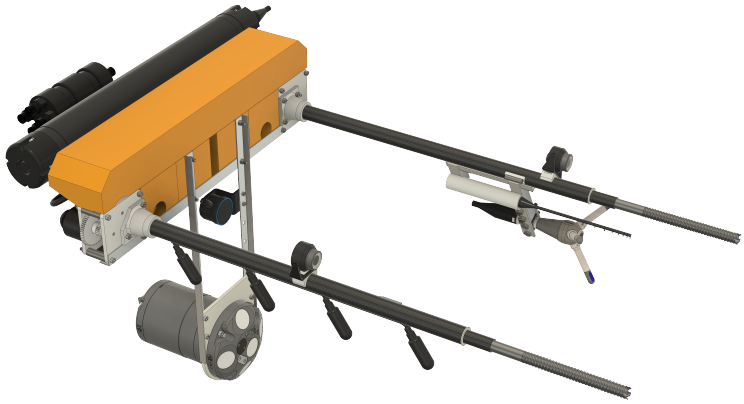
MeltStakeCommander

Sam Felsted

January 31, 2025



Contents

1 Requirements

1.1 Setup

This project was programmed in Java and thus the JVM is required to run the application. The .Jar file should be able to run on any operating system as long as the JVM is installed, however the exe wrapper is specified for Windows.

1.2 Software

The following software is required to build and update the application:

- Java [1]
- IDE for writing java code [2]
- Maven

[1] https://www.java.com/en/
[2] I personally use IntelliJ Idea https://www.jetbrains.com/idea/

2 Operation

Using the GUI application should be relatively intuitive, however there are some minor quirks to operation. Sometimes the application won't connect if the app was opened before the ROV was turned on. As is standard in programming, simply opening and closing the app fixes this issue.

2.1 Directory

Located on the desktop of the ROV laptop should be the MeltStakeCommander directory (This file should also be in that directory). Inside of that directory is an .exe file, settings.yaml, and a log file for the exe launcher. It is critical that the .exe file lives in the same directory as the settings.yaml as that is how the application loads data.

2.2 Settings

There are many configurable settings of the MeltStakeCommander app. Below is a table designed to familiarize yourself with the settings and their expected values.

Parameter	Type	Default	Description
ip	IP address (IPv4)	192.168.2.50	This is the IP address of the ROV raspberry pi used for networking to the beacon
antenna	Integer (ID)	XXX	This is the acoustic beacon id of the MeltStake
portOUT	Integer (PORT)	4000	This is the port that the client uses to SEND messages to the beacon
portIN	Integer (PORT)	4001	This is the port that the client LISTENS for messages on
pingRate	Integer (MS)	60000	Periodically the client sends a DATA ROT command to the meltstake to ensure data is flowing to the ROV and in theory the MeltStake. It must be noted the connection status may say connected if it has communication to the ROV but not necessarily the MeltStake.

In addition to the above parameters there is a button to control the AutoRelease safety feature on the MeltStake. The AutoRelease feature is an internal timer on the MeltStake that resets with every drill command or manual reset command. The timer is set for 5 minutes and at the end of the timer the MelStake will automatically run the release command. The release command will attempt to reverse the drills and eject itself from the glacier if it detects an underwater environment. This is useful in the event we lose communication with the MeltStake and need to recover it.

2.3 Usage

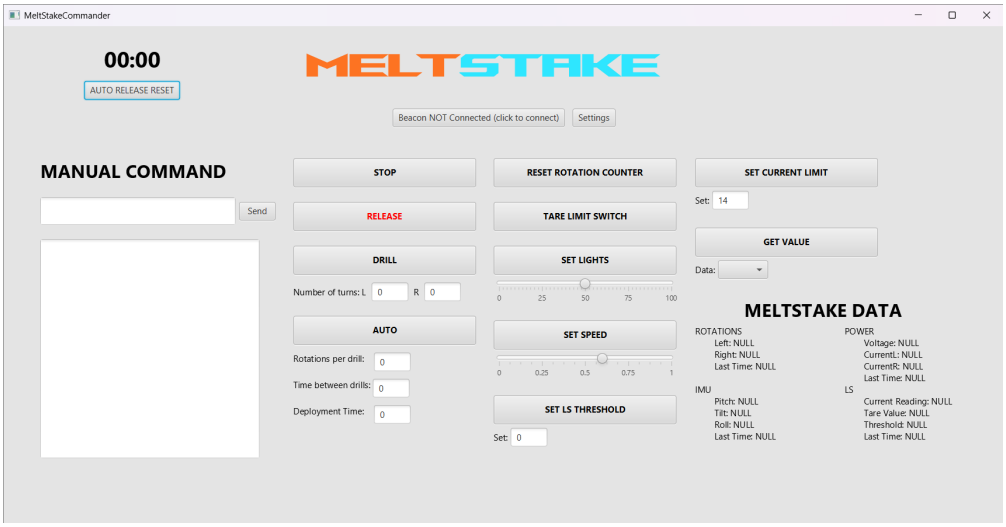


Figure 1: GUI

The MeltStakeCommander Application implements 2 priority queues to send and receive messages. Clicking a command button or manually entering a command will add this command to the outgoing queue. Because of the nature of the acoustic beacons, users are strongly discouraged from spam clicking buttons. Hovering over a button will lead to a popup tooltip to better explain the purpose of the command.

In addition to the command buttons, there is a section to type out commands. commands will follow the following format:

```
[BEACON ID] [COMMAND] [PARAMETERS]
202 DATA IV
202 DRILL 1 0
```

The full list of commands can be found here: https://github.com/noahaosman/MeltStake/blob/main/Operations.py

Hitting send or pressing enter will clear the commandline and add the command to the queue. Below the manual command is the log of the communication between the client and the MeltStake.

3 Programming

The MeltStakeCommander application is primary written in Java with the JavaFX library. Java is an object-oriented programming language that compiles into a .jar file. This is an all encompassing file that contains everything needed to run the application.

3.1 Compiling and Building

The .jar file for the application created using Maven, a java compiler suit, and Shade a Maven plugin to wrap all needed libraries into the Jar . To compile the project into a .jar file, use the Maven Package command. It is best practice to run Maven Clean before this, however.

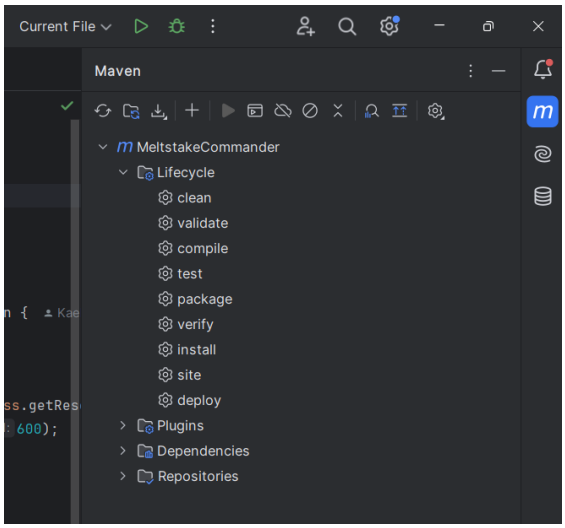


Figure 2: GUI of Maven operations in IntelliJ

The final build of this project is compiled again into an .exe file using the launch4j toolkit. Inside the project directly lives a launch4j.xml file that can be opened to preconfigure all necessary settings. To create an .exe from the .jar file, click the gear button. To test run the .exe file, hit the blue play button.

An example compilation process would look like this

- Changes are made to the application
- The project is compiled with maven package (the programmer will click this process in the maven sub menu to run)
- The user opens launch4j
- The settings for launch4j are imported from the project directory
- The jar is wrapped into an exe
- The exe is placed into its directory on the desktop folder

3.2 Project Structure

```
/MeltstakeCommander
    /src
        /main
            /com.meltstakecommander
                *java files*
            /resources
                *fxml files*
                *images/icons*
    /target
        *maven compile outputs*
        MeltstakeCommander-X.X.jar
    readme.md
    settings.yml
    pom.xml
    launch4j.xml
    mvnw
```

The Meltstake commander application follows the standard java template for file organization. Most interactions will be in the /src/main/com.meltstake commander directory for writing code and /target for compiled jar files to turn into executables.

3.3 Architecture

The Main.java file is purely a driver for the Application.java file. The Application.java file is a driver for the GUI code. The reason for this structure is because of the Shade plugin. This allows all libraries to be wrapped nicely in a single .jar file. The front end of the app is written in FXML, a scripting language used for javaFX. The backend is controlled in the Controller.java. This has 2 helper classes, SettingsLoader.java and Client.java, that handle parts of the app for the controller. The Client.java is responsible for all networking related actions of the application. The SettingsLoader is primarily meant to interface between the loaded settings and the settings.yml used for the saved settings.

The likely first place to start when adding new features onto the GUI is the Controller.java file. This is the backend of the GUI written in FXML and handles the main thread of the application. Much of the code are functions marked with the @FXML tag to indicate it interacts with the GUI code directly. The GUI itself is in the resources folder and is named view.fxml. In the fxml file, you can see that buttons and other objects interface with the .java file using the names of the relevant functions. This system allows for the backend to be seperated from the front end.

JavaFX has great documentation found here: https://openjfx.io/

FXML also has documentation found here: https://openjfx.io/javadoc/23/javafx.fxml/javafx/fxml/doc-files/introduction_to_fxml.html

4 Networking

Understanding how the application communicates to the actual MeltStake is critical for debugging and development of the commander application. This is a complex system with multiple communication interfaces. The purpose of this section is to give a high level overview of the communication chain to give insight into the process. The general chain of a message from the application to the meltstake is as follows:

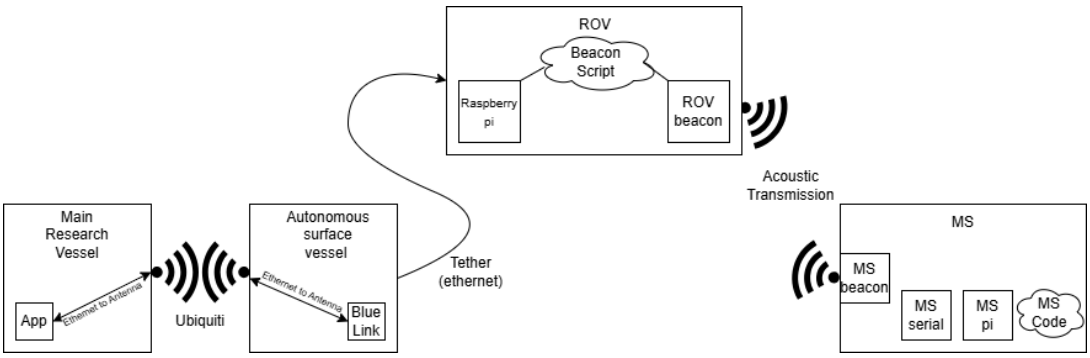


Figure 3: Meltstake interfacing

The application runs on the main research vessel and communicates to the autonomous research drone using Ubiquity. Ubiquity is a networking server and thus both parties in communication are ethernet connected to antennas to communicate. From the autonomous vessel, information is then routed through BlueLink, the box that communicates to the ROV. This connection is the orange tether cable (which is just an insulated ethernet cable). Then communication is routed into the ROV's raspberry pi. The PI is running a server script called **Beacon** which is used for relaying data from the ROV to the Meltstake. Physically, the ROV then has a serial port which connects to an acoustic antenna (also known as a beacon). The server script sends communication to the receiving antenna on the meltstake which then uses serial connection to connect to the Meltstake PI.

However, in this application's code, this is all abstracted and networking to the meltstake is as simple as opening a network socket to the ROV's IP address.

4.1 Application

On the MeltStakeCommander application, the client.java class handles all networking involved in the application. As previously mentioned, the incoming and outgoing network traffic is stored in two priority queues. These are internally managed by the application in a thread for each of them. Outgoing and incoming client networking interfaces with the ROV raspberry pi over network sockets.

4.2 Application to Meltstake

In the outgoing networking, the MeltstakeCommander app acts as if it is a client to the ROV server (beacon). When a command is registered, it is added to the queue. The outgoing thread dequeues the command as a string and opens a connection to the beacon server. The command is then sent and the connection is severed after each command transmission. The beacon notes the IP of the client and uses that for incoming network communication. The network address of the socket is

the IP of the ROV and the port is set to 4000. This is a setting on the client but is hardcoded into the beacon code.

4.3 Meltstake to Application

For incoming network traffic, the MeltstakeCommander is now the host and the beacon acts as the client. This operation is on port 4001. Similarly, this is a setting in the app, and it is hardcoded on beacon. When a connection is open, a thread in the application adds received information into an input queue and then closes the connection. The main thread of the application handles the input queue.

5 Troubleshooting

As with all software, there are simply no flaws within the application. However, the following strictly hypothetical examples are provided

5.1 Application says no connection

First, check to make sure the laptop is connected to the correct network. If using ethernet, ensure that Wi-Fi is turned off to avoid attempting to connect to the internet. Ensure that the IP and both port settings are correct.

Sometimes when the application is launched before connecting it can get into a stuck state. The cause of this is unknown but simply opening and closing the app fixes this error.

5.2 The application is not launching

Ensure that the settings.yml is in the same directory from were the application was launched. This can cause a crash on the execution of the application.

5.3 Some commands are not running

It is very possible for the commands to be dropped through the acoustic communication. Acoustic communication is inherently not the most reliable. This is why commands that are state based such as the auto release override wait for a received confirmation before changing the clients state. In addition, avoid sending commands back to back as the acoustics can get overloaded. The best practice is to wait a seconds in between commands.