ARDUINO BLUETOOTH COMMUNICATION

COMMS F

EXPLAINATION OF THE LIBRARY AND FUNCTIONS

Contents

[Initialize 3](#_Toc526535706)

[UnoBlueTooth() 3](#_Toc526535707)

[begin() 3](#_Toc526535708)

[connect() 3](#_Toc526535709)

[Communication 3](#_Toc526535710)

[sendData() 3](#_Toc526535711)

[sendArray() 3](#_Toc526535712)

[receiveData() 3](#_Toc526535713)

[addMarker() 4](#_Toc526535714)

[removeMarker() 4](#_Toc526535715)

[Markers 4](#_Toc526535716)

[Settings and Status 5](#_Toc526535717)

[changeName() 5](#_Toc526535718)

[changeRole() 5](#_Toc526535719)

[getConnectionStatus() 5](#_Toc526535720)

[Testing Functionality 5](#_Toc526535721)

[readFromSerialTobT() 5](#_Toc526535722)

[readFromBlueTooth() 6](#_Toc526535723)

[readArray() 6](#_Toc526535724)

[Globals 6](#_Toc526535725)

[AltSoftSerial BTSerial 6](#_Toc526535726)

[connectionStatusPin 6](#_Toc526535727)

[MegaMAC 6](#_Toc526535728)

[Security 7](#_Toc526535729)

[encrypt() 7](#_Toc526535730)

[decrypt() 7](#_Toc526535731)

[Detection/Correction 7](#_Toc526535732)

[addCheckSum() 7](#_Toc526535733)

[removeCheckSum() 7](#_Toc526535734)

[readCheckSum() 7](#_Toc526535735)

[Other 7](#_Toc526535736)

[doATCommands() 7](#_Toc526535737)

[atResponse() 7](#_Toc526535738)

# Initialize

## UnoBlueTooth()

Constructor.

Here you can initialize some variables, but it is better practice to do so in a custom begin/initialization method.

## begin()

This function initializes the default aspects of the class.

This includes the ‘begin’ function being called for all the necessary serial ports, Serial and BTSerial (custom name for AltSoftSerial) which are hardcoded to ports 8 and 9.

Here we also call doATCommandSetup to change the settings only available before the BlueTooth is paired.

## connect()

Perform AT command to connect to the Mega’s BlueTooth, using the MAC address stored within the library.

# Communication

## sendData()

This function handles sending a single String.

If the message is successfully sent – i.e. the Mega acknowledges receiving the message – this function returns true, else false.

The sending process should include functions and manipulation to the data in the given order:

1. Adding marker
2. Encrypting data
3. Adding headers
4. Adding checksum bits
5. Transmit

## sendArray()

Similar to sendData but instead is capable of processing multiple lines / array of Strings.

Depending on our discussion and agreement with the Uno web server backend team and the Mega drive base team, only one send function may be required to be implemented.

## receiveData()

Handles reading from the buffer. It should be capable of identifying and omitting random noise (use markers for this)

This function should be the inverse of the send functions, performing manipulation to the received message in the following order:

1. Read checksum bit to confirm message is intact.
2. Reading and extracting information from the headers then removing them.
3. Decrypting the data
4. Removing the markers
5. Rebuilding structure if necessary
6. Passing data to relevant code

## addMarker()

## removeMarker()

## Markers

This is where you use pre-defined symbols – e.g. @ # $ > { - to act as a mark/reference point for reading purposes.

Examples.

|  |  |
| --- | --- |
| **Symbol** | **Purpose** |
| < | Start of message |
| > | End of message |
| $ | Start of line |
| % | End of line |

Initial message to be transmitted:

0 Hello

1 Blue: 1

2 Red: 3

3 Green: 2”

After adding markers:

0 <$Hello%

1 $Blue: 1%

2 $Red: 3%

3 $Green: 2%>

Converted to serial data (of type byte), it will have the equivalent structure:

<$Hello%$Blue:1%$Red:3%$Green:2%>

If there is leftover data from errors or the Bluetooth picks up static, the message may look like:

@@@@<$Hello%11111$Blue:1%2222222$Red:3%333333$Green:2%>!!!!!

Since you know the markers, you can ignore everything that is not in between the correct pair of markers. In this case, you are able to omit the @, 111, 222, 333, !!!

Difference between having vs not having markers

|  |  |
| --- | --- |
| **Using marker** | **Not using marker** |
| Hello | @@@@Hello111 |
| Blue: 1 | 111Blue: 1222 |
| Red: 3 | 222Red: 333 |
| Green: 2 | 333Green: 2!!!!!! |
|  |  |

# Settings and Status

## changeName()

AT command to change the name of the Bluetooth module. Name is defined by a String parameter passed to this function. Length limit of the name is ???? refer to datasheet on iLearn.

## changeRole()

AT command to change the role of the Bluetooth module. Role is either master or slave, defining whether the module can perform certain AT commands which includes initiating connection/pairing.

Role is defined by the integer parameter. 1 = master. 0 = slave.

## getConnectionStatus()

Reads from the Digital pin ‘connectionStatusPin’.

Polls the pin multiple times over a short period. If the output from this pin is HIGH for every polling, the Bluetooth is on. If the output is a mix of HIGH and LOW, then it is ‘blinking’ and not paired.

Returns 1 if paired

Returns 0 if not paired.

# Testing Functionality

These functions are adaptations from the code provided on iLearn (by Rex). They are used to test the if other functions are working correctly, whether the data is correctly sent/received/processed.

The people working on sending and receiving data should take a look at this as a reference point for developing their methods.

## readFromSerialTobT()

Checks if the serial monitor has anything in the buffer.

If the buffer contains anything, read from it one character at a time and store it into a variable.

Transmit the stored character, and then print it to the serial monitor/display for the user to see

This method does not check for unintended data/input and can contain random stuff from whatever errors may occur, nor does it guarantee anything.

DO NOT ASSUME THAT THIS IS GOOD ENOUGH TO USE FOR FINAL SOLUTION.

## readFromBlueTooth()

Check the Bluetooth buffer for data. If there is anything, it will read it character by character and print it to the serial monitor/display for the user to see.

Like the above, it is only for testing purposes. It is not reliable. It does not guarantee anything.

DO NOT ASSUME THAT THIS IS GOOD ENOUGH TO USE FOR FINAL SOLUTION.

## readArray()

Reads an array and prints it line by line. Any elements will print out “empty”.

It requires 2 parameters, an array and the size of the array.

This method was intended to test passing various types of data, including Strings, Arrays, Points, or other data structures.

# Globals

## AltSoftSerial BTSerial

An instance of the AltSoftSerial library, used for the Bluetooth module. It transmits using the Digital 8 pin, and receives on the Digital 9 pin.

Refer to the following links for more information.

<https://www.pjrc.com/teensy/td_libs_AltSoftSerial.html>

<https://github.com/PaulStoffregen/AltSoftSerial>

## connectionStatusPin

Global variable from which we read the voltage output from the BT module’s/HM-10’s STATE pin.

When the pin is HIGH (1), the Bluetooth module is paired with another device.

When the pin is alternating between HIGH(1) and LOW (0) – BLINKING – the Bluetooth module is not paired.

## MegaMAC

MAC address of the Bluetooth module connected to the Arduino Mega.

# Security

## encrypt()

## decrypt()

# Detection/Correction

## addCheckSum()

Adds additional information to every byte - checksum. The additional information is relative to the byte so that the receiver is able to compare the byte itself to the checksum, if the two do not match then the data has been compromised.

Example: <https://github.com/bakercp/CRC32>

## removeCheckSum()

## readCheckSum()

# Other

## doATCommands()

## atResponse()

Listens for a response on the serial monitor.

Read from the buffer, stores it into a single string.

Passes the final string back.