for more information on the enrollment of nodes.

Some of the defaults that are currently used for demonstration purpose:

- **Serial Number**: Auto generated in the FMxxx range (stays the same after re-flashing)
- **Node Key**: Set to 11:11:11:11:11:11:11:11:11:11:11:11:11:11:11:11
- **Network Key**: Set to 22:22:22:22:22:22:22:22:22:22:22:22:22:22:22:22
- **Network Id**: Set to 11
- **Enrollment State**: Set to true
- **BLE Address**: Uses the unique address of each chip

What's Next

Take a look at the Features page for a detailed overview of the possibilities and check out Basic Usage for usage instructions. If you're ready to contribute to the development of FruityMesh, cf. Developers for a roadmap and for instructions on how to participate.

If you want to start programming with FruityMesh, you should have a look at the Tutorials page for a guided introduction.