# Unity class explanation

To understand the rest of this document, I will need to explain a few Unity concepts used here.

## Monobehaviour

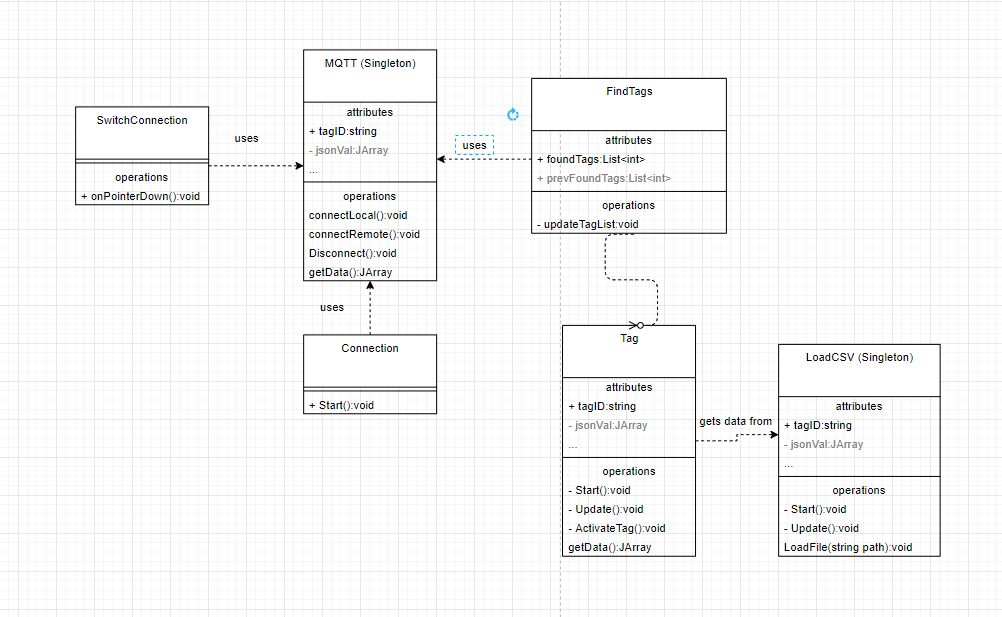
C# base class you need to inherit from, if you intend to use your class in a unity scene. This means each “Monobehaviour” class you see in this document is tied to at least one Gameobject in the scene.

The most important thing to know about this class, in the context of this document, is that it allows you to run code in each rendered frame, and react to UI events.

# MQTT Connection

This part of the program handles the MQTT connection of the program, and getting data from it.

Overall: Connection itself seems to work fine, actual connection with the hardware needs to be tested but if there are bugs, they should be easily fixed.



Here you see all the relations between classes. Some oddities here are for example, the Tag class never actually getting any of the Json data from the MQTT class, meaning the Tag never actually updates from MQTT data.

Another odditiy here is that the FindTags class creates instances of the Tag class, but never actually saves them, meaning the FindTags can’t access them anymore after creating. This is a weird Unity quirk of which I’m not sure how to model it in UML.

Written below are some explanations of all the classes:

## Connection:

Class that only enables the MQTT remote connection. Should be deleted since SwitchConnection also handles this.

## FindTags:

Reads json files from the MQTT broker, containing tag data.

Tag id is saved here as int, and when a tag is detected a new Tag Monobehaviour is created.

## MQTT:

Singleton that Connects to either the local or remote broker. Seems to work as intended, however will need to be tested once the hardware is available.

**Note: IP address and topic are hard coded in this class. We will need to edit those for our own setup.**

## SwitchConnection:

Button logic to switch between Local and Remote MQTT broker. Not much to say about this.

# Tag Sensor code

## Tag

The Tag Monobehaviour handles the data of a tag, it’s a UI element, and the drawing of the tag object in the scene.

This class reads data from the loaded CSV file reads at a specific timestamp in the file, based on the UI slider value on the bottom of the screen. (From the Slider Object on the ReplaySlider GameObject).

It currently does not get data from MQTT, even though a tag object gets made using MQTT. Based on this the intention was probably that MQTT data was meant to be implemented here.

For some reason, all Gameobjects are children of the Canvas Gameobject, meaning the tag 3d objects in the scene are part of the UI. This should definitely be changed.

## CSV File reading

The logged tag data is read from a csv file, of which the values get read hard coded by array positions in the C program.

For now, to make it easier to work with this, these are the values of each array element:

0: line number

1: Timestamp : float

2: Sensor : int

3: Succes :Boolean

4: PosX : float

5: PoxY : float

6: Posz : float

7: AccelX : float

## Conclusion

Reading CSV files and video files work, it is quite nicely visualized using a Line renderer, and this code is definitely reusable for us. It does however need work to implement live trackers using MQTT and has a bunch of its logic hidden in UI code.

# Missing features

Based on the code and the UI, it is clear that the following features were intended to be implemented but are either unfinished or still bugged.

Currently the following features are either unfinished or do not work correctly:

* Reading Tag data from MQTT instead of CSV
* Automatic playback of CSV data, currently you have to scroll the slider manually
* Playback of multiple CSV files at the same time in Compare mode, you can swap between the two CSV files but you can’t watch both at the same time

The following parts of the code should probably be improved:

* Reading CSV file, currently hardcoded to array positions, annoying for future developers
* Separate program logic and UI logic, currently the UI slider also implements CSV/video playback, etc
* UI has a few bugs, like the sliders being able to move when they shouldn’t be able to, or the pause button flipping when playback of a video is finished
* Change the File explorer to using Windows API instead of an external library, it’s currently throwing errors in the editor and the UI for it is bad