`greenerthumb` Design

`greenerthumb` is implemented with a subprogram approach. This allows each sensor or client to be implemented with the least extra code and the most reuse of tests. It also simplifies using different languages where they make sense. An example is that this facilitates using Python for the sensors where library support is excellent, Go for infrastructure where efficiency is important, and c++ for GUIs where OpenGL bindings are mature.

Some downsides of this architecture are working with the paths to the subprograms, remembering all the necessary subprograms for a task, and messaging. An activate.sh/deactivate.sh pair is provided to mitigate the first problem. The scripts create aliases to all programs such that they can be run from anywhere. The second problem is addressed by giving composite scripts for some useful combinations of subprograms. Finally, internal messaging is handled by passing JSON lines from STDIN to STDOUT. This allows subprograms to not have to understand the messages, just how to handle JSON

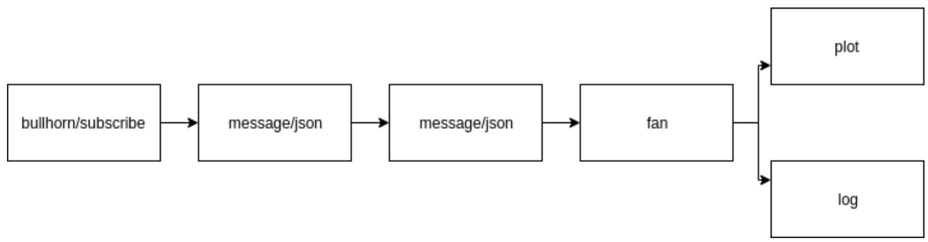
Subprograms are described in later sections. Some of the subprograms also fulfill requirements themselves. An example of this is the `process` subprograms. Four major composite programs are provided:

`run-air`

`run-soil`

`run-logger`

`run-plotter`



`bullhorn` Design

`bullhorn` allows data to be sent on a network from publishers to subscribers (3).

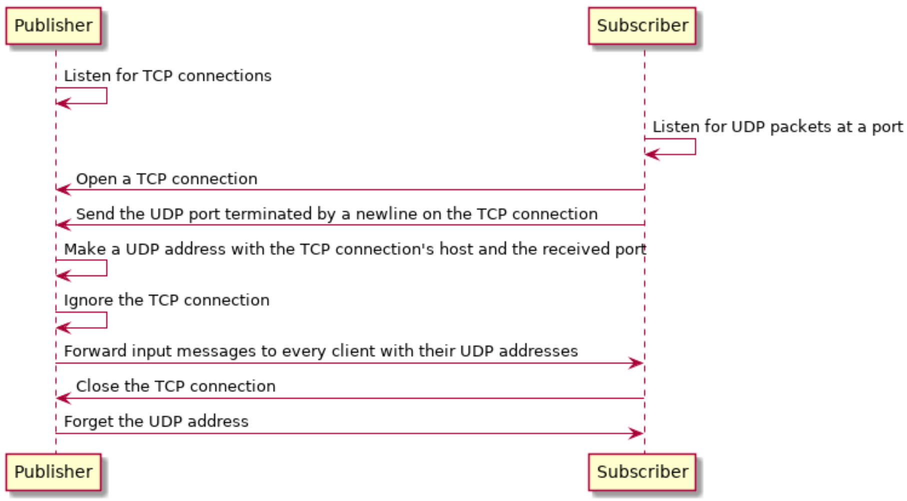


Figure 1: Sequence Diagram

This is done via a pub/sub system which operates over UDP with a TCP trunk. The TCP trunk allows the publisher to know when to stop publishing to subscribers and lets the subscriber know when it needs to try to reconnect to a publisher if reconnect is enabled. The unreliable UDP connection is fine because mostly periodic statuses are sent through the system.

The publisher will publish all newline-separated lines it receives over STDIN to every subscriber until STDIN is closed.

The subscriber prints all newline-separated lines it receives from the publisher until the publisher is closed if reconnect isn't enabled. The subscriber never closes if reconnect is enabled and will just periodically attempt reconnects. The subscriber will always exit with a failure to connect unless terminated because it will either try to reconnect forever or fail to connect to a terminated publisher.

`fan` Design

`fan` connects its STDIN to the STDINs of listed out-programs and STDOUTs from listed out-programs to STDINs of listed in-programs.

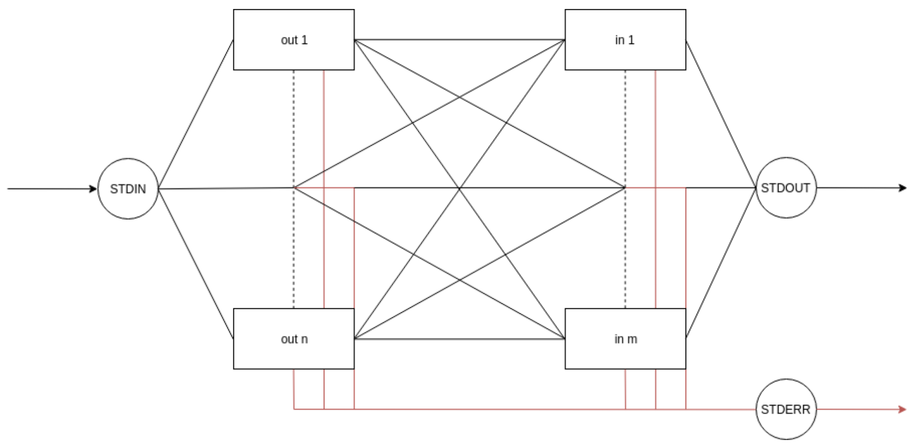


Figure 1: Diagram

Once data is written to `fan`'s STDIN, it gets written to the STDIN of all the in-programs. All of the in-programs' STDOUTs is copied into all of the out-programs' STDIN. All of the out-programs' STDOUT is then recombined and copied to `fan`'s STDOUT. Every program's STDERR is copied to `fan`'s STDERR. An in or out program failing just filters out that program and the rest of `fan` keeps running. `fan` exits once all of its programs exit.

`log` Design

`log` messages from STDIN to a file (4).

This just copies each line of STDIN to a file. Logs have the time appended to the file-name and are rotated each day.

`message` Design

`message` is where `greenerthumb` messages from the ICD are defined and the bytes-JSON conversion is implemented. Converters are provided for both directions. The converters run for continuous input instead of one message at a time so it is easier to pipe with other programs. Errors in input are ignored with a log so the converter can continue running.

plot Design

`plot`s `greenerthumb` JSON messages from STDIN (5).

UI

`plot` will convert input messages to points on a line-graph. The expected message format is:

```
{
  "Name": <message_name>,
  "Timestamp": <timestamp>,
  <name>: <value>,...
}
```

Messages don't necessarily need to be `greenerthumb` messages. They just need to fit this format.

Each non-ID and non-timestamp field will become a line in the graph. The name of the line will be determined by concatenating the <message_name> and the field's <name>. The line's will be assigned unique random colors which will be displayed in a legend with the message name's on the right side of the plot.

Each line will be overlayed to allow trend comparison. To do this, each line will have units normalized to eachother. The x-axis will have units of hours scaled to the period of all the received messages. Ranges of units are presented in the legend to account for the normalization.

If received messages have the same timestamp, the newest message will overwrite the older messages.

A save button makes screenshots.

`plot` only closes once commanded to close instead of once STDIN is closed.

`process` Design

`process` `greenerthumb` data.

Programs

All programs accept `greenerthumb` ICD JSON messages from STDIN and report results to STDOUT. Each program terminates with a message printed to STDERR if any JSON message is malformed.

`summarize`

`summarize` reads all input until STDIN is closed and then reports a 5-number-summary for each data-type along with how many instances of that data-type were included.

`flatten`

`flatten` smooths data by keeping a sliding window of 3 instances of a data-type and replacing it with a weighted average of the 3 instances biased towards the middle instance. The first instance and last instance have a copy of themselves used as the instance to the left and right of them.

The left and right values are weighted by 1/6 each while the middle value is weighted 2/3.

`filter`

`filter` instances of data-types by specifying a list of ANDing conditions in the set of less than or equal to, less than, equal, greater than, and greater than or equal to and filtering STDIN according to the conditions.

An epsilon value for comparisons can also optionally be passed. The system epsilon is used otherwise.

`clean`

`clean` reads all input until STDIN is closed and filters instances that are more than a passed number of standard deviations away from the mean.

`sense` Design

`sense` programs write `greenerthumb` JSON messages from sensors to STDOUT. These can be `fan`ned into `message/bytes` piped into `bullhorn/publish`.

Sensors

`air`

`air` senses the 'Air Status Message' at 0.1 hertz (2a).

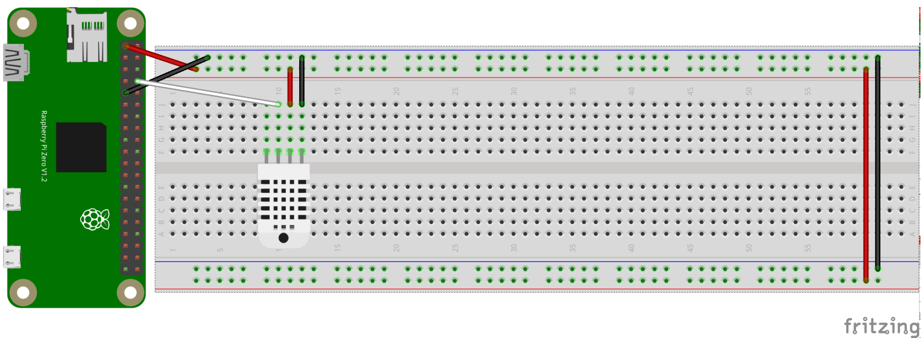


Figure 1: Air Schematic

`soil`

`soil` senses the 'Soil Status Message' at 0.1 hertz (2b).

Emulators

Emulators are provided for all programs and each accepts an optional rate flag.

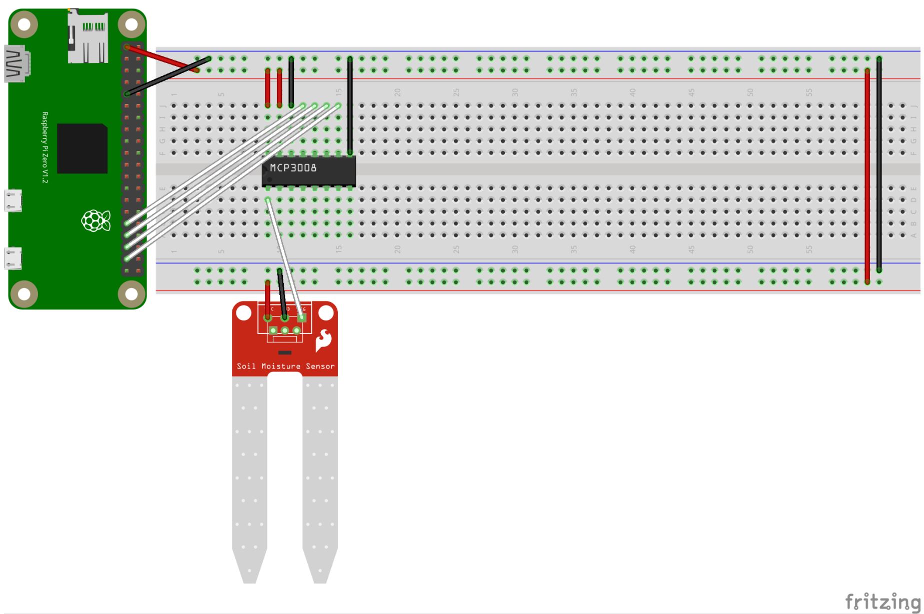


Figure 2: Soil Schematic