

# **Telematics- CANBUS Monitoring**

**Module: CTEC3426**

**Mohammed Sheikh – P14165162**

## Contents

Introduction .....	1
Technical Specifications .....	2
Filter and Mask .....	3
CAN Messages.....	4
Outgoing CAN Message .....	4
Incoming CAN Message .....	4
SMS Format.....	5
Analysis .....	6
Visual Aspect and Usability .....	6
Technical Considerations .....	7
Automatic Temperature Control .....	9
SMS .....	11
Test Report.....	12
User Guide .....	14
Receiving and sending messages on the CAN network .....	14
How to send commands on the CAN network.....	15
How to use Automatic Temperature Control .....	15
How to send SMS .....	17

## Introduction

In this report I will be detailing the project that I have been tasked to undertake as part of the **Telematics** module. The purpose of this project is to understand how the Controller Area Network (CAN) operates and allows for wired communication between CAN nodes. CAN is a widely used communications protocol which has a lot of uses in the modern world; it is typically found in industrial applications, the automotive industry etc. For example, vehicles have integrated CAN networks that allow Electrical Control Units (ECU's) to communicate with each other.

As part of the project, I have created a piece of software (fig 1) using the **C#** language to allow a CAN board (fig. 2), which acts as a node- to remotely communicate with and control another board. The boards contain an array of components that the program will be able to analyse and control, when a connection is established.

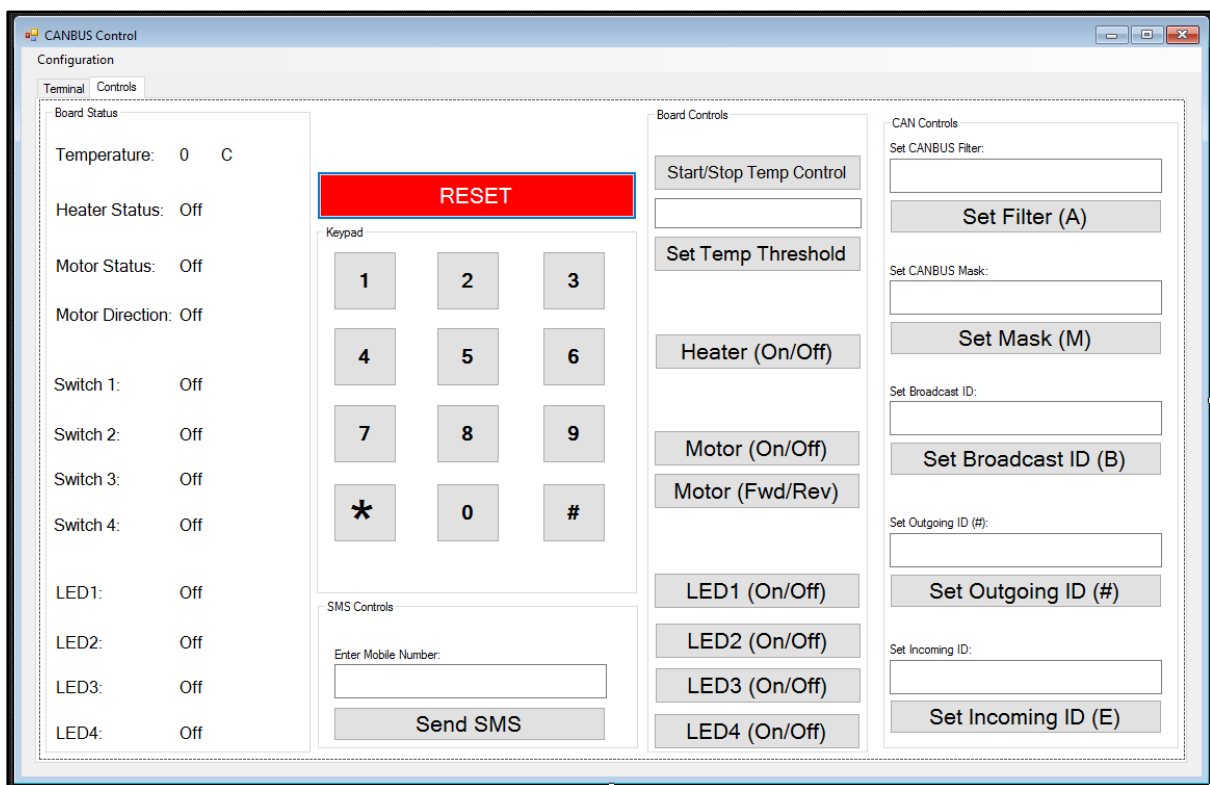


Figure 1: The program's Graphical User Interface (GUI)



Figure 2: A CAN board with integrated components

## Technical Specifications

The main features of the software are as follows:

- **Display Temperature**

View the current temperature of the remote board and display in °C

- **Regulate Temperature**

Regulate the temperature of the remote board- based on a set threshold- by dynamically controlling the motor and heater components

- **Display LED status**

Display the ON/OFF status of the LED lights of the remote board

- **Display Motor Status**

Display the ON/OFF and direction status of the motor of the remote board

- **Display Heater Status**

Display the ON/OFF status of the heater of the remote board

- **Display Keypad Status**

Display the pressed key status of the keypad of the remote board

- **Display Switch Status**

Display the ON/OFF status of the mechanical switches of the remote board

- **Send SMS Message**

Send an SMS text message to a GSM phone, displaying the status of the remote board components

- **Set CAN Filter**

Set CAN filter of local board to set up an identifier for the messages to retrieve

- **Set CAN Mask**

Set CAN mask of local board to define how of the CAN filter needs to be considered

- **Send CAN Messages**

Send CAN messages across the network

- **View CAN Messages**

Read CAN messages that remote boards have sent via the network

- **Control LED's**

Control the ON/OFF status of the LED's of the remote board

- **Control Heater**

Control the ON/OFF status of the heater of the remote board

- **Control Motor**

Control the ON/OFF status and direction of the motor of the remote board

## Filter and Mask

The **Mask** and **Filter** are two concepts that are used to retrieve messages from the **CAN** network. Filter and Mask work in combination allowing a node on the network to accept/reject messages that have been sent by another node. This is essential as the messages that are sent across the CAN network are sent to all nodes as it is a **Broadcast** network. As the messages that are sent contain a unique ID associated to it, any node can receive the message if the message ID is known. This matching process is called **Filtering**. Now to accept/reject the incoming message, **Masking** is used basically to specify how much of the message, the node needs to take into account.

The Mask is set up in format containing 1's and 0's. A mask bit that is set to 1 means that the filter and message ID should match, whereas a mask bit set to 0 means that the message ID bit can be anything (0 or 1) (fig 3).

		Hex	Binary											
NODE 1	Message ID	354	0	0	1	1	0	1	0	1	0	1	0	0
NODE 2	Mask	FFC	1	1	1	1	1	1	1	1	1	1	0	0
NODE 2	Filter	354	0	0	1	1	0	1	0	1	0	1	X	X
	Accept	354	0	0	1	1	0	1	0	1	0	1	0	0
	Accept	355	0	0	1	1	0	1	0	1	0	1	0	1
	Accept	356	0	0	1	1	0	1	0	1	0	1	1	0
	Accept	357	0	0	1	1	0	1	0	1	0	1	1	1
	Reject	358	0	0	1	1	0	1	0	1	1	0	0	0
	Reject	359	0	0	1	1	0	1	0	1	1	0	0	1
	Reject	35A	0	0	1	1	0	1	0	1	1	0	1	0
	Reject	35B	0	0	1	1	0	1	0	1	1	0	1	1

Figure 3: An example of a node accepting/rejecting messages based on the CAN Mask

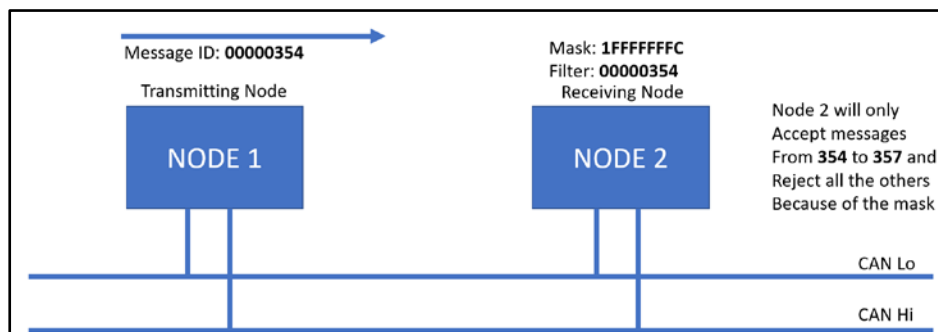


Figure 4: How the CAN network allows nodes to transmit messages

In the above diagrams, it can be seen how the filtering and masking process works. Node 2 in the end has the ability to accept messages that are in the format which is highlighted in fig 4. This is because the masking process is specifying which bits to accept and ignore when a message comes through.

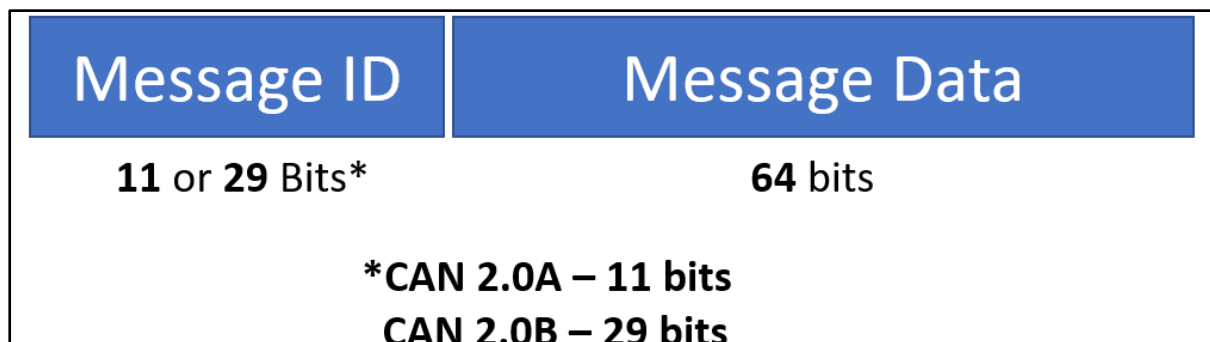
## CAN Messages

### Outgoing CAN Message

As part of the outgoing CAN message, the identifier (ID) section is **29 bits** long and formatted using hexadecimal. The ID starts with a 0 or 1, 0 being a message with higher priority than a message starting with 1.

For example: **1233AADC**

As part of the CAN message format, after the ID section follows the 'Data' section which is a total of **64 bits (8 bits)** (fig 5).



*Figure 5: Format of a CAN message*

**Byte 1:** Contains the Heater, Motor, Switch statuses

**Byte 2:** Contains keypad status

**Byte 3:** Contains temperature information

**Byte 4:** Unused

**Byte 5:** LED status

**Byte 6:** Unused

**Byte 7:** Unused

**Byte 8:** Unused

### Incoming CAN Message

The incoming message format for the CAN network message is basically the same as the outgoing one. This includes the fact that it must start with a 0 or a 1 and that the 'Data' section is **64 bits** long. The difference is that only, the first byte contains the data needed to read the remote components status- the rest of the bytes are just empty.

**Byte 1:** Heater, Motor, LED statuses

**Byte 2:** Unused

**Byte 3:** Unused

**Byte 4:** Unused

**Byte 5:** Unused

**Byte 6:** Unused

**Byte 7:** Unused

**Byte 8:** Unused

## SMS Format

The CAN board can deliver SMS messages to GSM phones via the built-in GSM modem. The SMS format has a limit of 160 characters per message. In 'Text' mode, an SMS message can allow 160 7-bit characters in 140 bytes.

I have designed a format for the content of the SMS message that displays the status of the remote board's components:



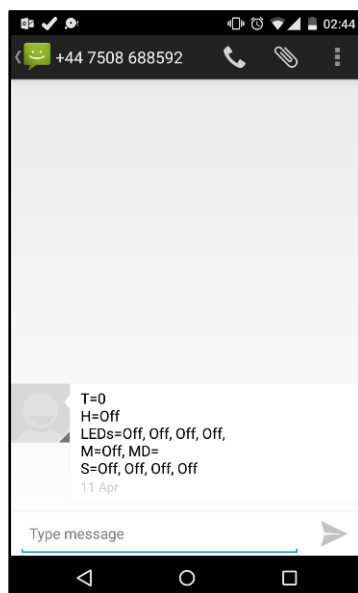
```
T=20  
H=Off  
LEDs=Off, Off, Off, Off,  
M=Off, MD=  
S=Off, Off, Off, Off
```

*Figure 6: My proposed SMS message format to display component statuses*

- T= Current Temperature of remote board in C°
- H= Heater ON/OFF status
- LEDs= LED ON/OFF status
- M= Motor ON/OFF status
- MD= Motor direction FORWARD/REVERSE status
- S=Switch ON/OFF status

A diagram can be seen below of how the output of the message looks on my mobile phone.

I have used 'Carriage Return' to enable a new line to be started as I wanted to separate the different statuses as part of a single SMS message.



*Figure 7: SMS message output on mobile device*

## Analysis

### Visual Aspect and Usability

I have created a visual layout using C# Windows Forms with which I have managed to fit in the required controls on one form. I have separated the form into two tables- one for the terminal that shows the output from the serial port and one for the main controls (figures 8 and 9). This way I can easily switch between the terminal view to see the raw CAN data and the controls tab to issue commands and read the status of the components. On the controls tab, I have logically grouped the controls using the 'GroupBox' tool, built into Visual Studio. This way, it is easier to get used to the layout as the controls and output fields have been arranged in an easy-to-use manner; the statuses of the components are on the left, the buttons have large, easy-to-read headings, and the input fields are clear to read and enter text into. I have also included messages that are displayed when there is an error or there is a confirmation message, for e.g. SMS message being sent successfully (fig 10). I have included a big, red 'Reset' button as that is used quite frequently when the board sometimes is not functioning fully.

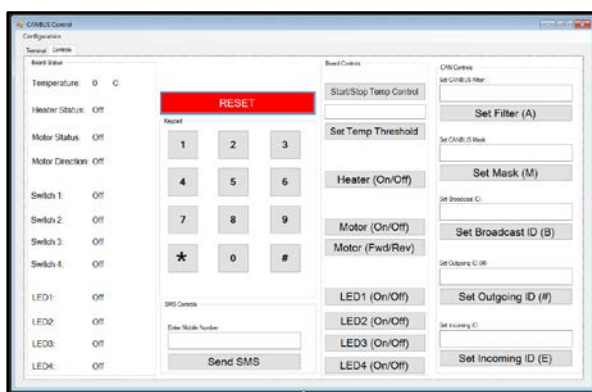


Figure 8: Main 'Control' Tab

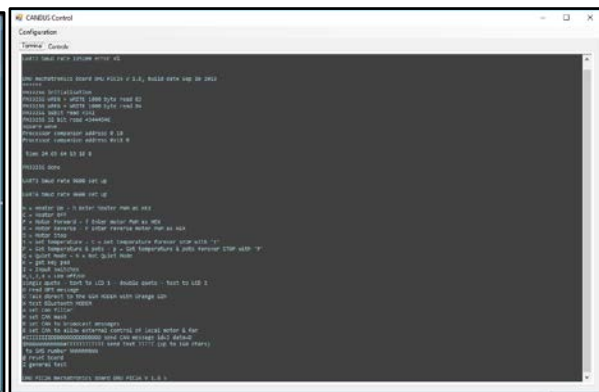


Figure 9: Terminal view

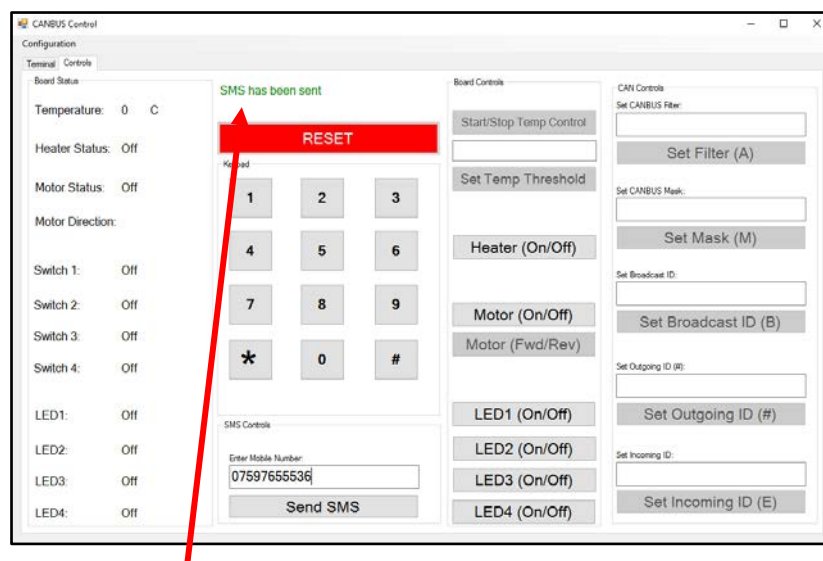


Figure 10: Status message stating that SMS has been sent successfully



## Technical Considerations

By making use of the 'Broadcast ID', 'Outgoing ID' and 'Incoming ID' mechanisms, the boards and the program by extension, is able to read the data that is transmitted across the CAN network. The data is received in the program in hexadecimal format. For example, in fig (11), one can see how the data is received in raw format into the board. So in order to read the data, I have created code that converts the data into a format which the programming language understands.

```
DMU PIC24 mechatronics board DMU PIC24 V 1.8 >
enter CANBUS input identifier for EXTERNAL motor control, etc (hex - 0xFFFFFFFF to disable) ?
1234AABC
DMU mechatronics board >RXB0 IDENT DATA
#1234AABC 00 00 1D 00 00 00 00 00
RXB0 IDENT DATA
#1234AABC 00 00 1D 00 00 00 00 00
RXB0 IDENT DATA
#1234AABC 00 00 1D 00 00 00 00 00
RXB0 IDENT DATA
#1234AABC 00 00 1D 00 00 00 00 00
RXB0 IDENT DATA
#1234AABC 00 00 1D 00 00 00 00 00
```

Figure 11: Raw CAN network output from terminal

For example, consider the code below:


```
1 reference | MohammedSheikh, 1 day ago | 1 author, 2 changes
void GetRemoteTemp(String[] data)
{
    /*temp*/
    for (int i = 0; i < data.Length; i++)
    {
        String byteThree = Convert.ToString(data[2]);
        String byteFour = Convert.ToString(data[3]);
        //array 2 - byte 3 contains information about the temperature.

        //here converting the array of hex into int
        tempByteOne = int.Parse(byteThree, System.Globalization.NumberStyles.HexNumber);
        tempByteTwo = int.Parse(byteFour, System.Globalization.NumberStyles.HexNumber);
        currentTemp = Convert.ToString(tempByteOne + "." + tempByteTwo);
        lblTemp.Text = currentTemp;
    }
}
```

Figure 11: Method that gets the remote temperature

With the for loop, I am looping through the entire line of data that is coming in, and acquiring the third byte and fourth bytes. The third byte represents the temperature whole number part (fig 12) and the fourth represents the temperature's decimal values (e.g. .4).

```
DMU mechatronics board >RXB0 IDENT DATA
#1234AABC 00 00 1D 00 00 00 00 00
```



The diagram shows two boxes labeled '3rd Byte' and '4th Byte'. Red arrows point from these boxes to the '1D' and '00' bytes in the CAN message data field.

Figure 12: CAN message bytes that represent temperature

So within the code, I am getting this data in hex format and parsing it so that the system can read it. Once that is done, I am able to then put the value into a label or textbox for example, so that it is displayed in the GUI.

The method that does this is called within the 'getCANbusData()' method, which is method that is running on a different thread. This way, the temperature is dynamically updating in the background even as I perform different operations.

In the same way, I am getting data regarding LED's, motor, switches by reading off the serial port output and parsing the hexadecimal data to be readable and manipulated by the GUI. As all the data is contained within the 8 bytes of the CAN message, I have created a global array variable that does this job.

```
int[] array = new int[7];
```

*Figure 13: Status message stating that SMS has been sent successfully*

## Automatic Temperature Control

Briefly speaking, the Automatic Temperature Control feature is one of the main features of my system. It allows the regulation of a remote board's temperature via the activation of the heater and motor of the remote board. A temperature threshold can be set where a user may want the remote board to hover around that temperature. For example, if one wants the remote board to hover around 30°C.

Technically, I have managed to implement the feature with the C# code by firstly acquiring the temperature of the remote board and then implementing Boolean IF statements that activate certain features depending on the temperature.

As seen below, the method that I have implemented acquires the line of data that a remote board is broadcasting, gets the bytes that represent temperature and depending on the threshold, controls the remote motor and heater depending on the current temperature.

```
//method to control temp
0 references | MohammedShaikh, 1 day ago | 1 author, 2 changes
void TempControl(String[] data)
{
    //temp control code that applies different behaviours depedning on temp

    for (int i = 1; i < data.Length; i++)

        if (tempSet == true)
        {
            //If current temp is lower than threshold, turn on heater

            if (tempThreshold > tempByteOne && !string.IsNullOrEmpty(txtSetTemp.Text))
            {
                //heater ON status method
                HeaterOn();
                lblSMSMessage.Text = "Temp is below threshold";
                RemoteCommand();
            }

            //If current temp is gretae than threshold, motor turns on
            else if (tempThreshold < tempByteOne)
            {
                //motor ON status method
                MotorOn();
                lblSMSMessage.Text = "Temp is above threshold";
                RemoteCommand();
            }
            //if same temp
            else if (tempThreshold == tempByteOne && !string.IsNullOrEmpty(txtSetTemp.Text))
            {
                lblSMSMessage.Text = "No threshold set";
                RemoteCommand();
            }
        }
    }
}
```

Figure 14: Method that regulates temperature of remote board

### Advantage(s)

- One advantage of this method is that once the threshold has been set, the temperature of the remote board does indeed stay pretty consistent around the threshold mark

### Disadvantage(s)

- As the temperature fluctuates quite rapidly, sometimes the switching on of the motor/heater isn't quite accurate as it could be.

**Alternative(s)**

Another way to possibly regulate remote temperature would be to monitor the temperature over a certain period of time and store this in some sort of logical storage mechanism e.g. an array. After a while, by making use of statistics, the temperature would be regulated at an efficient level as time goes on, and based on the temperature stats in the array, the software could be modified to allow more meticulous temperature control.

## SMS

The program allows the user to send an SMS message to a GSM device. Once the appropriate number has been entered and the 'Send' button is clicked, the message gets sent containing the status of the remote components. The integrated GSM modems within the CAN boards allow the transfer of SMS messages in 'Text' mode. This allows the user to send a maximum of **160 7** bit characters in a single message, although there is a **140** byte limit. These 160 characters can be made up of alphanumeric characters in any order.

In addition to the data, the SMS message also contains information regarding:

- Sender Information- The sender's number and the service centre's number
- Protocol Information- The protocol used to send the SMS
- Time Information- the date and time of the transmission of the message.

An SMS message can be transmitted in two ways:

- Text Mode
- Protocol Description Unit (PDU) Mode

### Advantage(s)

- An advantage of using SMS is that is by far the simplest way for a server to send a message to a GSM modem. This is because most GSM devices are not assigned a static IP address.
- Can be sent in different modes (Text or PDU)

### Disadvantage(s)

- Using 'Text' mode to send SMS does not provide proof of delivery of the message
- The character limit means that unlimited data cannot be sent in one message
- Speed of transmission is typically slower than Instant Messaging (IM), for example- this can be due to the inefficiencies of message centre's.

## **Alternative to SMS**

A popular alternative to SMS can be IM. This is a type of text messaging that works over the Internet. Now that millions across the World have access to the Internet, IM can be a decent alternative to SMS messaging.

### Advantage(s)

- Real-time data message transmission; much faster than SMS
- More interactive way of communication; for example, images/videos/attachments can be transmitted quickly

### Disadvantage(s)

- IM needs the Internet to function; some places don't have good Internet connection; therefore messaging may prove to be difficult, in comparison to SMS which is quite an easy-to-access technology
- In terms of pricing, some IM services may require additional subscriptions.

## Test Report

To enable robustness and ensure that the software is of the highest quality, testing is an important part of any development project and lifecycle. Therefore, below I have presented a table of test cases that show the results of my tests of the software features:

Test Case	Result
Display remote Motor ON status	PASS
Display remote Motor OFF status	PASS
Display remote Motor FORWARD direction status	PASS
Display remote Motor REVERSE direction status	PASS
Display remote Heater ON status	PASS
Display remote Heater OFF status	PASS
Display remote Switch 1 ON status	PASS
Display remote Switch 1 OFF status	PASS
Display remote Switch 2 ON status	PASS
Display remote Switch 2 OFF status	PASS
Display remote Switch 3 ON status	PASS
Display remote Switch 3 OFF status	PASS
Display remote Switch 4 ON status	PASS
Display remote Switch 4 OFF status	PASS
Display remote LED 1 ON status	PASS
Display remote LED 1 OFF status	PASS
Display remote LED 2 ON status	PASS
Display remote LED 2 OFF status	PASS
Display remote LED 3 ON status	PASS
Display remote LED 3 OFF status	PASS
Display remote LED 4 ON status	PASS
Display remote LED 4 OFF status	PASS
Display remote board current temperature	PASS
Display remote keypad status	PASS
Set local board CAN Filter	PASS

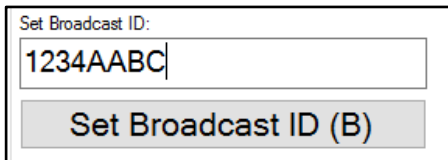
Set local board CAN Mask	PASS
Set Broadcast ID of local board	PASS
Send CAN message using Outgoing ID (# command)	PASS
Set Incoming ID of local board	PASS
Set temperature threshold	PASS
Start automatic temperature control feature to regulate remote board temperature, based on the threshold	PASS
Stop automatic temperature control feature to stop regulating remote board temperature	PASS
Manipulate remote board components when temperature threshold is ABOVE the current remote board temperature	PASS
Manipulate remote board components when temperature threshold is BELOW the current remote board temperature	PASS
Start remote motor manually	PASS
Stop remote motor manually	PASS
Set remote motor FORWARD direction	PASS
Set remote motor REVERSE direction	PASS
Start remote heater manually	PASS
Stop remote heater manually	PASS
Start remote LED 1 manually	PASS
Stop remote LED 1 manually	PASS
Start remote LED 2 manually	PASS
Stop remote LED 2 manually	PASS
Start remote LED 3 manually	PASS
Stop remote LED 3 manually	PASS
Start remote LED 4 manually	PASS
Stop remote LED 4 manually	PASS
Reset local board	PASS
Send SMS message to GSM phone displaying status of remote board components	PASS

## User Guide

### Receiving and sending messages on the CAN network

I will be referring to the local board as **Board A** and the remote board as **Board B**.

1. Firstly, **Board B** must be set up to broadcast data (which contains the board's current temperature) to the CAN network. Within the GUI, this is done by entering a broadcast identifier in the 'Broadcast ID' field. An example of an identifier could be **1234AABC**.

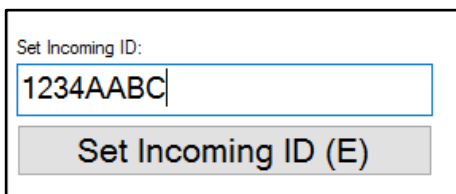


Set Broadcast ID:

1234AABC

Set Broadcast ID (B)

2. **Board A** then needs to accept this incoming data from the CAN network. This identifier must match the identifier stated in Step 1. This is done by entering this into the 'Incoming ID' field. Once this is entered, the local board will be able to receive the data sent by **Board B**. This data could include temperature status, LEDs status etc. Once this is done, **Board B** can send messages to **Board A** which contain relevant data

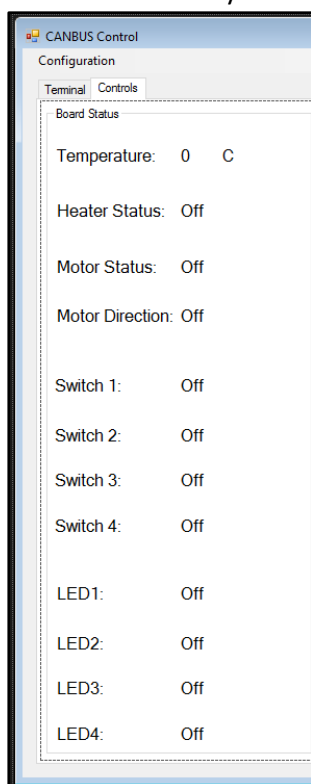


Set Incoming ID:

1234AABC

Set Incoming ID (E)

3. I have programmed the GUI so that the relevant data received can easily be read by the user. On the left-hand side of the GUI in the 'controls' tab, all the components status of **Board B** can be displayed in real-time. These include temperature status, switch status, motor status etc. It is quite easy to interpret the data as on the GUI it clearly states the status (e.g. ON/OFF) of a component.



CANBUS Control

Configuration

Terminal Controls

Board Status

Temperature: 0 C

Heater Status: Off

Motor Status: Off

Motor Direction: Off

Switch 1: Off

Switch 2: Off

Switch 3: Off

Switch 4: Off

LED1: Off

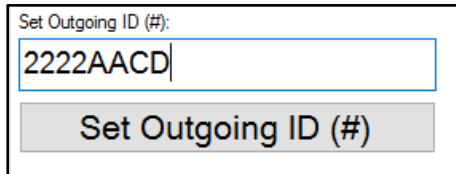
LED2: Off

LED3: Off

LED4: Off



- Another way to send a message from the board is to use the 'Outgoing ID' field on the GUI to send a single CAN message across the network (rather than broadcast messages continuously). This message can also contain data that basically can control a remote component. An example of the 'Outgoing ID' could be **2222AACD**. As long as the receiving board has completed 1 Step 2, the message should go through.



Set Outgoing ID (#):

2222AACD

Set Outgoing ID (#)

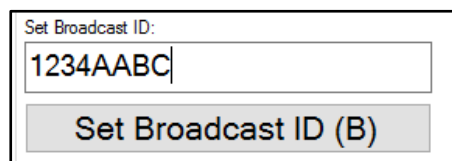
### How to send commands on the CAN network

If one wants **Board B** to be controlled by **Board A**, once **Board B** has performed Step 2 from the above command (by setting up 'Incoming ID'), then any other board that performs Step 4 may then be able to remotely send commands to **Board B**'s components. By using the GUI, this is very easy; all of the buttons are self-explanatory and one can easily send commands using these buttons.

### How to use Automatic Temperature Control

To use the Automatic Temperature Control feature, a connection with the remote board must be established. Once it is established, the local board can then regulate the temperature of the remote board by controlling the remote board components. I will be referring to the local board as **Board A** and the remote board as **Board B**.

- Firstly, **Board B** must be set up to broadcast data (which contains the board's current temperature) to the CAN network. Within the GUI, this is done by entering a broadcast identifier in the 'Broadcast ID' field, within the 'CAN Controls' section. An example of an identifier could be **1234AABC**.

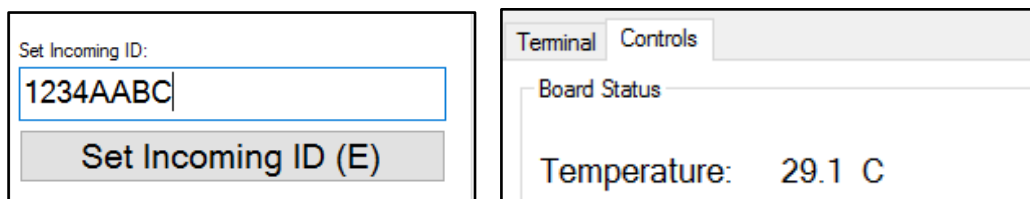


Set Broadcast ID:

1234AABC

Set Broadcast ID (B)

- Board A** then needs to accept this incoming data from the CAN network. This identifier must match the identifier stated in Step 1. This is done by entering this into the 'Incoming ID' field. Once this is entered, the GUI which is receiving the data will start displaying the current temperature (in °C).



Set Incoming ID:

1234AABC

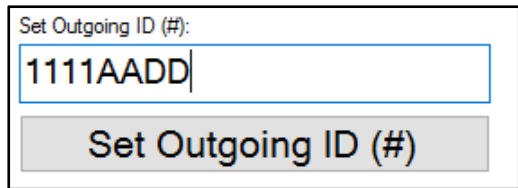
Set Incoming ID (E)

Terminal Controls

Board Status

Temperature: 29.1 C

3. For the GUI connected to **Board A** to control **Board B**'s remote components for temperature regulation, **Board B** needs to allow **Board A** to control it. For this, the 'Outgoing ID' field must be entered in **Board A**'s GUI.

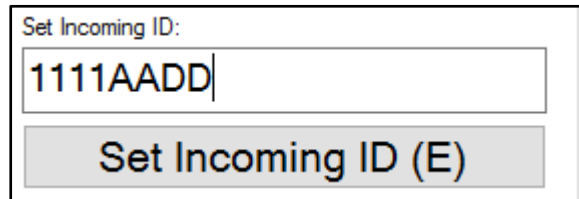


Set Outgoing ID (#):

1111AADD

Set Outgoing ID (#)

4. The 'Incoming ID' must be entered in **Board B**'s GUI- this must match the 'Outgoing ID' entered in Step 3

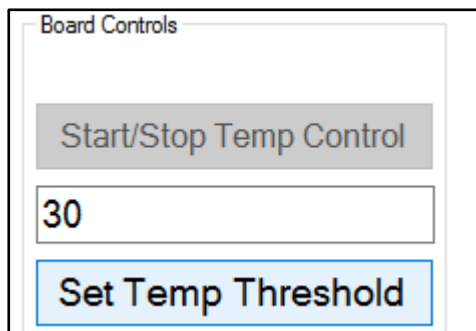


Set Incoming ID:

1111AADD

Set Incoming ID (E)

5. Now that **Board A** is receiving **Board B**'s temperature stats, a temperature threshold can be set which the temperature control feature will use to control the components. Enter the desired temperature threshold in this field and select 'Set Temp Threshold' button. In order to manually start regulating the temperature select the button above.



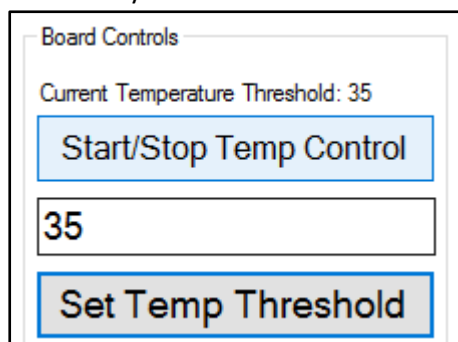
Board Controls

Start/Stop Temp Control

30

Set Temp Threshold

6. Depending on the threshold, **Board B**'s motor or heater should turn on- if the current temperature is ABOVE the threshold, the motor will turn on, else the heater will turn on. Gradually, the temperature should stay at a consistent level due to the two components being activated.



Board Controls

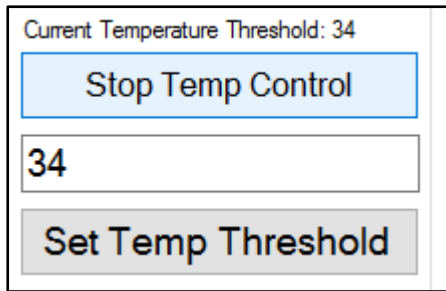
Current Temperature Threshold: 35

Start/Stop Temp Control

35

Set Temp Threshold

7. Once the temperature control has been started, there is an option to stop it too by pressing the 'Stop Temp Control' button.



Current Temperature Threshold: 34

Stop Temp Control

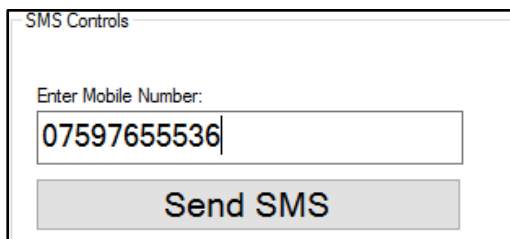
34

Set Temp Threshold

### How to send SMS

One of the simpler features to use is the SMS message feature. The purpose of this feature is to send a live status update of the remote board components to a GSM device in the form of an SMS text message. As long as the phone number is valid and the correct number of digits (11) have been entered, the SMS message should be sent successfully:

1. Type in SMS number to send message and press 'Send SMS'



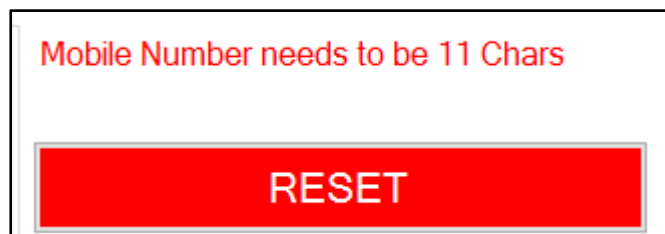
SMS Controls

Enter Mobile Number:

07597655536

Send SMS

2. The validation will fail if the entered number is invalid in terms of characters (must be 11 digits)



Mobile Number needs to be 11 Chars

RESET

3. This is how the status data looks in the form of the SMS message

