Design document

Uwb positioning area 51



**Table of contents**

[**Requirements**](#_m1rs7mlugub) **3**

[Must have](#_bk6mfncz9d94) 3

[Should have](#_rl8qb1eustqx) 3

[Could have](#_f71fj9lnj9fz) 3

[Won’t have](#_t030lhjjhjy5) 3

[**Class diagram**](#_k39m6fjkhytf) **4**

[Version 0.1](#_uyw9lltlzl5n) 4

[Version 0.2](#_n2521iojy749) 5

[Version 0.3](#_jio7bq4zgi6y) 6

[Version 0.4](#_yr7kkzk3xd2g) 7

[Version 0.5](#_66sjqttiju3a) 7

[**User interface**](#_pbdi05pcxmdw) **8**

[Version 0.1](#_b5rbmu7rpsia) 8

[Version 0.2](#_d2ike37lpojo) 8

# Requirements

The team has identified a range of requirements for the end product. These requirements are categorized using the MoSCoW system. This splits the requirements into “**Must have**”, “**Should have**”, “**Could have**” and “**Won’t have**”.

## Must have

* See the speed/Acceleration/Height/etc. (i.e relevant available data) at current timestamp
* Be able to connect 1 MQTT server for receiving pozyx data.
* Developers should be able to easily add more protocols and/or file formats for importing/exporting (live or regular) data
* for the SD6 course
  + Automatic unit testing
  + Automatically published (working) executables for windows
  + Automatically published (working) executables for android
* Export data to the following formats:
  + JSON and CSV
* Import data from the following format(s)/protocol(s):
  + JSON
* Record measurement data for 1 user per recording device

## Should have

* Offset the measurement data in time
* Offset the (positional) measurement data in space
* Be able to connect to more than 1 sensor system for receiving data
* Measure the average speed over a period of time
* User can select a timestamp and the system provides data at that timestamp (e.g. speed, position)

## Could have

* Save and load user settings

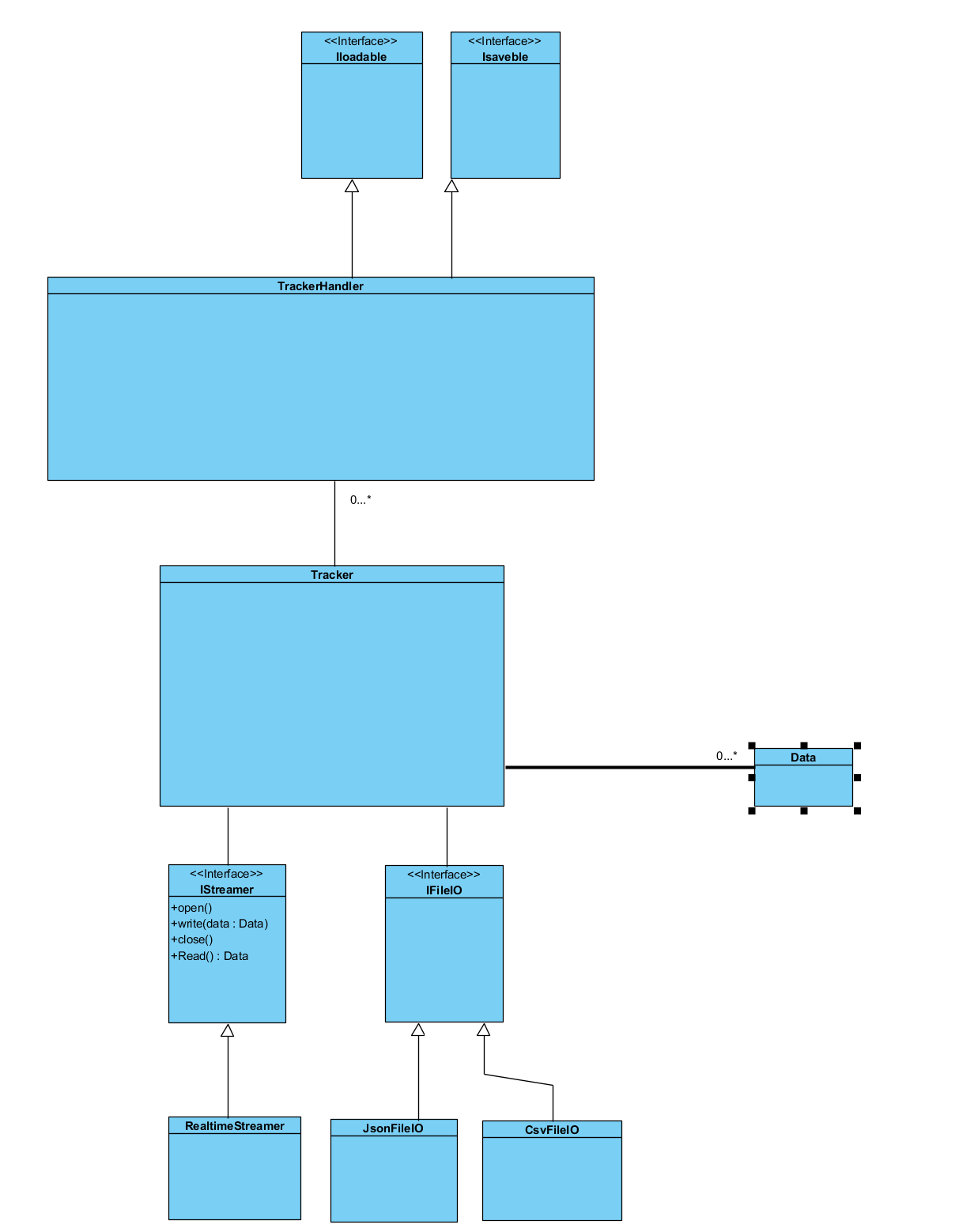
## Won’t have

* Finalized user interface
* Data synchronisation between program instances.
* ability to connect to more than 1 sensor system (though the possibility to do so should be in the design)
* The new 3d view of the park
* Video support

# Class diagram

The team has iteratively developed on their class diagrams. To show this iterative process, multiple versions of the class diagrams will be discussed in the following chapters.

## Version 0.1



This is the first iteration of the design. In this iteration, the main layout of the design (also seen in the next iterations of it) was determined.

The biggest design choice that was made in this step is that all sensor data comes in through a streamer class, which works similarly to Input/Output streams in C, though instead of streaming bytes directly, it gives a stream of a predetermined data-type (e.g. pozyx sensor data, heart rate data, etc.).

This allows future developers to easily add more data input types, as they only need to implement an interface. This design has some issues (apart from not being finished), which are fixed in the next design.

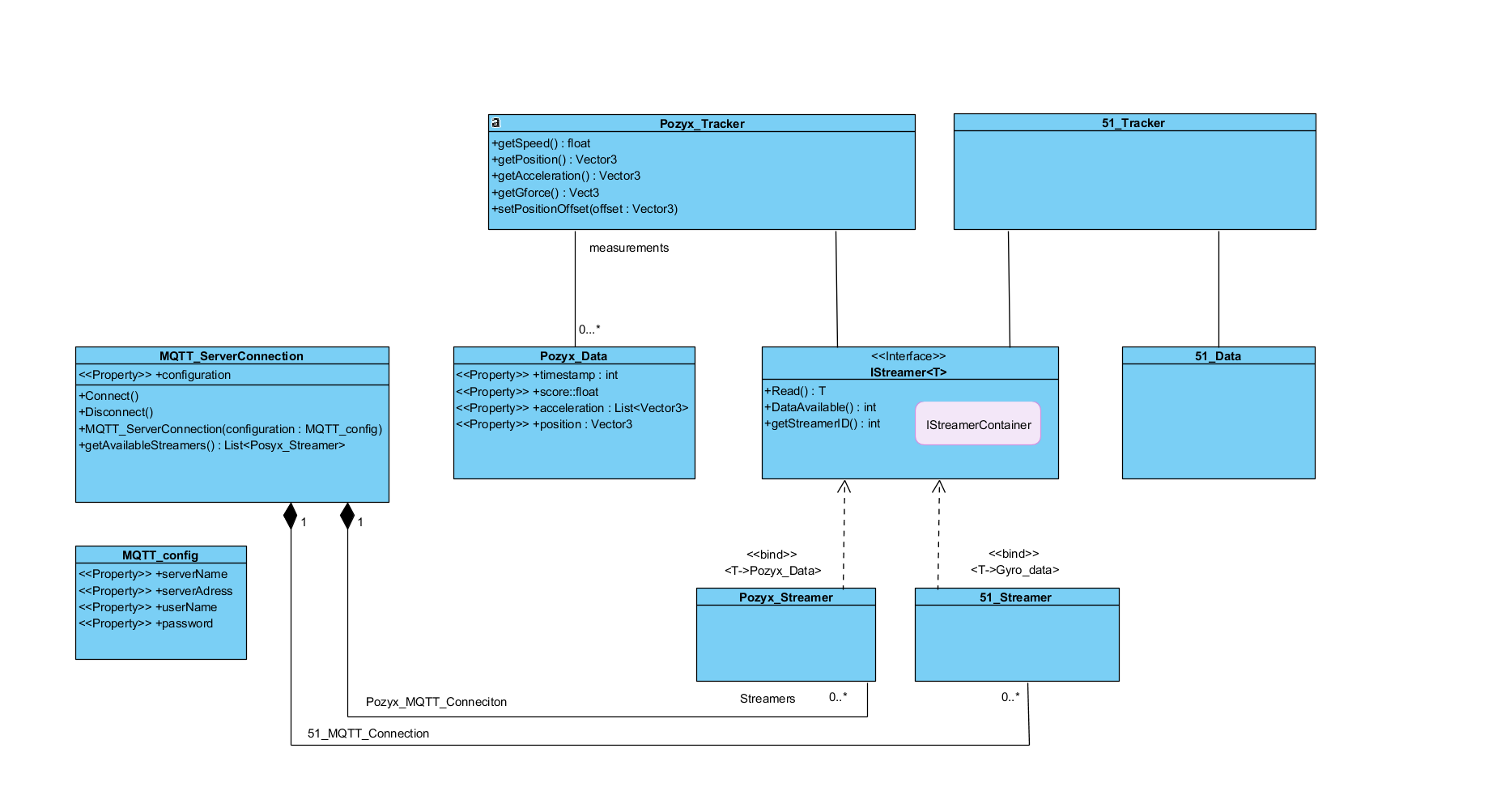
The bottom-most classes are intended to regulate and parse file in- and outputs (e.g. receiving MQTT data, opening JSON/CSV files, etc.). These classes all implement either IStreamer (for MQTT or other “live” protocols) or IFileIO (for file interaction).

This data is all handled by the tracker class (which in our case represents a pozyx sensor). This class will provide data aggregation for 1 sensor (providing f.e. speed, acceleration, etc.)

A selection of sensors is handled by trackerHandler, which can do data aggregation over multiple trackers (for for example speed difference)

## 

## Version 0.2

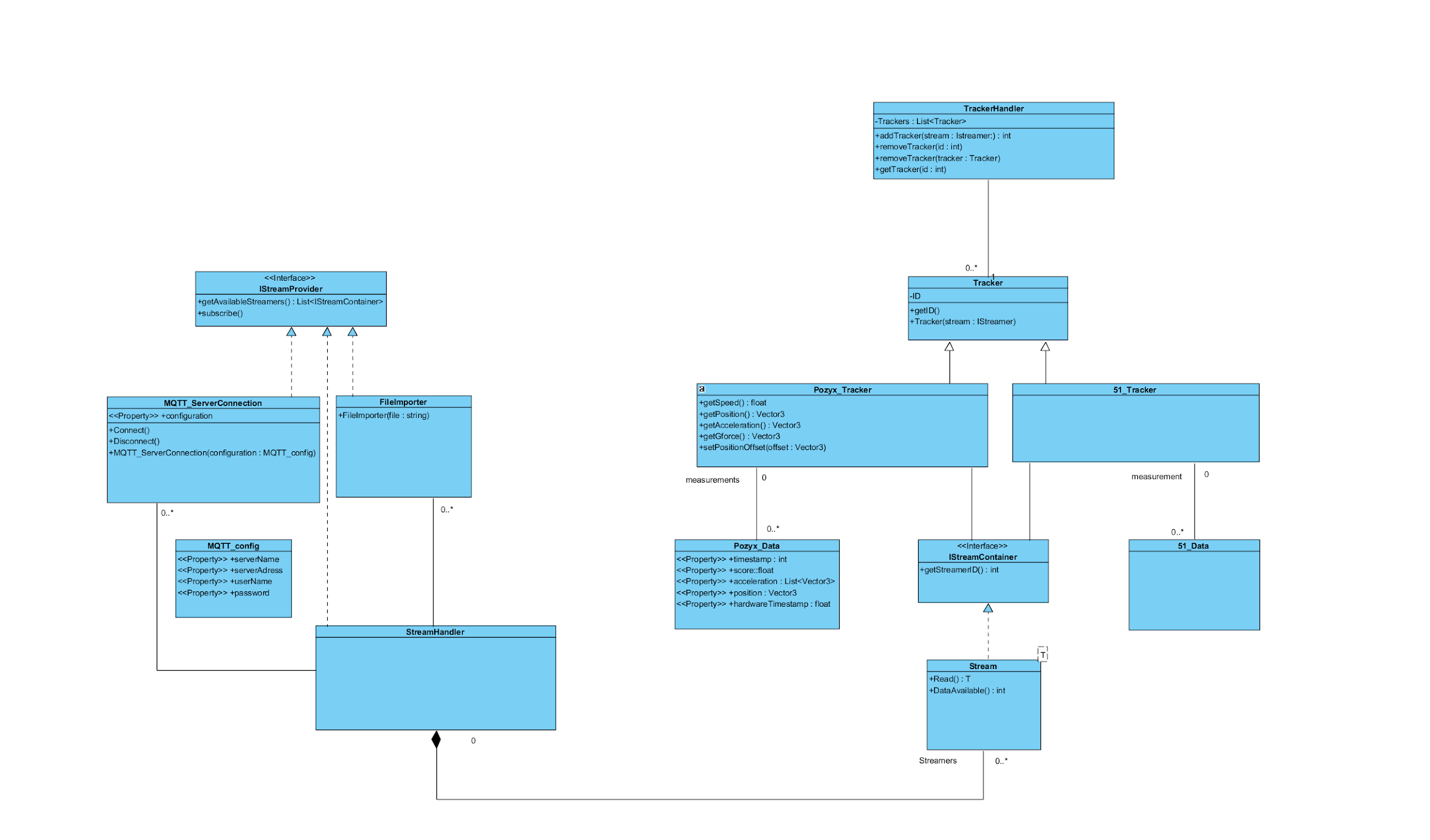


In the second iteration of the design, we got a lot closer to the final design. We added the MQTT class that creates multiple streamers. We also added the possibility of adding another sensor (tracker) with its own data. In this design we did not implement fileIO. We added this in the next version.

At this point in the design, the team had not yet decided on how to connect the pozyx\_streamer (and 51\_streamer) class to the MQTT\_ServerConnection class, because of the possibility of having multiple data input types (for example MQTT and via a HTTP webserver).

## 

## Version 0.3

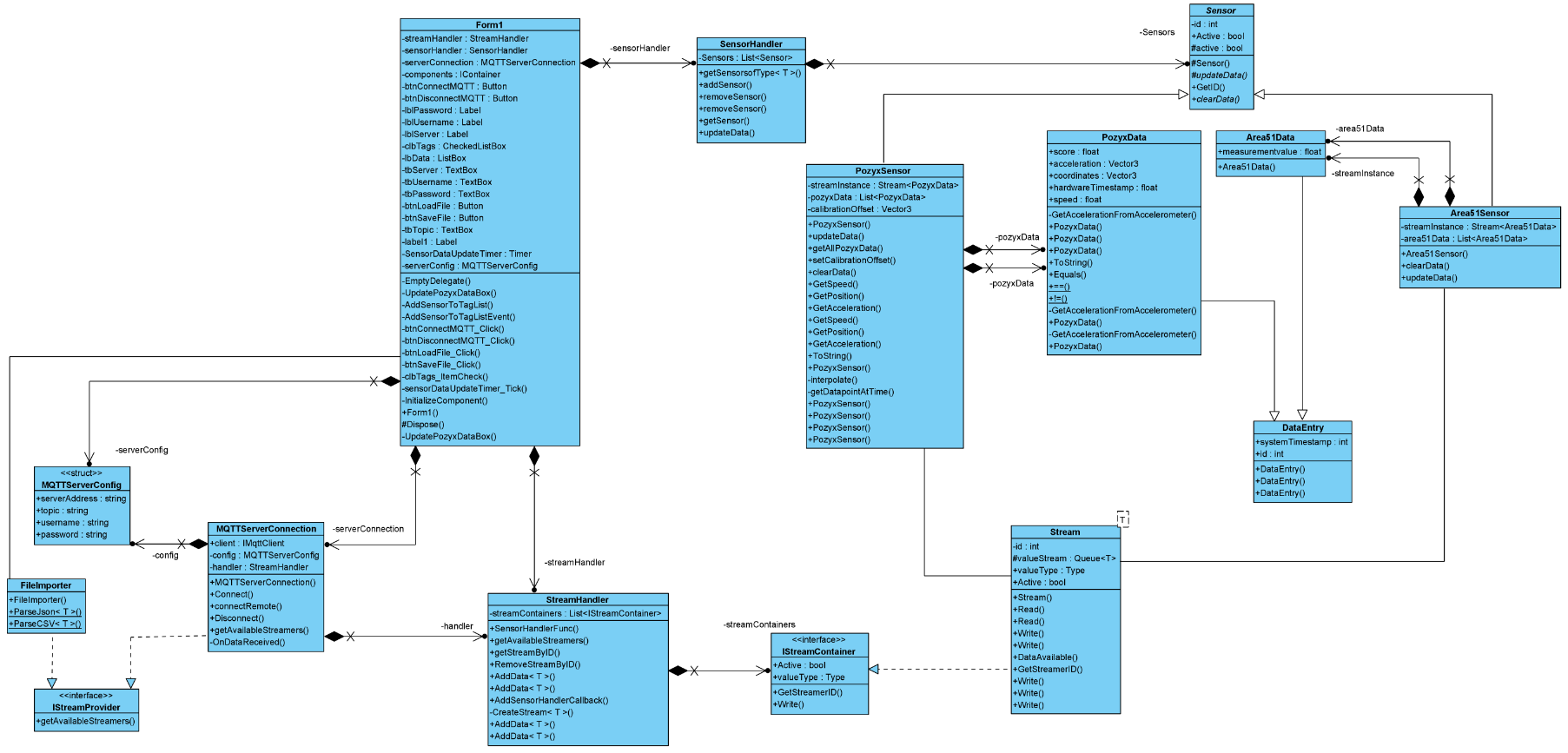
First iteration of the design that will be implemented (in part). The biggest changes here, are that we changed the IStreamContainer to actually work in C#, changing the class that implements IStreamContainer from an Interface to a templated class, and adding a StreamHandler between the classes that create Streams, that will allow the front-end to create stream providers to be given to the StreamHandler.

In the current design, classes that communicate with the front-end are not yet implemented, This part of the design we will create when the actual back-end is mostly finished. We will base this on requirements and feedback from the customer.

The design as it stands is made to be used as a library. This choice was made such that the code can be worked on as separate pieces to ensure portability, reusability and testability.

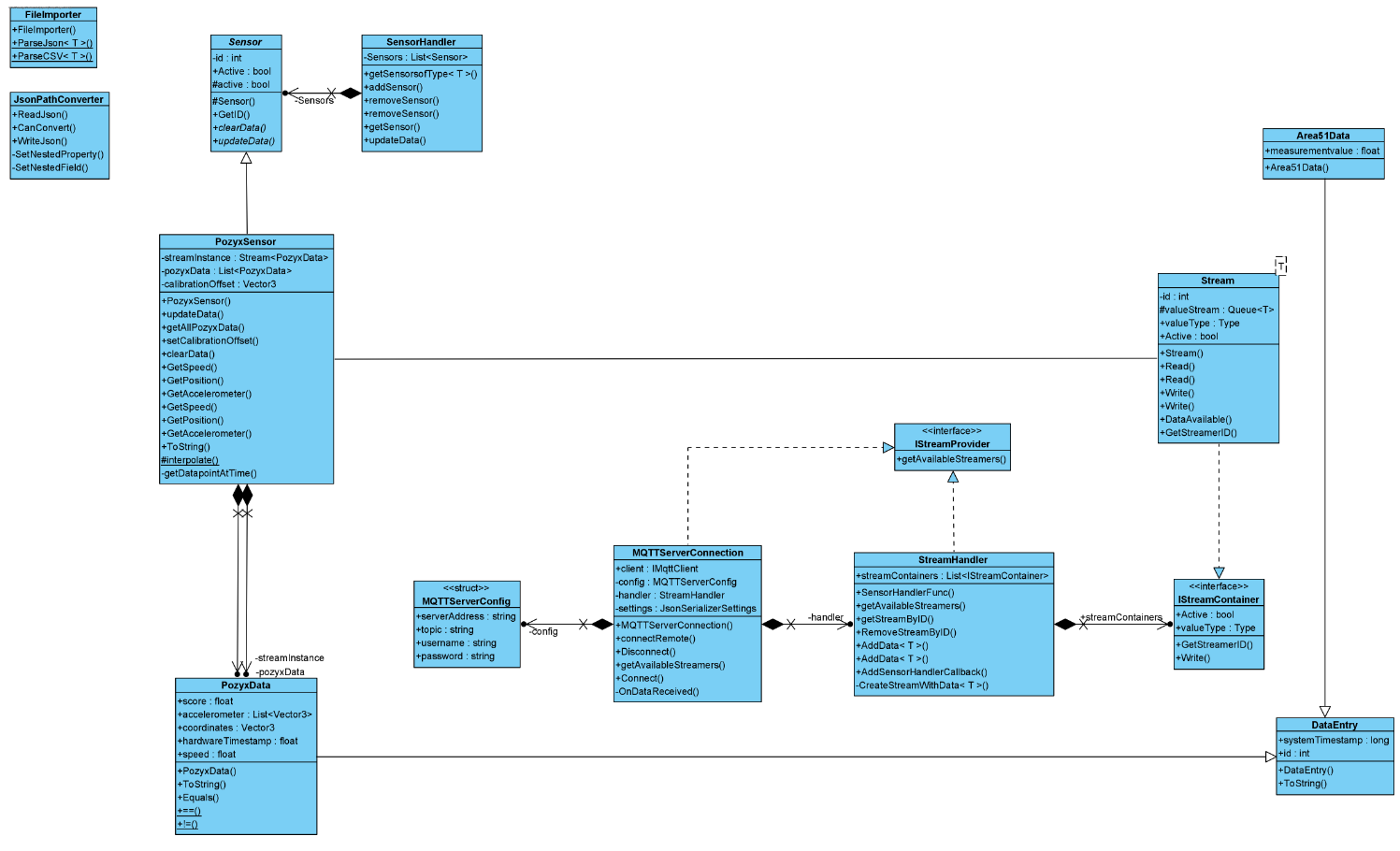
It should be noted that this design is also not finalized, as (for example) file output is not yet implemented.

## Version 0.4



Second iteration of the design that will be implemented (in part). We added the form1 to the class diagram to show how the sensorHandler is connected to the streamers.

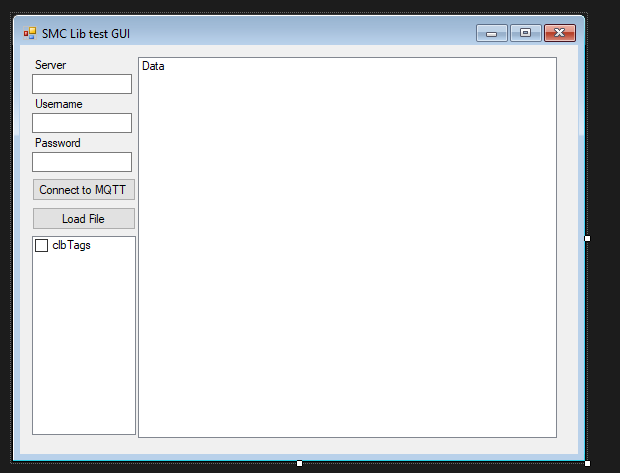
## Version 0.5



# User interface

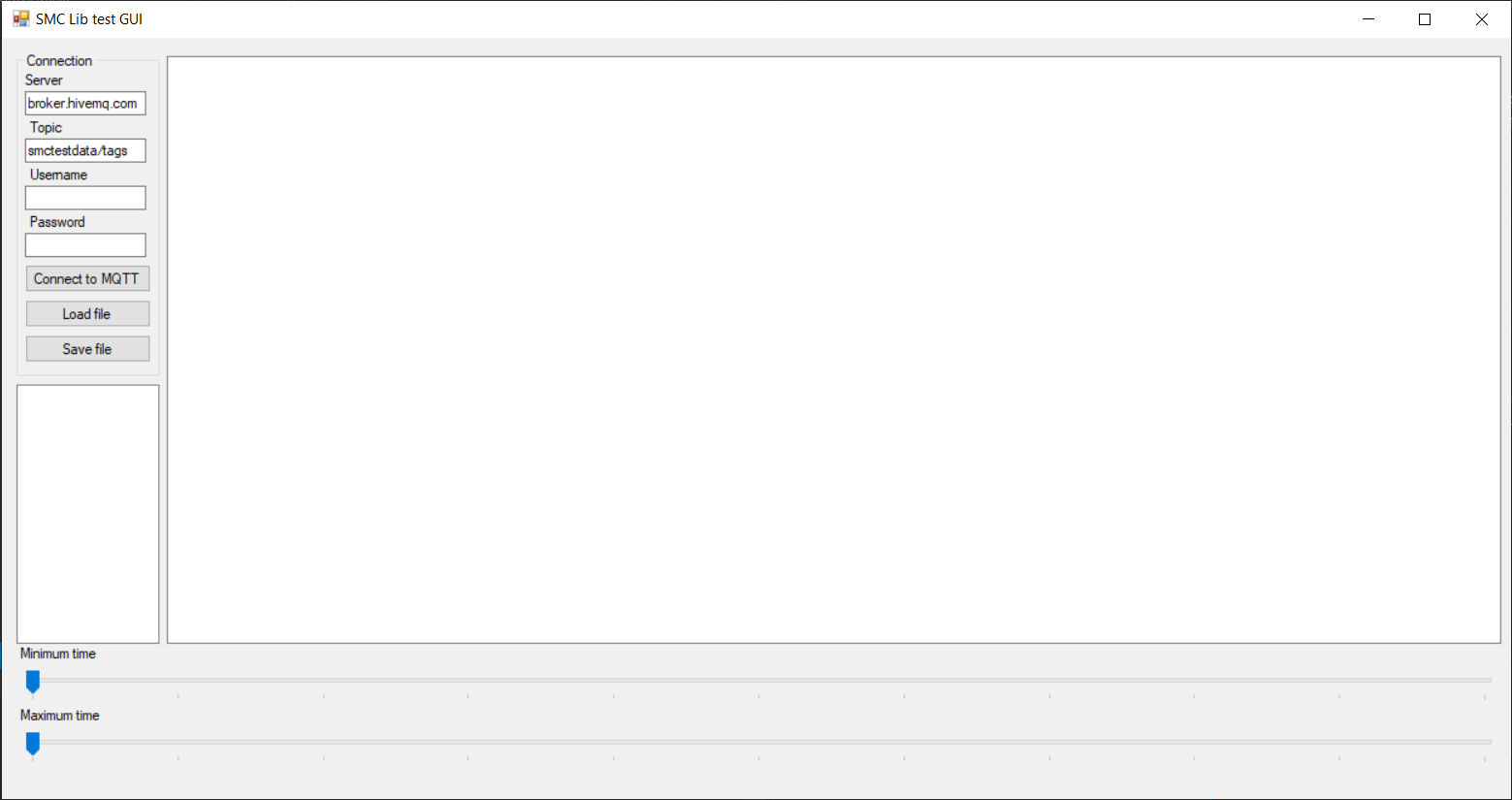
During this project we are not going to focus on the user interface. We did however design a quick UI mainly for testing and showing that the code works.

## Version 0.1



In this design we can connect to the mqtt by subscribing to a topic. We can also load in JSON and CSV files. In the bottom left there is a little listbox showing all the tags connected to the system. In the big listbox we can see all the data coming in (for now only from the pozyx system).

## Version 0.2



In this second design we added a save file button to export the data. We also added 2 sliders so you can view data at a certain point in time.