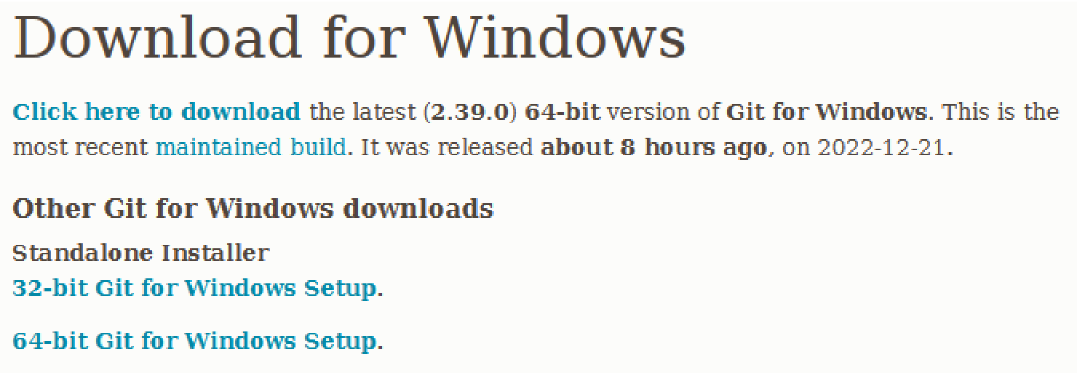
Firmware Setup

Installing Git

1. Download git for windows



Download for Windows

Click here to download the latest (**2.39.0**) **64-bit** version of **Git for Windows**. This is the most recent maintained build. It was released **about 8 hours ago**, on 2022-12-21.

Other Git for Windows downloads

Standalone Installer
32-bit Git for Windows Setup.

64-bit Git for Windows Setup.

2. Open Git Bash
3. Setup up global user credentials

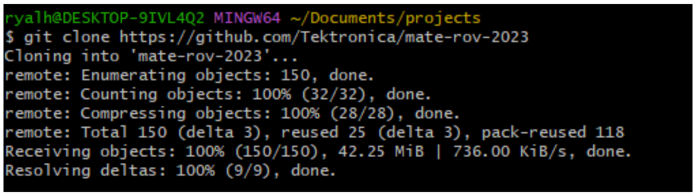
```
$  git config --global user.name “example”
$  git config --global user.email “example@gmail.com”
```

4. Create a new “projects” directory in your documents folder

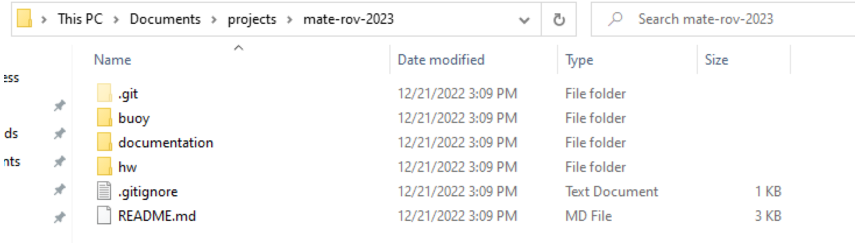
```
            $ cd Documents
~/Documents $ mkdir projects && projects
```

5. clone the remote repository from GitHub inside the projects folder

```
~/Documents/projects $ git clone https://github.com/Tektronica/mate-rove-2023
```



```
ryalh@DESKTOP-9IVL4Q2 MINGW64 ~/Documents/projects
$ git clone https://github.com/Tektronica/mate-rov-2023
Cloning into 'mate-rov-2023'...
remote: Enumerating objects: 150, done.
remote: Counting objects: 100% (32/32), done.
remote: Compressing objects: 100% (28/28), done.
remote: Total 150 (delta 3), reused 25 (delta 3), pack-reused 118
Receiving objects: 100% (150/150), 42.25 MiB | 736.00 KiB/s, done.
Resolving deltas: 100% (9/9), done.
```



Name	Date modified	Type	Size
.git	12/21/2022 3:09 PM	File folder	
buoy	12/21/2022 3:09 PM	File folder	
documentation	12/21/2022 3:09 PM	File folder	
hw	12/21/2022 3:09 PM	File folder	
.gitignore	12/21/2022 3:09 PM	Text Document	1 KB
README.md	12/21/2022 3:09 PM	MD File	3 KB

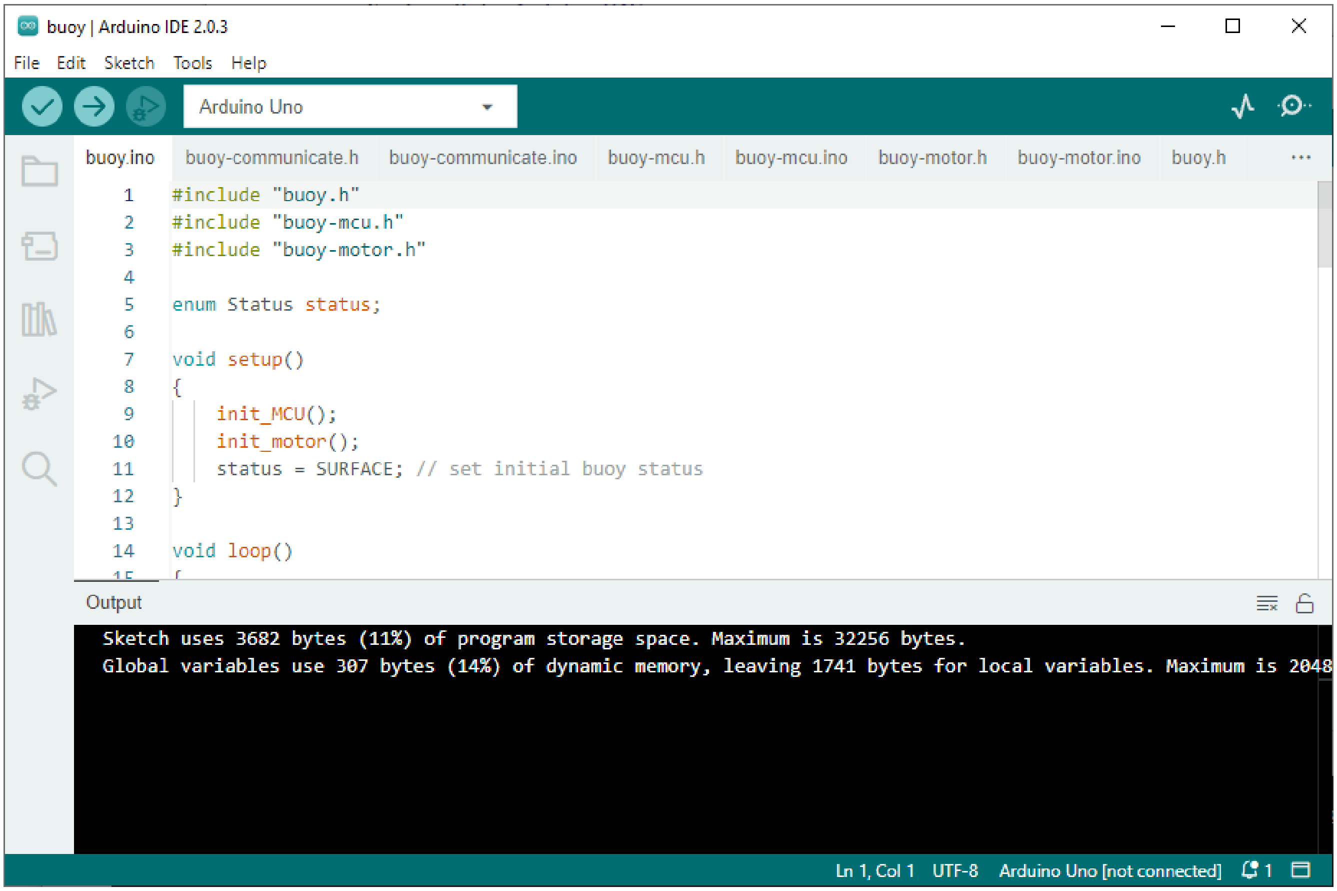
Opening Project

6. Install the [Arduino IDE]

7. Open the IDE

8. open `buoy.ino`

9. Verify the code by clicking the checkmark icon in the top left of the interface



```
#include "buoy.h"
#include "buoy-mcu.h"
#include "buoy-motor.h"

enum Status status;

void setup()
{
    init_MCU();
    init_motor();
    status = SURFACE; // set initial buoy status
}

void loop()
{
```

```
Sketch uses 3682 bytes (11%) of program storage space. Maximum is 32256 bytes.
Global variables use 307 bytes (14%) of dynamic memory, leaving 1741 bytes for local variables. Maximum is 2048
```

What is a Sketch File?

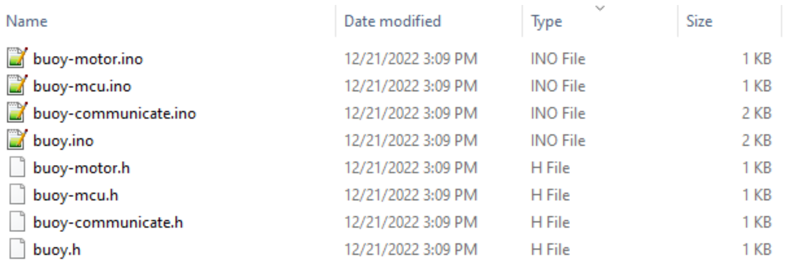
An Arduino project consists of one or more Sketch files which get compiled and flashed onto your Arduino. The sketch file is denoted by the extension, ".ino", which contains the source code written in the Arduino programming language. The language is more or less built on top of the C / C++ language. Consequently, a lot of the familiar coding paradigms are translated over to your Arduino project. However, there is a notable difference:

A minimal Arduino C/C++ program consists of only two functions:

- **setup():** This function is called once when a sketch starts after power-up or reset. It is used to initialize variables, input and output pin modes, and other libraries needed in the sketch. It is analogous to the function main().
- **loop():** After setup() function exits (ends), the loop() function is executed repeatedly in the main program. It controls the board until the board is powered off or is reset. It is analogous to the function while(1).

File Structure

At the time of writing, the `buoy` project folder consists of eight (8) files. Four Arduino Sketch files make up the entire source code while four additional "header" files accompany these sketch files. The header file has the extension ".h" which contains C function declarations to be shared between several source files. The main project file is buoy.ino, which is the application layer of our code. More on that later.



Name	Date modified	Type	Size
buoy-motor.ino	12/21/2022 3:09 PM	INO File	1 KB
buoy-mcu.ino	12/21/2022 3:09 PM	INO File	1 KB
buoy-communicate.ino	12/21/2022 3:09 PM	INO File	2 KB
buoy.ino	12/21/2022 3:09 PM	INO File	2 KB
buoy-motor.h	12/21/2022 3:09 PM	H File	1 KB
buoy-mcu.h	12/21/2022 3:09 PM	H File	1 KB
buoy-communicate.h	12/21/2022 3:09 PM	H File	1 KB
buoy.h	12/21/2022 3:09 PM	H File	1 KB

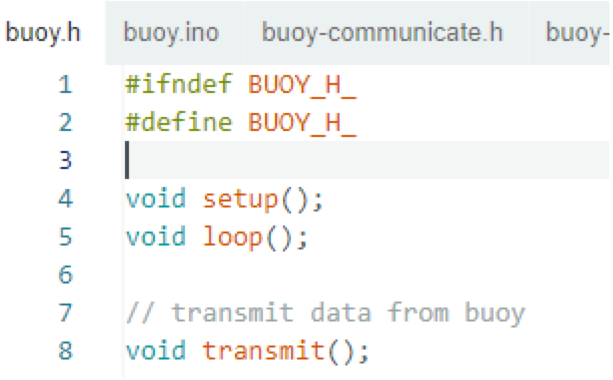
Filename	File Type
buoy.ino	Arduino Sketch file
buoy-mcu.ino	Arduino Sketch file
buoy-motor.ino	Arduino Sketch file
buoy-communication.ino	Arduino Sketch file
buoy.h	Header file
buoy-mcu.h	Header file
buoy-motor.h	Header file
buoy-communication.h	Header file

Header Files… Briefly

Header files fundamentally are no different than source files. In fact, a header can contain the same code constructs as in the ".ino" source file. By convention, however, the header file distinguishes it should be #included by other files, which isn't typically done with source files.

In essence, header files are like baseball cards. We don't necessarily want to trade around the actual players; only their *playing cards*. So, just like a baseball card, the header file doesn't contain any functionality, but it declares function names… kinda like a player's batting average, the name, and perhaps their favorite color.

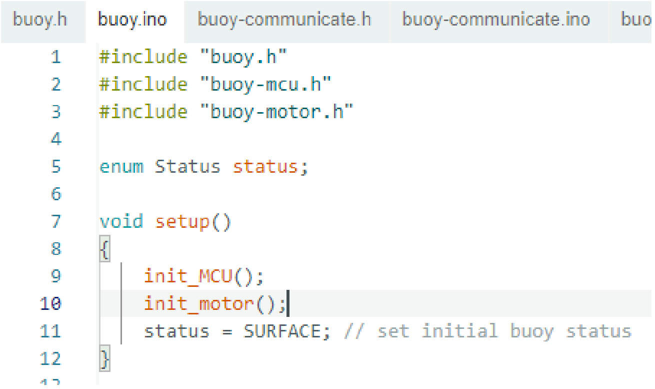
So, why use header files? Well, that's tricky to answer since Arduino doesn't really require header files for source files! Only Libraries. The Arduino compiler knows to link all the individual ".ino" source files without headers. However, the use of headers is a convention taken from C/C++ for larger multi-file projects. I include them in the project for completeness.



```
#ifndef BUOY_H_
#define BUOY_H_

void setup();
void loop();

// transmit data from buoy
void transmit();
```



```
#include "buoy.h"
#include "buoy-mcu.h"
#include "buoy-motor.h"

enum Status status;

void setup()
{
    init_MCU();
    init_motor();
    status = SURFACE; // set initial buoy status
}
```

Header file
buoy.h

Arduino source file
buoy.ino

You may have noticed the use of `ifndef` and `endif` keywords. These are conditional preprocessor blocks that check whether the header file by the ID "`BUOY_H_`" has already been loaded. `ifndef` means **if n**ot **def**ined, therefore this block will 'execute' if and only if `BUOY_H_` is not defined.

Firmware Design Philosophy

There are many ways to write, package, and deploy code. However, consistency across projects is a key element to ensuring any project is robustly maintainable. This maintainability can be considered a set of coding "best-practices", or patterns, which allow anyone to review your code effortlessly without having to dig too deep into your own "best practices" that no one seems to understand.

Modularity is also an important features of larger projects. Many Arduino projects are plagued by bad coding practices packed up inside a behemoth single source file. So, it's up to us to ensure we don't fall into this common habit and consistently check whether our code is well-structured. After all, who wants to read spaghetti code? I sure don't, at least.

Firmware Organization

As already mentioned, the project consists of eight files. However, let's just forget about the header files for now. The source files were written with the intention to be used as components. Much like an orchestra, each file has a job and as programmers, we want to assembly these files as modules we can easily maintain in our ensemble. The project is organized into three categories:

Layer	Analogy	Associated Files
Application	This is our musical conductor. This layer doesn't care who is playing the instruments, just whether everything is playing in the right order. This is the heart of our project.	buoy.ino
Peripheral	These are our instruments. We want to wire up RF modules, measurement sensors, and motors here!	buoy-motor.ino buoy-communication.ino
MCU	This is our instrument player. The performer that makes sure everything actually happens when the application wants it too. The Arduino is our MCU layer and it controls all the peripherals and its own board.	buoy-mcu.ino