Plensor code documentation

Owner	Thijs@plense.tech
Data science ideas	
Last edited	@22 augustus 2024 11:06
Last edited by	Thijs@plense.tech
Meerdere selecteren	
Notebook	
Status	Active
Tags	

- [x] ~~Parser.ino volledig documenteren~~
- [x] ~~PlenseFunctions.ino volledig documenteren~~
- [ ] IO.ino volledig documenteren
- [ ] Arduino code
  - [ ] Communication
    - [x] ~~Receive uitleggen~~
    - [ ] Transmit uitleggen
  - [x] ~~Commands~~
- [ ] C code
  - [ ] Commands functies uitleggen

Summary

This document aims to summarise and explain the code used for the Plensor device, entailing both the Arduino and C code. blablabla

Arduino code

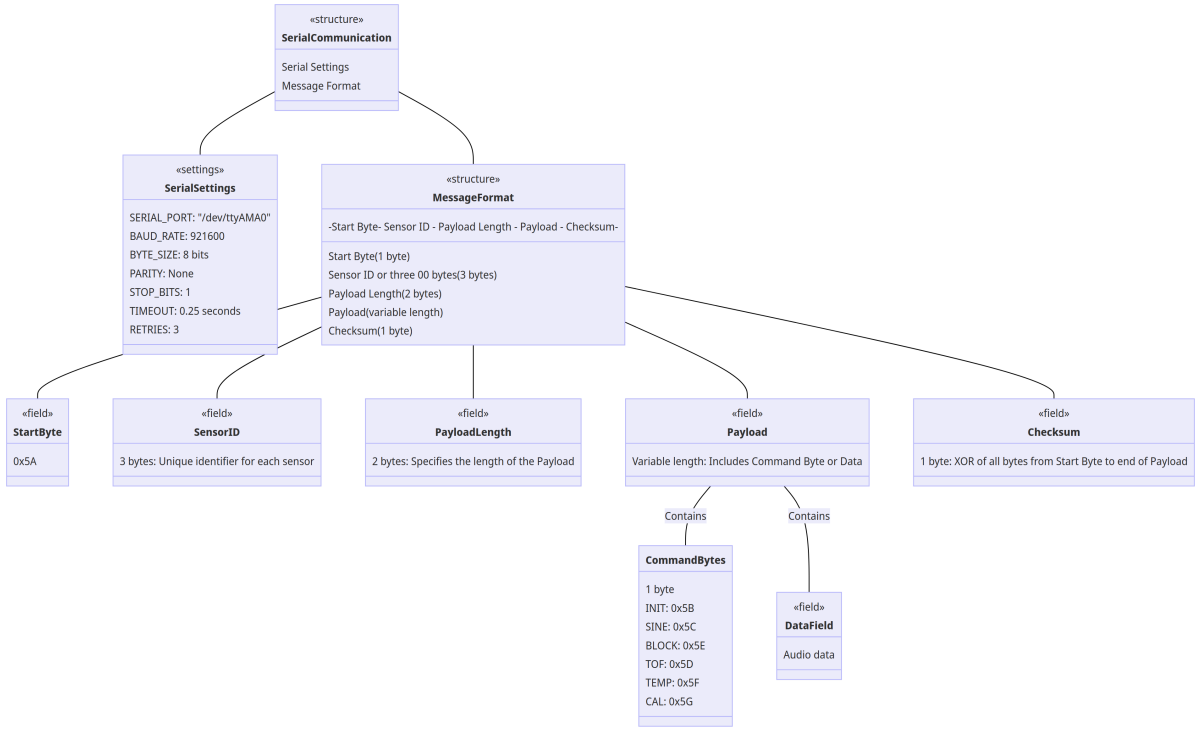
The Arduino code is used to manage the overall operation of the device, acting as the main control system. It coordinates the high-level functions, processes inputs and outputs, and ensures the smooth execution of the program logic. This code will be explained in this section, first describing the overarching process, followed by the various code functions associated with that specific process, and an explanation on how those functions work.

Communication

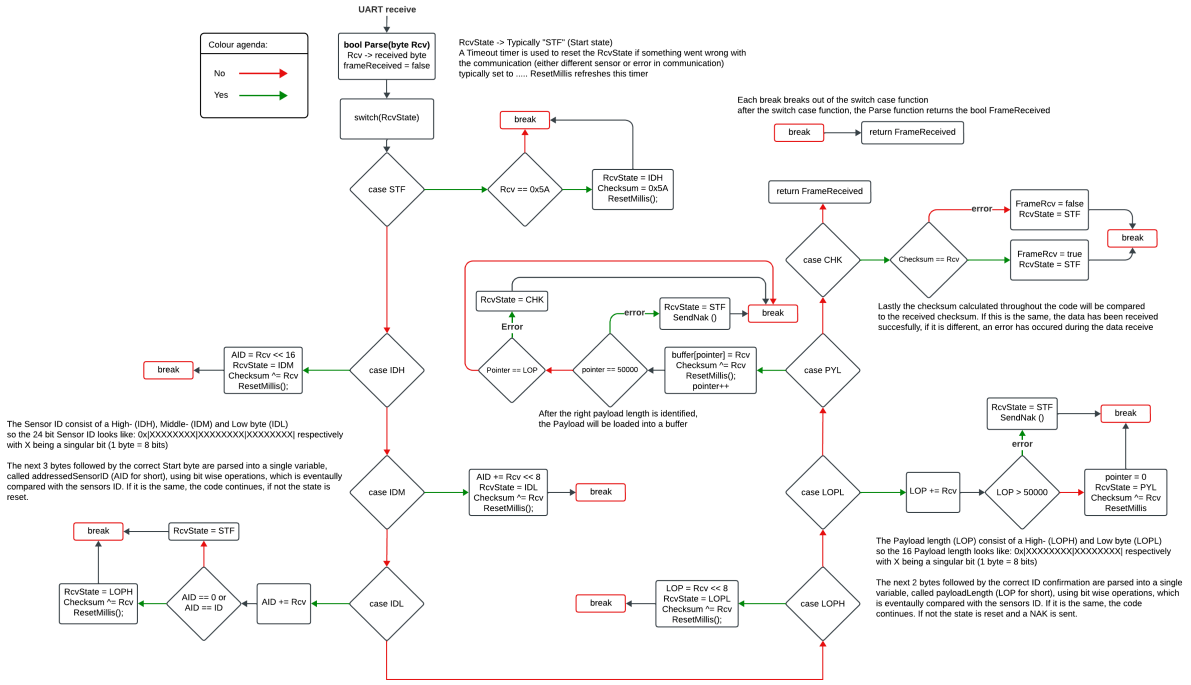
The Plensor is driven by a raspberry pi, communicating through the RS485 protocol. The raspberry pi tells the Plensor what to measure and when to do it, making it a flexible system.

Receiving commands

The Plensor can start measure certain things when it is requested to do so. To request a command, the raspberry pi emits a specific string of bytes through RS485, following a specific protocol. This protocol, consisting of a StartByte, SensorID, PayloadLength, Payload and Checksum, looks like so:



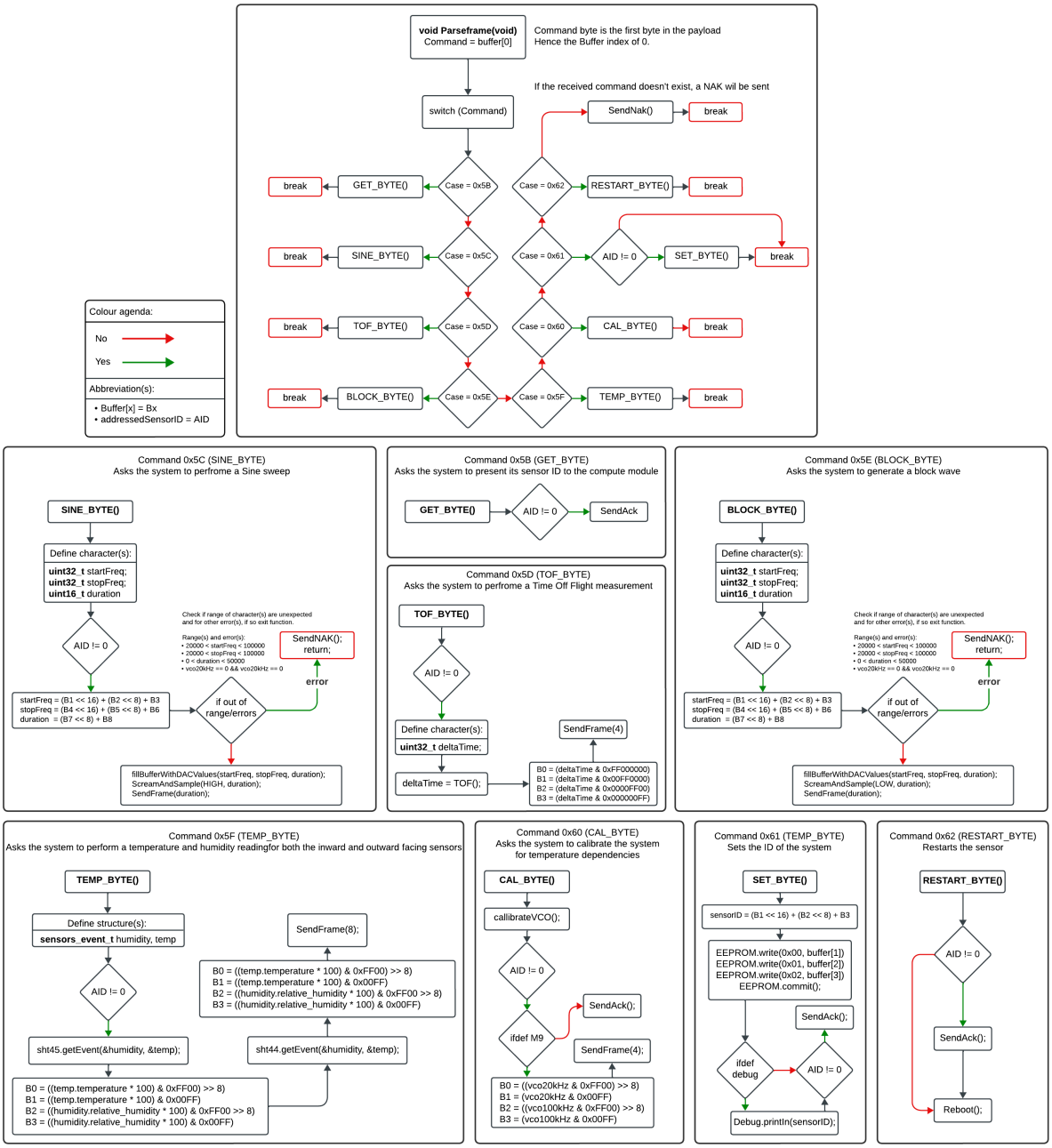
The Plensor then processes the command, checking each byte to identify the right start byte, sensor ID, checksum, etc. Using a state controlled switch-case function “bool Parse(byte Rcv)”, this function processes the command using the following switch-case logic:



Sending information

Commands and calling upon execution

Both the Arduino code and C code include functions that processes and executes commands. The Arduino side handles most of the processing. after a command is received through communication, the ‘*ParseFrame*’ function gets called, which then invokes the appropriate function corresponding to that specific command. Each command function then extracts the necessary parameters from the payload and calls upon the relevant C code functions, which are further explained in the “C code” section:



C code

The Plensor is used to gather information about a plant using ultrasonic technology. The C code in the Plensor is responsible for directly interacting with the hardware, such as sensors and actuators, handling low-level tasks like signal processing, timing, and communication with the ultrasonic components. This code will be explained in this section, first describing the overarching process, followed by the various code functions associated with that specific process, and an explanation on how those functions work.

Command execution

Vragen/opmerkingen

betreft de onderstaande functie SET_BYTE. Volgens mij mag het ID geen 0 zijn. als dat zo is is het misschien handig om na de sensorID definitie te kijken of de sensorID != 0, en als dat zo is pas de EEPROM en debug.print te doen. dan kan je niet perongelijk het ID van het systeem 0 maken:

```
//SET BYTE: 0x61 Sets the ID of the
void SET_BYTE(void) {
  sensorID = (buffer[1] << 16) + (buf
  EEPROM.write(0x00, (uint8_t)buffe
  EEPROM.write(0x01, (uint8_t)buffe
  EEPROM.write(0x02, (uint8_t)buffe
  EEPROM.commit();

#ifdef debug
  Debug.println(sensorID);
#endif
  //sensorIDinEEPROM.write(sensorI
  if (addressedSensorID != 0) {
    SendAck();  // Send Ack
  }
}
```

```
//SET BYTE: 0x61 Sets the ID of the
void SET_BYTE(void) {
  sensorID = (buffer[1] << 16) + (buf
  if (sensorID != 0) {
      EEPROM.write(0x00, (uint8_t)bu
      EEPROM.write(0x01, (uint8_t)buf
      EEPROM.write(0x02, (uint8_t)bu
      EEPROM.commit();

    #ifdef debug
      Debug.println(sensorID);
    #endif
      //sensorIDinEEPROM.write(sens
      if (addressedSensorID != 0) {
        SendAck();  // Send Ack
      }
  }
}
```