ARDUINO to PLATINUM COMMS

HELP DOCUMENT

Dynament Limited

Hermitage Lane Industrial Estate · Kings Mill Way · Mansfield · Nottinghamshire · NG18 5ER · UK.

Tel: 44 (0)1623 663636

email: sales@dynament.com · www.dynament.com

Contents

Dynament Limited 1
Connecting the Sensor 3
Arduino IDE 5
Code Explanation 9
Packet Breakdown 11
Using Serial.read() 13
  Advanced Conversion Notes 14

Connecting the Sensor

This data sheet uses the Arduino Mega as an example. The Ardunio Mega provides more than one comm port, therefore comm port 1 is used to communicate with the sensor and comm port 0 is used to print to the PC.

The Arduino uses 5v logic high whereas the Platinum Sensor uses 3.3v, so to prevent damage to the Sensor a voltage divider must be used.

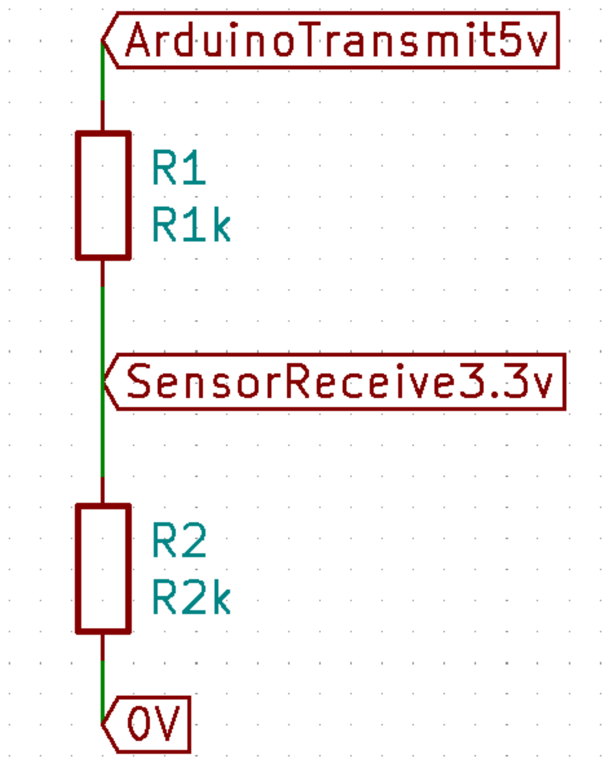


Figure 1: Lowers the voltage to useable level

The Sensor transmit line going to the Arduino receive doesn't need a divider as 3.3v is an acceptable input to the Arduino.

In order to power the Sensor it must be connected to 5v and 0v. To do this you can use the pins on the Arduino.

After this is complete, the sensor should now have the following pins connected:

5v -> 5v Arduino pin

0v -> Arduino GND

Tx -> Arduino RX1

Rx -> Goes to the output of the potental divider. The input goes to Arduino Tx

After this is complete your Platinum Sensor should be connected as shown:

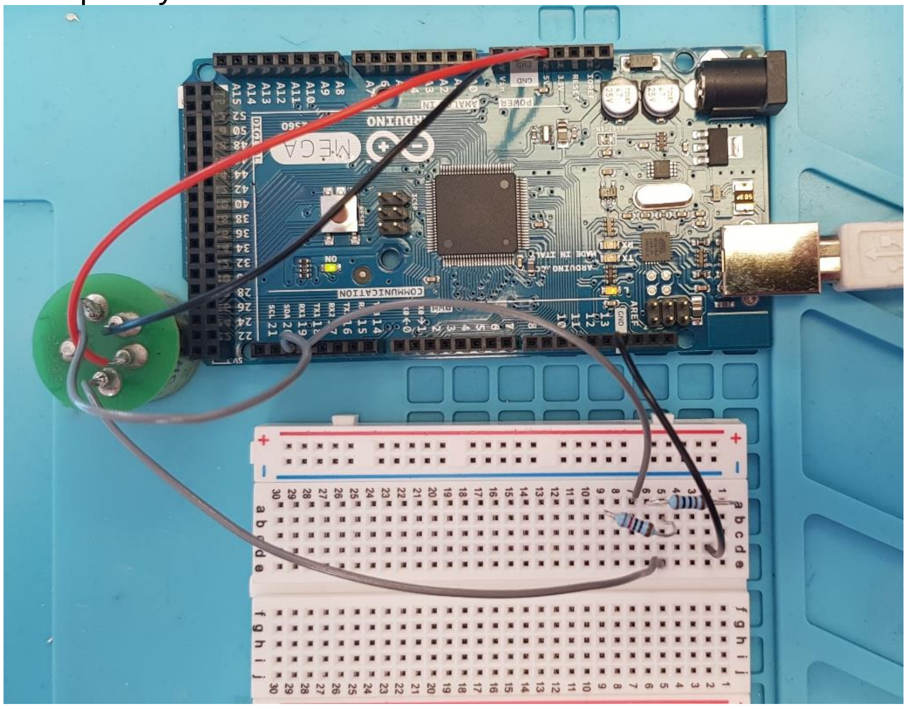
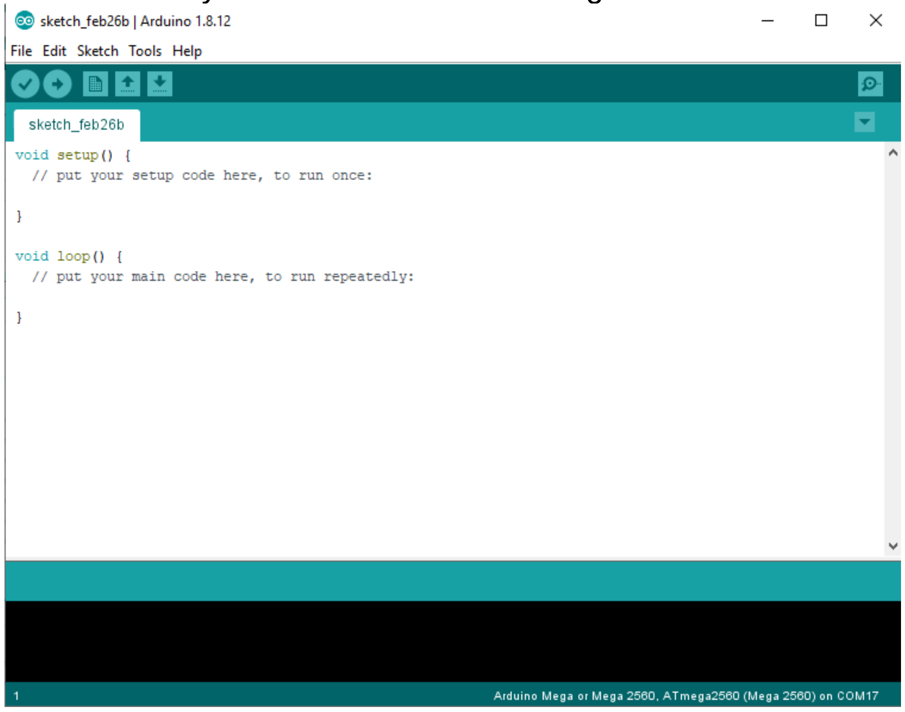


Figure 2: Sensor is shown upside down with a solder adapter

If you are using an Arduino with only one comm port (like the Arduino Uno) you will have to connect it to that, however when you use the serial monitor (shown later) it will also show the hex that is transmitted.

Arduino IDE

Go to the Arduino website and download the newest version of the Arduino IDE software. Once installed you should see the following screen:

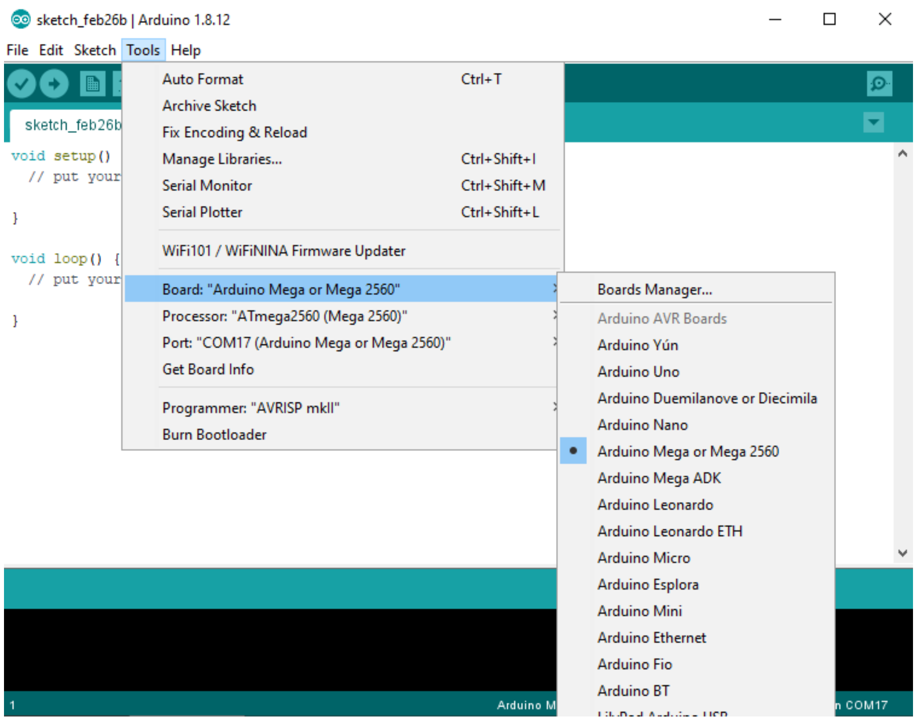


```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

```
1                Arduino Mega or Mega 2560, ATmega2560 (Mega 2560) on COM17
```

Figure 3: Arduino home screen

In the tools drop down menu select the Arduino board, processor and port you are using:



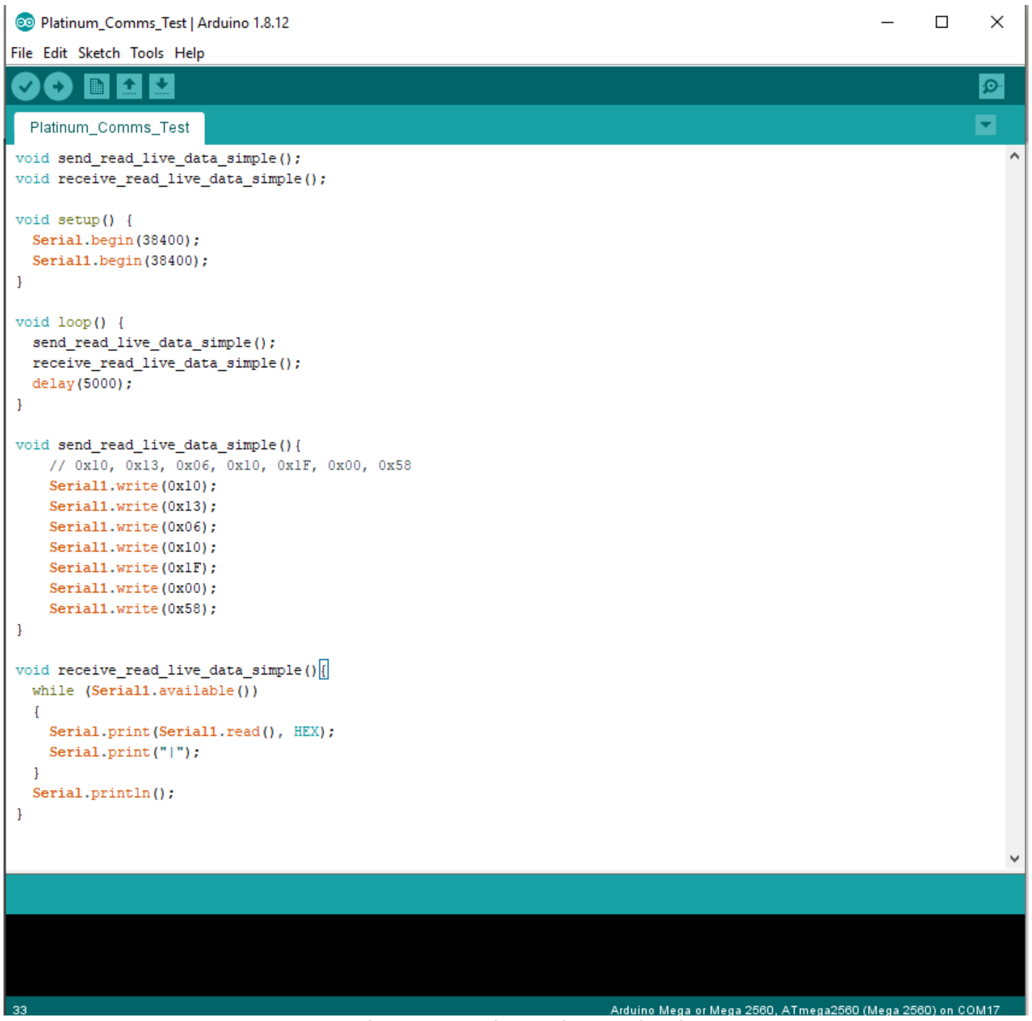
Board: "Arduino Mega or Mega 2560"
Processor: "ATmega2560 (Mega 2560)"
Port: "COM17 (Arduino Mega or Mega 2560)"

```
void setup()
  // put your

}

void loop() {
  // put your

}
```

Figure 4: Select Board, Processor and Port options

Copy in this example code:

```
void send_read_live_data_simple();
void receive_read_live_data_simple();

void setup() {
  Serial.begin(38400);
  Serial1.begin(38400);
}

void loop() {
  send_read_live_data_simple();
  receive_read_live_data_simple();
  delay(5000);
}

void send_read_live_data_simple(){
    // 0x10, 0x13, 0x06, 0x10, 0x1F, 0x00, 0x58
    Serial1.write(0x10);
    Serial1.write(0x13);
    Serial1.write(0x06);
    Serial1.write(0x10);
    Serial1.write(0x1F);
    Serial1.write(0x00);
    Serial1.write(0x58);
}

void receive_read_live_data_simple(){
  while (Serial1.available())
  {
    Serial.print(Serial1.read(), HEX);
    Serial.print("|");
  }
  Serial.println();
}
```


Platinum_Comms_Test | Arduino 1.8.12
File Edit Sketch Tools Help
Platinum_Comms_Test

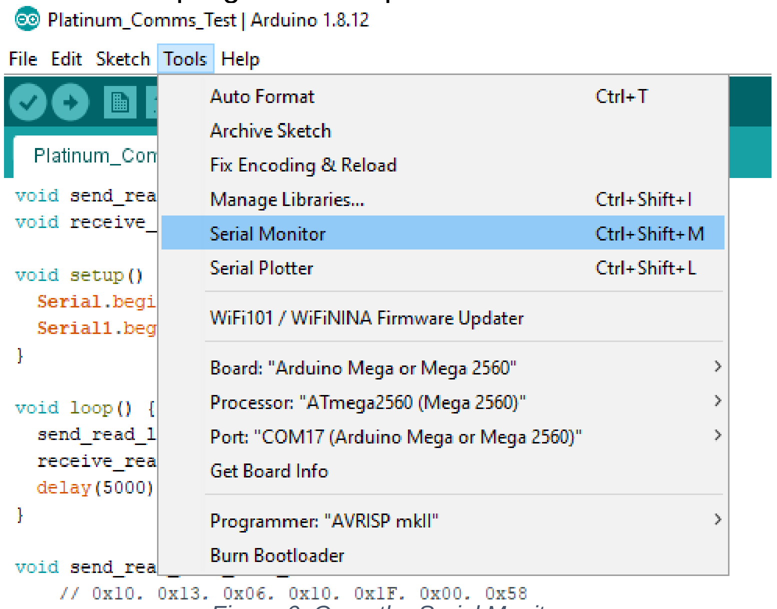
```
void send_read_live_data_simple();
void receive_read_live_data_simple();

void setup() {
  Serial.begin(38400);
  Serial1.begin(38400);
}

void loop() {
  send_read_live_data_simple();
  receive_read_live_data_simple();
  delay(5000);
}

void send_read_live_data_simple(){
    // 0x10, 0x13, 0x06, 0x10, 0x1F, 0x00, 0x58
    Serial1.write(0x10);
    Serial1.write(0x13);
    Serial1.write(0x06);
    Serial1.write(0x10);
    Serial1.write(0x1F);
    Serial1.write(0x00);
    Serial1.write(0x58);
}

void receive_read_live_data_simple(){
  while (Serial1.available())
  {
    Serial.print(Serial1.read(), HEX);
    Serial.print("|");
  }
  Serial.println();
}
```

```
33                              Arduino Mega or Mega 2560, ATmega2560 (Mega 2560) on COM17
```

Figure 5: Code ready to upload

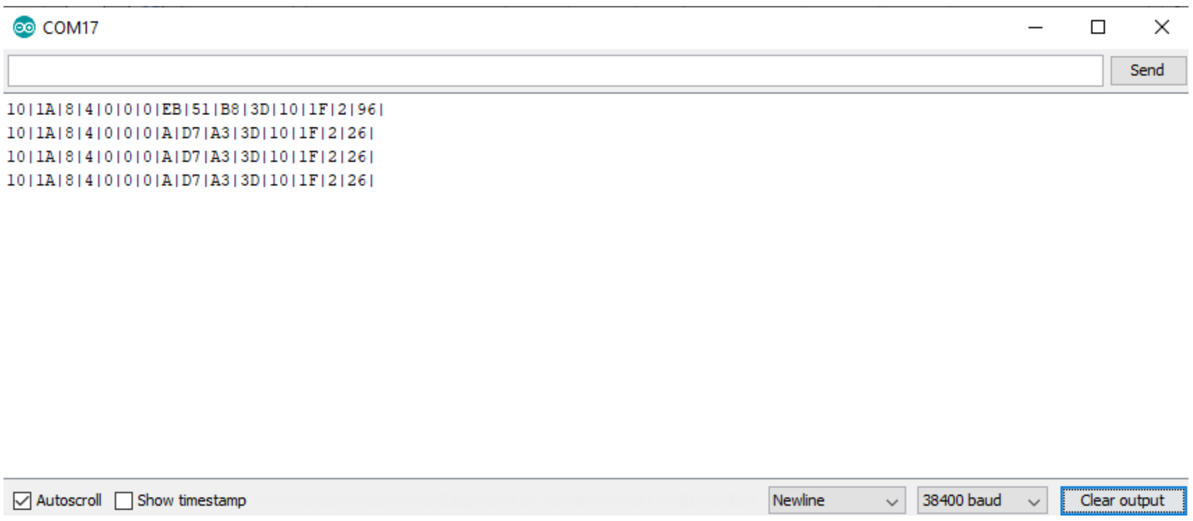
Click the arrow to upload the code to the Arduino.

After the Arduino has been programmed open the serial monitor.


Platinum_Comms_Test | Arduino 1.8.12
File Edit Sketch Tools Help
Platinum_Con
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Manage Libraries... Ctrl+Shift+I
Serial Monitor Ctrl+Shift+M
Serial Plotter Ctrl+Shift+L
WiFi101 / WiFiNINA Firmware Updater
Board: "Arduino Mega or Mega 2560"
Processor: "ATmega2560 (Mega 2560)"
Port: "COM17 (Arduino Mega or Mega 2560)"
Get Board Info
Programmer: "AVRISP mkII"
Burn Bootloader

```
void send_rea
void receive_

void setup()
  Serial.begi
  Serial1.beg
}

void loop() {
  send_read_l
  receive_rea
  delay(5000)
}

void send_rea
    // 0x10, 0x13, 0x06, 0x10, 0x1F, 0x00, 0x58
```

Figure 6: Open the Serial Monitor



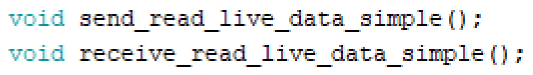
```
10|1A|8|4|0|0|0|EB|51|B8|3D|10|1F|2|96|
10|1A|8|4|0|0|0|A|D7|A3|3D|10|1F|2|26|
10|1A|8|4|0|0|0|A|D7|A3|3D|10|1F|2|26|
10|1A|8|4|0|0|0|A|D7|A3|3D|10|1F|2|26|
```

Newline · 38400 baud

Figure 7: The Serial Montor shows the packet that has been received

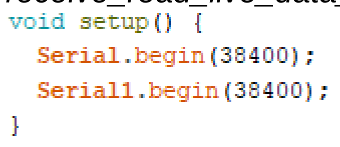
Code Explanation

The Arduino IDE uses C++ to program the Arduino.



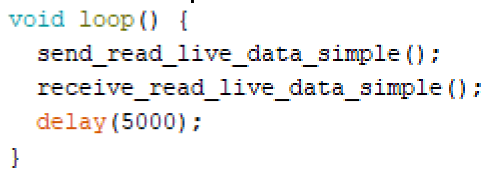
```
void send_read_live_data_simple();
void receive_read_live_data_simple();
```

This line is a forward declaration. This is used to tell the Microcontroller that further down in the program the '*send_read_live_data_simple'* function and the '*receive_read_live_data_simple*' function will be called.



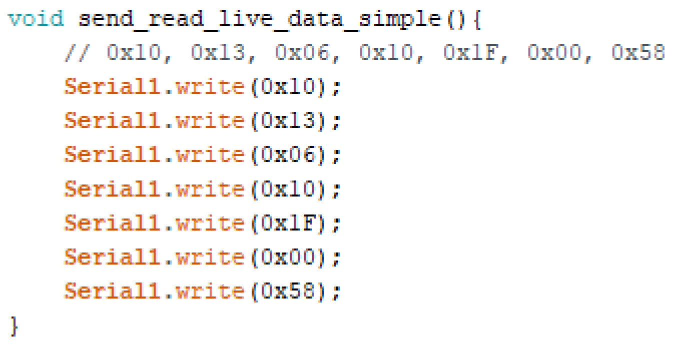
```
void setup() {
  Serial.begin(38400);
  Serial1.begin(38400);
}
```

Next is the setup function. This code gets run only once on startup. It starts the Serial0 and Serial1 ports. Serial0 is what is shown in the serial monitor screen. Serial1 is the port to communicate with the sensor.



```
void loop() {
  send_read_live_data_simple();
  receive_read_live_data_simple();
  delay(5000);
}
```

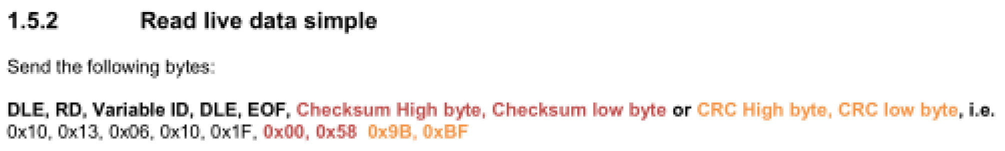
This is the main loop, this code gets repeatedly looped. You can see by reading the function names that it sends a request to read a simplified version of the live data struct. Then it reads the receive port to read the reply. After this the Microcontroller waits 5000mS.



```
void send_read_live_data_simple(){
    // 0x10, 0x13, 0x06, 0x10, 0x1F, 0x00, 0x58
    Serial1.write(0x10);
    Serial1.write(0x13);
    Serial1.write(0x06);
    Serial1.write(0x10);
    Serial1.write(0x1F);
    Serial1.write(0x00);
    Serial1.write(0x58);
}
```

This function writes the request to get the live data simple struct to serial port 1. As previously mentioned if you only have one serial port you should change Serial1 to Serial.

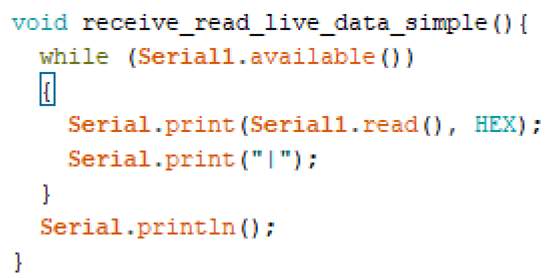
To see the full list of commands, refer to the *Premier sensor Communications protocol* document. Here is the part of the document that tells you what to write for this command:



1.5.2 Read live data simple

Send the following bytes:

DLE, RD, Variable ID, DLE, EOF, Checksum High byte, Checksum low byte or CRC High byte, CRC low byte, i.e.
0x10, 0x13, 0x06, 0x10, 0x1F, **0x00, 0x58** 0x9B, 0xBF



```
void receive_read_live_data_simple(){
  while (Serial1.available())
  {
    Serial.print(Serial1.read(), HEX);
    Serial.print("|");
  }
  Serial.println();
}
```

This function loops the read function while there is still data to be received from the Platinum Sensor. Serial1.read() reads the data from Serial1 which is connected to the sensor and prints it on Serial0 so it can be seen on the serial monitor. The character ‘|’ is then printed to break up each byte that is received to make it clearer on the serial monitor.

After this is complete it writes a new line to the serial monitor.

Packet Breakdown

Figure 8 and 9 show the output of a serial decoder connected to the receive and transmit lines.

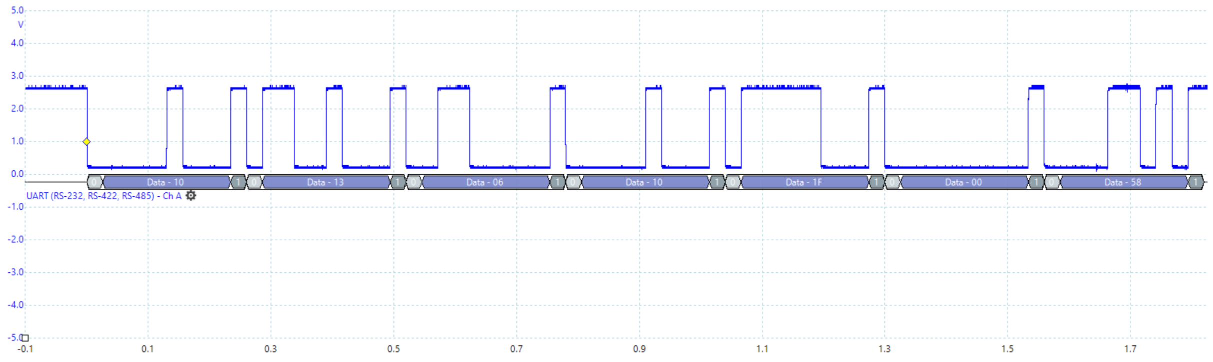


Figure 8: Outgoing Packet

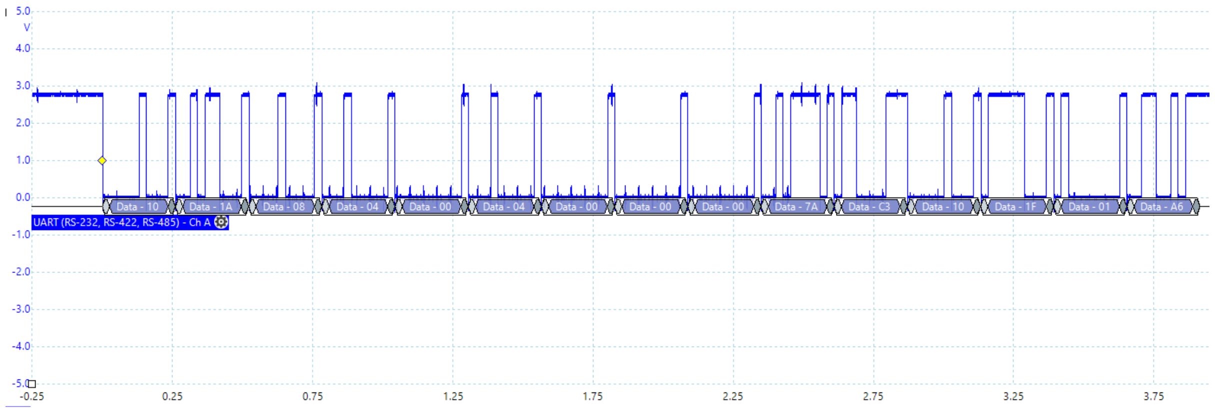
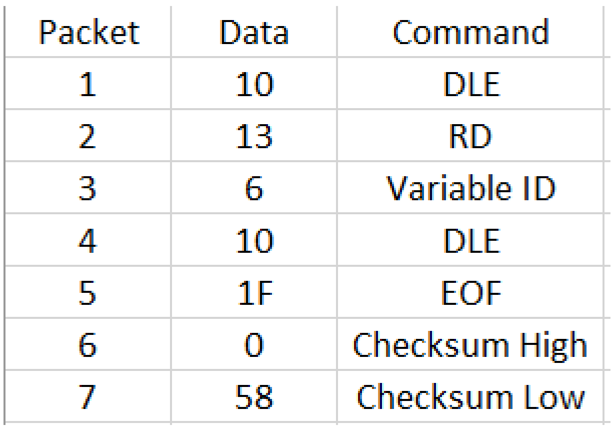


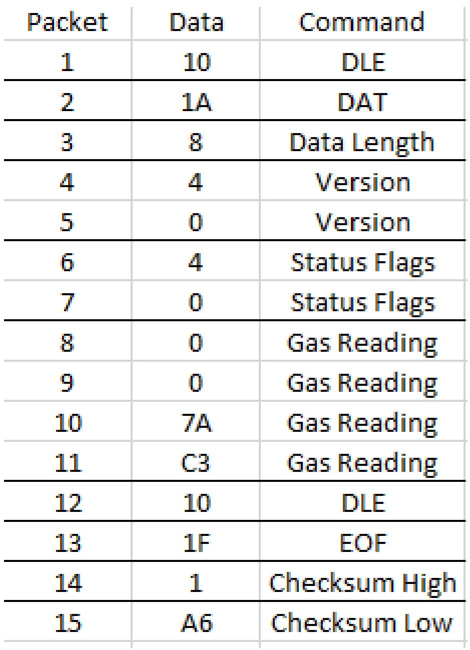
Figure 9: Incoming Packet

Figure 10 and 11 show the outgoing and incoming hex respectively with a column that shows which command it is.



Packet	Data	Command
1	10	DLE
2	13	RD
3	6	Variable ID
4	10	DLE
5	1F	EOF
6	0	Checksum High
7	58	Checksum Low

Figure 10: Outgoing Packet Description



Packet	Data	Command
1	10	DLE
2	1A	DAT
3	8	Data Length
4	4	Version
5	0	Version
6	4	Status Flags
7	0	Status Flags
8	0	Gas Reading
9	0	Gas Reading
10	7A	Gas Reading
11	C3	Gas Reading
12	10	DLE
13	1F	EOF
14	1	Checksum High
15	A6	Checksum Low

Figure 11: Incoming Packet Description

Please note the Gas reading is a decimal not an integer. This decimal is in IEEE-754 format, you can use an online converter like this to convert it. The gas value in this case shows -250 (as it was in error mode at the time).

Using Serial.read()

The previous code only printed the data received to the serial monitor, if you want to save the data in variables you will need to do some further processing. The packet you receive is split into bytes, because of this you will need to concatenate some of this data into variables.

Serial1.Read() returns an int (which for Arduino is 16 bits), however, only the first 8 bits are used. Because of this we can copy it into a smaller data type that is only 8 bits, in this case I will use char.

```
char readByte = Serial1.read();
```

for the packets that are only a byte long, this works fine:

Packet	Data	Command
1	10	DLE
2	1A	DAT
3	8	Data Length

For the packets that are 2 bytes or 4 bytes long you will need to concatenate the data.

4	4	Version
5	0	Version
6	4	Status Flags
7	0	Status Flags
8	0	Gas Reading
9	0	Gas Reading
10	7A	Gas Reading
11	C3	Gas Reading

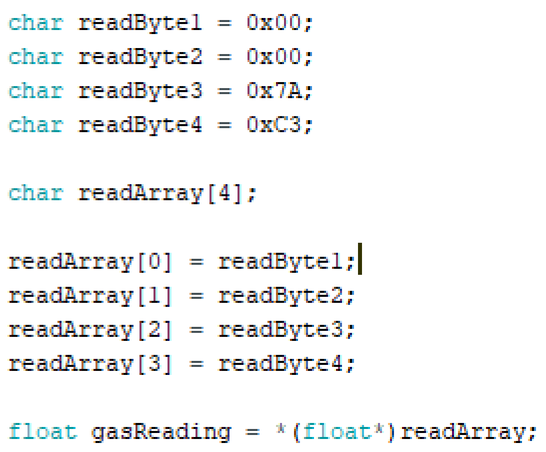
You can do this in a lot of different ways, here what I am going to do is left shift the data and then OR it.

```
char readByte1 = Serial1.read();
char readByte2 = Serial1.read();

int readInt = (int)readByte2 << 8 | readByte1;
```

Using this code, if readByte1 is 0x34 and readByte2 is 0x12.

```
(int)readByte2                  // this converts the 0x12 into 0x0012.
(int)readByte2 << 8             // this shifts the bits over by a byte making it 0x1200.
(int)readByte2 << 8 | readByte1 // this then gets OR'ed, with 0x34 making 0x1234.
```

Another way to do this would be to put the values into an array and then convert the array into the type you want:



```
char readByte1 = 0x00;
char readByte2 = 0x00;
char readByte3 = 0x7A;
char readByte4 = 0xC3;

char readArray[4];

readArray[0] = readByte1;
readArray[1] = readByte2;
readArray[2] = readByte3;
readArray[3] = readByte4;

float gasReading = *(float*)readArray;
```

chars are a byte long, whereas float is 4 bytes long. Because of this if we make an array of 4 chars with our values in it and change the type to float.

In this case readArray is a pointer to a char array. (float*)readArray this part casts it to a pointer to a float and then a * is added to the front to get the value of the float.

Advanced Conversion Notes

1. Serial.read() returns int instead of char because errors will return negative values. Your program should check for this.
2. uint8_t and uint16_t should be used in place of char and int respectively, as these types do not have a standard size (on my PC int is 32 bits whereas on the Arduino it is 16 bits).
3. The comms protocol contains byte stuffed characters (also known as control characters), this is explained in more detail in the tds0045_1 51 Premier sensor Communications protocol document. Because of this the read live data simple packet will occasionally be bigger than expected.