

Multi Comm Terminal

Hobbyist Terminal Program to communicate with Embedded systems

By Michael Roop

Table of Contents

Introduction	4
Supported Mediums	4
Data Storage.....	5
Lists of Commands	5
Terminators.....	5
WIFI Credentials.....	5
Languages	6
Settings.....	6
User Interface	6
Buttons.....	6
Wordless Buttons.....	6
Main Window.....	8
Elements:	8
Drop Down Menu.....	9
Language	10
Terminators.....	10
Commands	14
Credentials	16
About.....	17
Document Button	18
Bluetooth Classic Connections.....	19
Discovery.....	19
Detailed Info.....	20
Unparing.....	21
Connection	21
Pairing	22
WIFI Connections	22
Discovery.....	22
Connection.....	23
Bluetooth Low Energy (BLE) Connections.....	23
Discovery.....	24
Detailed info Gather	24

Connection.....	26
Appendix	27
Introduction to Embedded Messaging	27
Arduino Bluetooth Sample.....	27
Arduino BLE Sample	31
Arduino WIFI Sample	35

Introduction

This program will allow users to communicate with embedded devices such as Arduinos, Raspberry Pie, and others by multiple communication mediums.

Supported Mediums

- Bluetooth Classic:
 - Connects to a device which implements a Bluetooth serial access point.
 - Tested against an Arduino Uno with an Itte Bluetooth shield using an HC-05 module.
 - An Arduino code sample is provided
- WIFI:
 - Connects to a device which implements a WIFI Access Point.
 - The access point shows up as an SSID like any router.
 - The device also must provide a socket address and port.
 - Tested against an Arduino WIFI Rev2.
 - An Arduino code sample is provided.
- BLE (Bluetooth Low Power):
 - Connects to a device which implements BLE Characteristics for inputs and outputs.
 - As such it emulates a serial input output connection
 - Tested against an Arduino WIFI Rev2.
 - This was initially working with Arduino:
 - A Windows update changed connection requirement to require pairing.
 - Since Arduino does not support BLE pairing I removed connection functionality.
 - You can still get detailed info from devices that support the pairing requirement
 - When I am able to find an embedded device that supports the requirement for testing, I will enable connection.
 - An Arduino code sample is provided.
- USB:
 - Connection via hard wired USB port
 - Some initial work already done
 - Will be released in the future
- Ethernet
 - Connection via hard wired Ethernet cable
 - Will be released in the future

Data Storage

Data storage root depends if running as a Windows Application or a Win App Desktop Application (From the App Store)

- Windows Desktop Application: C:\Users\UserName\AppData\Local\MultiCommSerialTerminal
- Win App:
C:\Users\Michael\UserName\Local\Packages\LongGUIDSstring\LocalCache\Local\MultiCommSerialTerminal

An example of the long GUID string would be: *1c29e46d-d1d0-4f21-acbf-d63f3611c3a2_k00brw8eakxmj*

Data is stored in various directories under that root. The only time you might ever want to access it directly is to view the debug logs to report errors. In the future a log reader will be added for convenience.

All contents are in plain text in JSON format. Except for the credentials files which are encrypted. Do NOT change the file names as you will lose access to the data.

These are the current sub directories

- Logs: Up to 10 log files that contain data from running the App
- Scripts: Command sets
- Settings: Current App Settings
- WifiCredentials: WIFI connection credentials and parameters

Lists of Commands

The user can create, store, and retrieve lists of ASCII based messages. The messages can be sent to the embedded device once connection is establish. The embedded device can interpret those character strings as a command to launch certain actions. This allows the user to control the device remotely. For more information see the Intro to embedded messaging in the appendix

Terminators

The user can create, store, and retrieve end of message terminator sequences. The terminator(s) will be automatically added to the end of the outgoing message. The terminators are non-printable characters which tell the embedded device that the full message has been received. The device must also add those terminators to the end of any response or other synchronous messages from the device.

WIFI Credentials

On first connection, the WIFI password, as well as the host name (or IP address) and service port are entered and saved if the connection is successful. There is functionality to go back and edit or delete those stored credentials. They are encrypted

Languages

The App currently supports 6 languages in most areas. Supported:

- English
- French
- Spanish
- Chinese (Simplified)
- Japanese
- Korean

I followed a simple approach of single word descriptions for headers and buttons. I used the Microsoft Technical Language Portal to get the translation. See: <https://www.microsoft.com/en-us/language/Search?&searchTerm=Automatic&langID=303&Source=true&productid=0>. This allows for a high precision in rendering of the languages since it mostly avoids idioms.

Settings

The file contains, among other things, currently selected preferences so that the App starts with the same selections as last time. Currently those preferences are Language, Terminators, Command list. More will be added over time

User Interface

Buttons



Buttons have both an icon and text. So, if you inadvertently switch to a language you do not understand you can still navigate

Wordless Buttons

These are small and round and used throughout. The usage should be self-evident

Add: Add a new Item

View: Show an existing Item



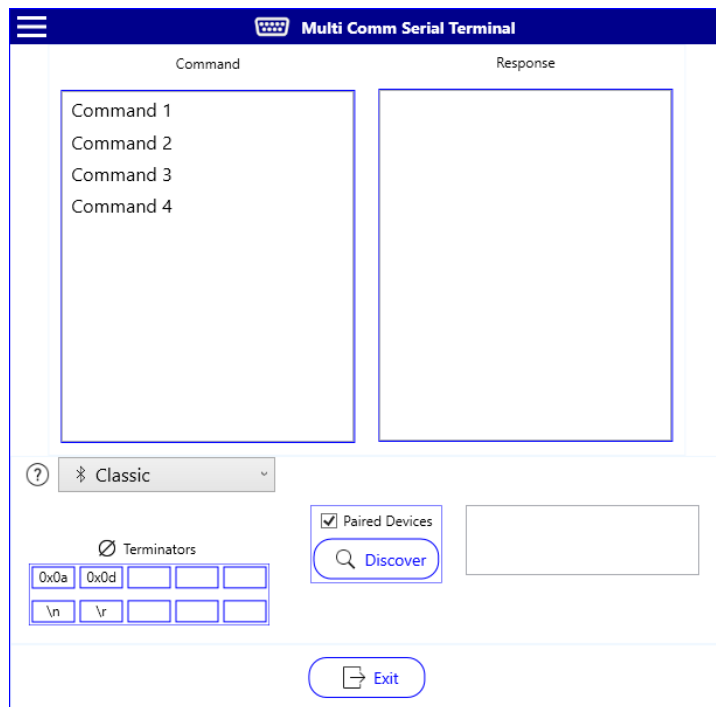
Edit: Edit and existing item



Delete: Remove an item



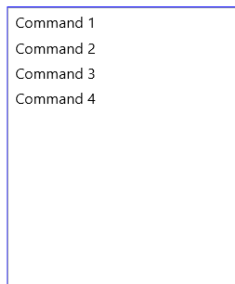
Main Window



Elements:



Command



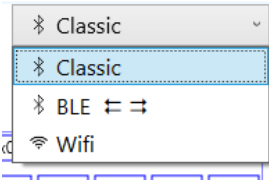
Response



'Hamburger' icon on title bar. Click on it to open drop down menu where you can set language, etc.

Top left is the command window. You see the message name rather than the outgoing message

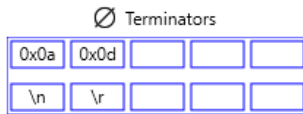
Top Right is the response window. You see the messages coming from the embedded device



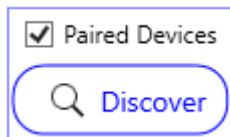
Below the Command list is a drop-down list of the different communication mediums



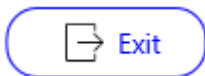
To the left of the Mediums drop down list is the info button. Clicking on it brings up a list of documentation, including code samples



Below the combo box is the currently selected terminators which are appended to outgoing messages. The embedded device must also add those terminators to its message bound for the App. Clicking on it will bring up the Terminator list editor where you can add, edit, delete sets



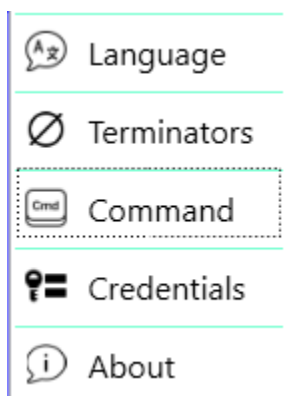
To the right of the Terminators is the button set for the currently selected medium. The number and type of buttons will depend on the medium and what state it is in. To the right will be a list box with the discovered devices



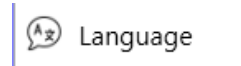
The Exit button will terminate the App

Drop Down Menu

The Hamburger icon brings up a menu



Language

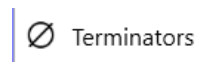


Clicking on this will bring up the Language selector. Currently there are 6 languages supported for UI elements



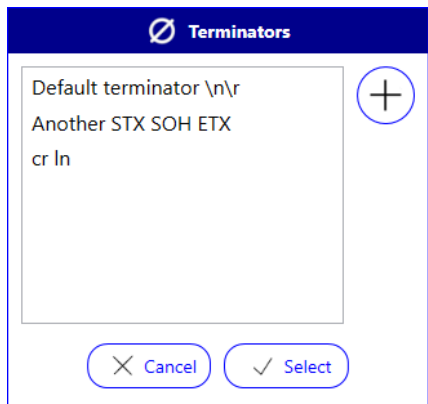
Highlight the requested language and you will see all the buttons change. Click Save to make that the current language. It will be saved as your preference for future sessions. Click on Cancel to revert to previous language

Terminators

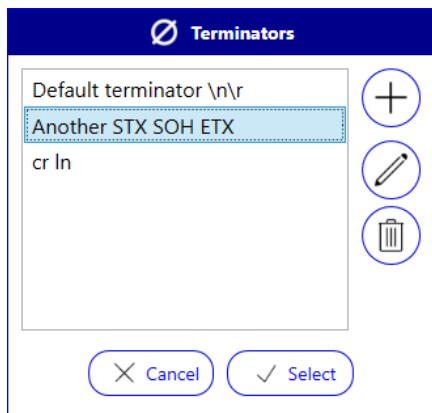


Clicking on this brings up a dialog box with currently stored terminator sequence.

- A terminator sequence contains one or more nonprintable characters that are added to the end of the outgoing message.
- The embedded device must be expecting that sequence to determine when an incoming message is complete.
- The embedded device must also all those terminators to its outgoing messages since the App is expecting those to determine when the incoming message is complete

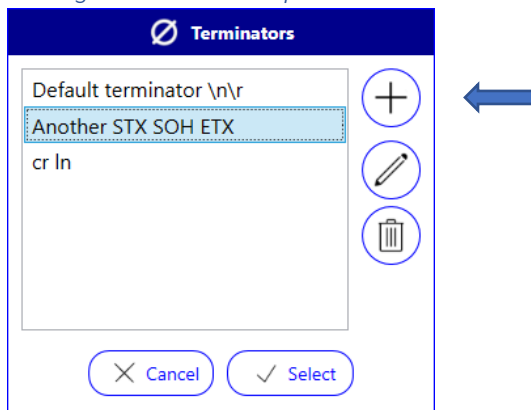


On Open, only the Add button is visible.

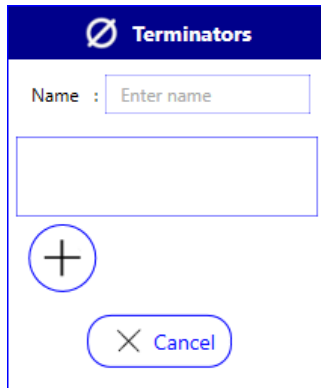


When you select a terminator set, the edit and delete button will appear. If you click on Select, the Dialog will close, the selected terminator sequence is saved, and the display on the Main Window is changed.

Adding a terminator sequence set



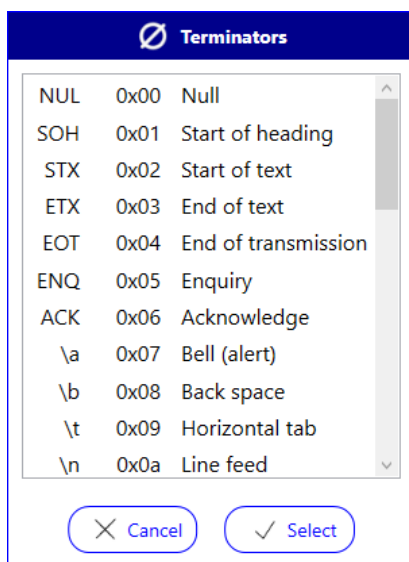
In the Terminator set dialog, click on the Add button and a Dialog appears to create a new set



Terminators

Name :

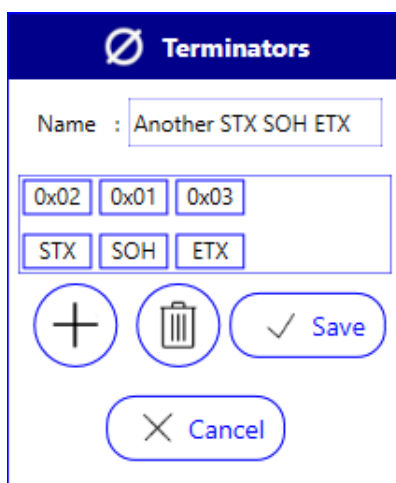
Enter the name of the set so you can identify it in the future. Click on the Add button to add a terminator character. You must do these in the sequence that is expected on the embedded device.



Terminators

NUL	0x00	Null
SOH	0x01	Start of heading
STX	0x02	Start of text
ETX	0x03	End of text
EOT	0x04	End of transmission
ENQ	0x05	Enquiry
ACK	0x06	Acknowledge
\a	0x07	Bell (alert)
\b	0x08	Back space
\t	0x09	Horizontal tab
\n	0x0a	Line feed

Select the desired terminator character from the list and Click Select to add, or Cancel to exit without adding one. Do this as many times as required

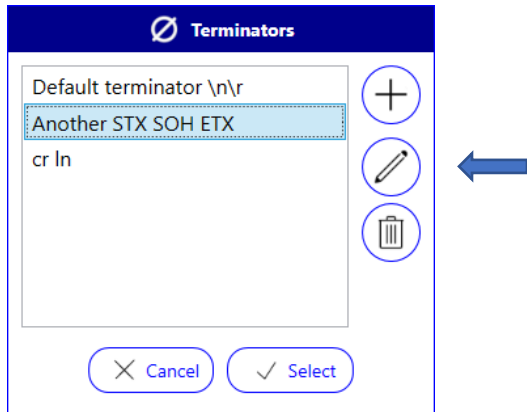


Terminators

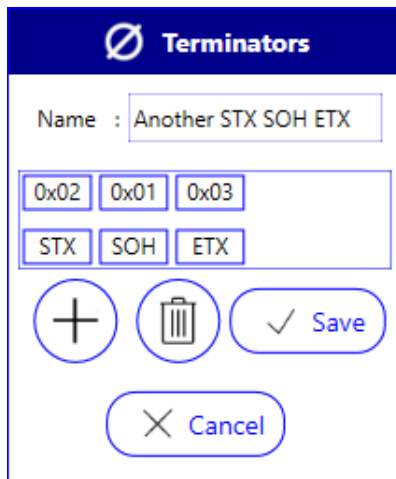
Name :

Each terminator will be added to the set. You can continue to add terminators or delete existing terminators. When you are finished, click Save to store the set. Note that you can only delete or add at the end. So, if you change your mind for a middle terminator you must delete all the following until you get to it. Delete that one then continue adding

Edit Terminator set



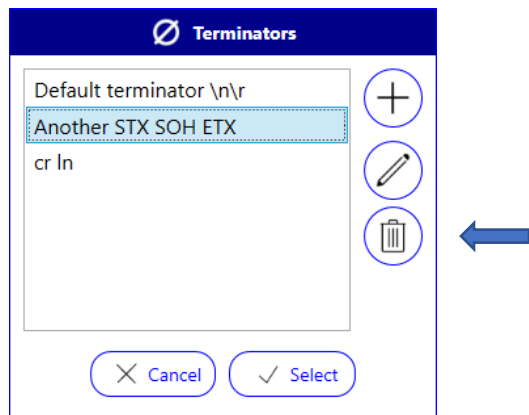
Back at the Terminator set list dialog, click on the Edit button for the editor dialog



You follow the same process as adding or deleting terminators in a new set

Deleting a Terminator set

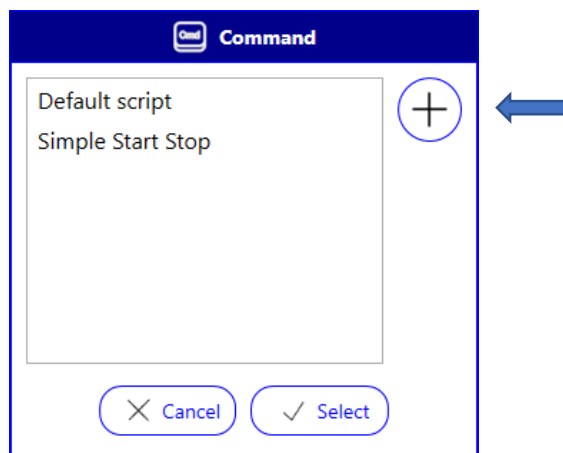
Back at the Terminator set list dialog, click on the Delete button to delete the selected set



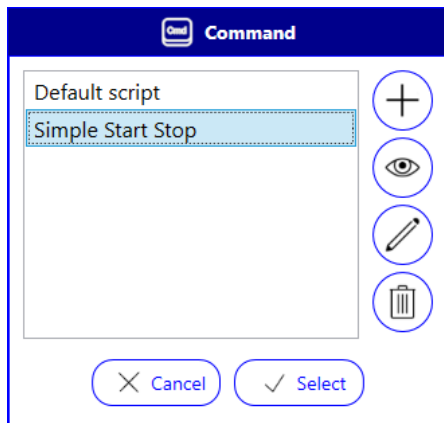
Commands



The Commands list displays a set of existing command sets that are stored.

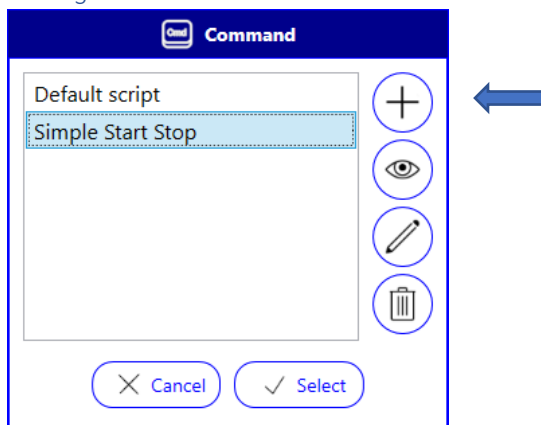


You can add a new Command set by clicking on the Add button or select an existing to bring up more options.

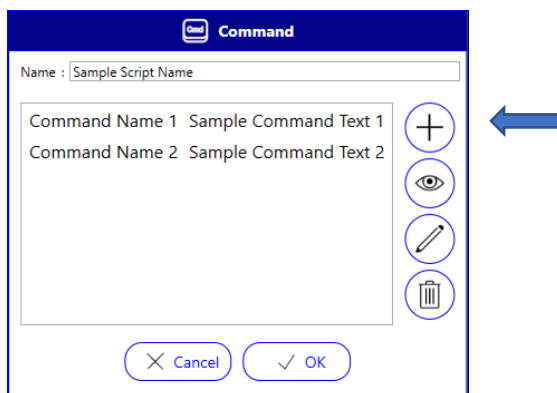


You can just click on Select to select the new set and exit. The change will show up in the commands list on the main window

Adding a command list

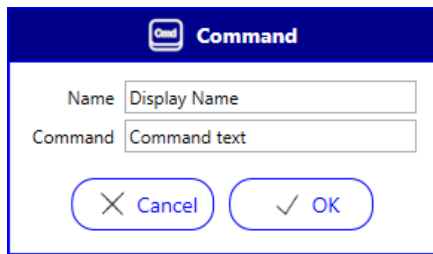


Click on the Add button to bring up the editor with a sample name and entries

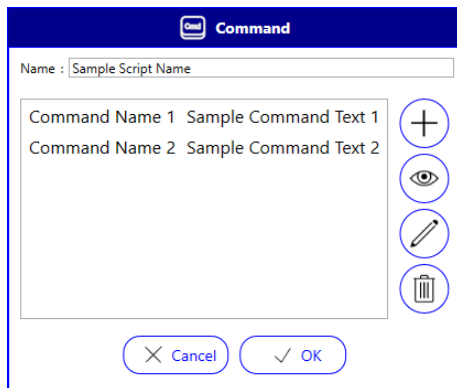


Edit the name to your liking. Click on the Delete button to remove the sample entries (or edit them to your liking).

Click on the Add button to create a new command

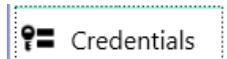
A dialog box titled "Command" with a blue header bar. It contains two text input fields: "Name" with the text "Display Name" and "Command" with the text "Command text". Below the fields are two buttons: "Cancel" with a red X icon and "OK" with a green checkmark icon.

Enter the name that will be displayed for this command. It will show up on the list. The Command text (message) is what is sent to the embedded device. It must be exactly what the device is expecting. Terminators are added apart from this text

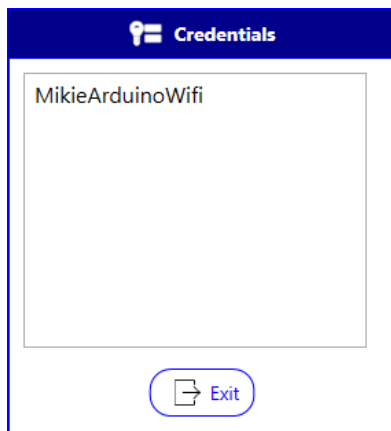
A dialog box titled "Command" with a blue header bar. It has a "Name" field with the text "Sample Script Name". Below it is a list of commands. The first two are "Command Name 1 Sample Command Text 1" and "Command Name 2 Sample Command Text 2". To the right of the list are four circular icons: a plus sign (+), an eye, a pencil, and a trash can. At the bottom are "Cancel" and "OK" buttons.

You can click on the view button to see a command, the edit button to change a command, or the delete button to remove it. Click on OK or Cancel

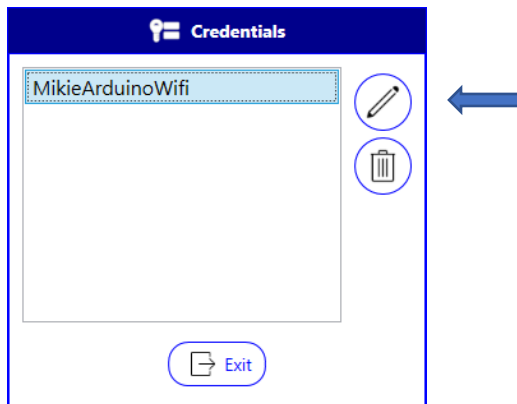
Credentials



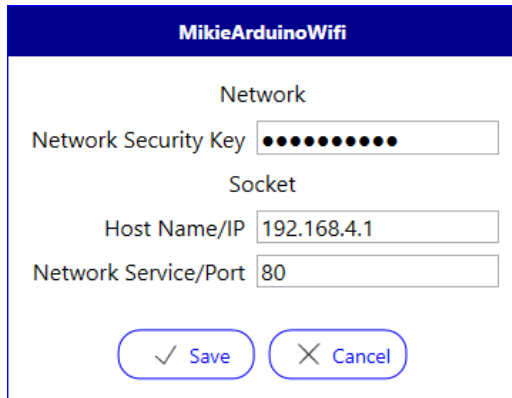
The Credentials dialog provides a way to edit or delete credentials and connection parameters for a previous WIFI connection. Clicking on this will open a dialog with a list of stored credentials

A dialog box titled "Credentials" with a blue header bar. It contains a list of credentials, with "MikieArduinoWifi" visible. At the bottom is an "Exit" button with a door icon.

When you select a credential from the list you are given the option of edit or delete



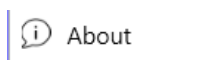
Click on edit to open the editor. The list has the SSID of the WIFI



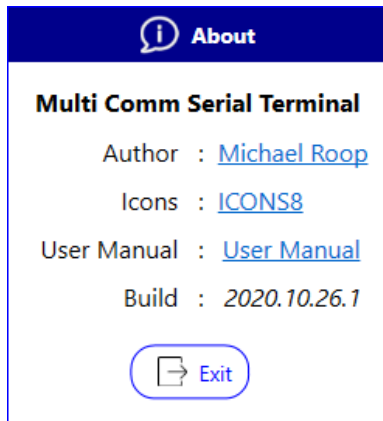
The security key is the password for the WIFI access point (like a router). The Host name/IP or those that you have exposed on your embedded device. See the Arduino WIFI code sample in the appendix.

Click on Save to store your changes, or cancel to discard

About



The About dialog has information on author, build and other things



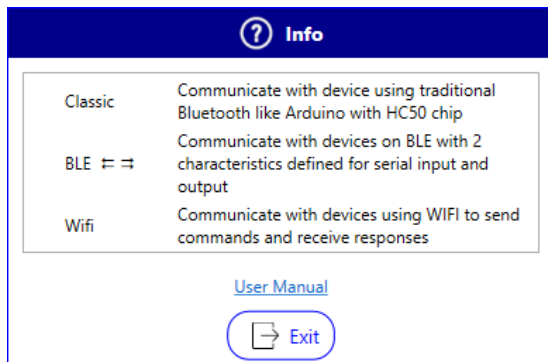
- The Author link directs you to my Linked in Profile in case you want to send feedback.
- The Icons link is to the site that provided the excellent icons
- The User Manual link will open the user manual PDF file in the browser

Document Button

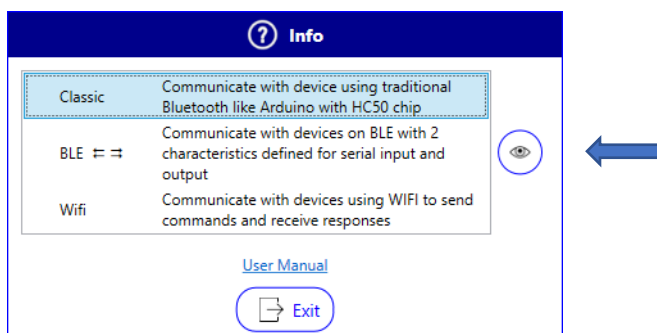
To the left of the Communication Mediums drop down you will find the '?' button



Click on it to bring up a document selection Dialog



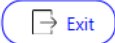

Select any one of those and the View button will appear.



Click on the **View** button to get related code samples or documentation.

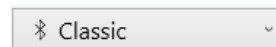
Classic

```
//-----  
// Name:      ArduinoBluetoothDataTests.ino  
// Created:   12/12/2019 3:14:05 PM  
// Author:    Michael  
//  
// Sample to send and receive data to and from Arduino Bluetooth shield  
//  
// Written and tested in Visual Studio using Visual Micro  
// Tested against Arduino Uno and itea Bluetooth shield with HC-05 module  
//  
// MUST HAVE BOARD TOGGLED TO DATA  
// MUST HAVE BOARD JUMPERS SET FOLLOWING for IO4 TX and IO5 RX  
// ****  
// |  
// *****  
//  
// MUST HAVE DEBUG SERIAL SET TO 9600 Baud  
// MUST HAVE BT SERIAL SET TO 38400 Baud  
//  
// Must pair with Bluetooth first. The Multi Comm Terminal provides that  
// functionality  
//-----  
#include <SoftwareSerial.h>  
  
#ifndef SECTION_DATA  
int i = 0;  
// The jumpers on BT board are set to 4TX and 5RX.  
// They are reversed on serial since RX from BT gets TX to serial  
SoftwareSerial btSerial(5, 4); //RX,TX  
bool hasInput = false;  
#define IN_BUFF_SIZE 100  
char buff[IN_BUFF_SIZE];  
unsigned char inIndex = 0;  
#endif // !SECTION_DATA
```

 Exit  Copy

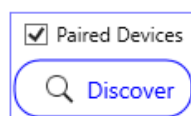
Bluetooth Classic Connections

The Bluetooth medium connects via an RfComm connection to the embedded device. This is provided on Arduinos with modules such as HC-05. To start, select Bluetooth Classic from the drop down

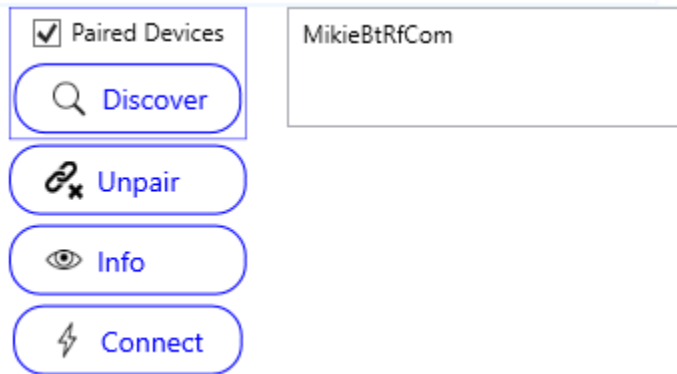


Discovery

You can see a list of the devices by clicking on the Discover button.



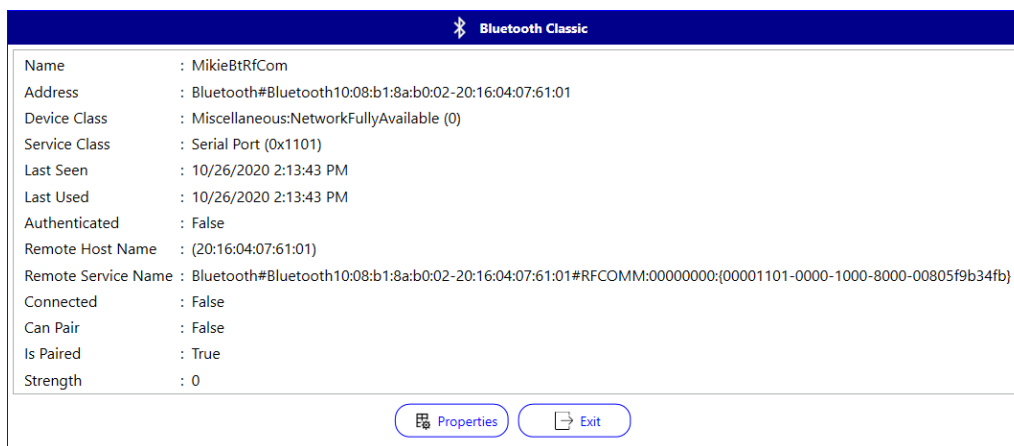
If the Paired Devices is checked, you will get a list of previously paired devices. This is very fast but the device itself may not be available. The list is populated from information stored the computer's OS. The paired devices will show up in the list box to the right



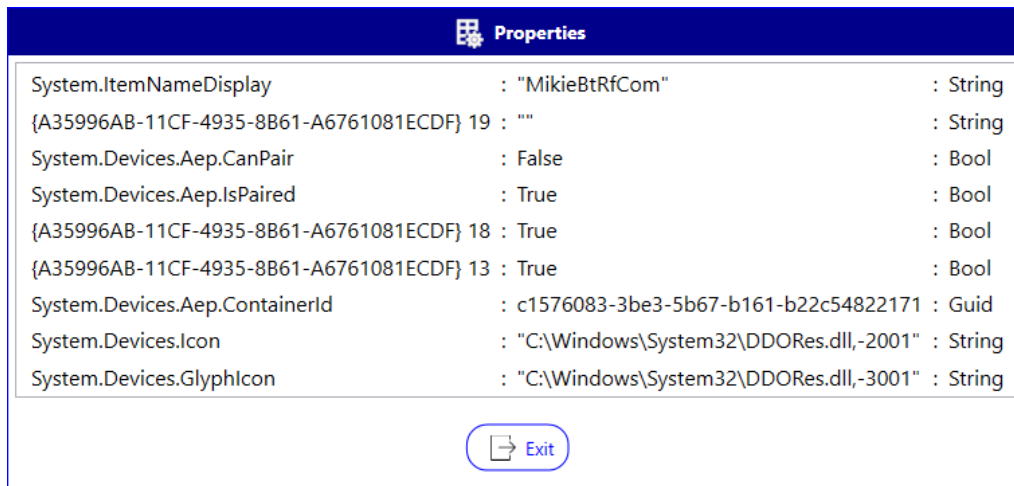
Extra buttons will appear. The *Unpair* button allows you to Unpair the device, *Info* will show a dialog with information, *Connect* will allow you to establish a connection

Detailed Info

Clicking on Info brings up a Dialog with information on the Bluetooth device



Clicking on the Properties buttons brings up additional information



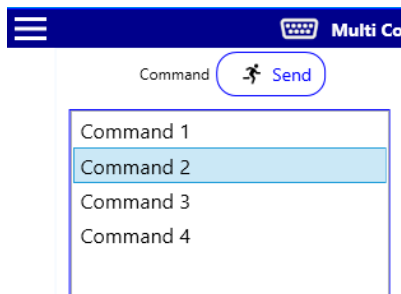
This information is highly technical. Lookup the Microsoft documents for more information.

Unpairing

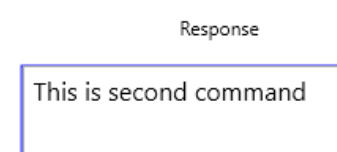
If the device shows up on the paired list, you can click on the *Unpair* button to unpair it.

Connection

Click on Connect to establish a connection. When the device is connected you can select a command and click on Send.



In my example Arduino code, I simply bounce back the incoming message. So, the response shows up on the Response list



In the real world, the Arduino would interpret the command, execute the appropriate action and maybe send back one or more response messages

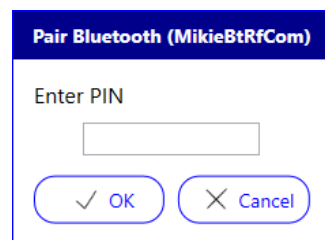
Pairing

If you want to see a list of unpaired devices, uncheck the *Paired Devices* check box and click *Discover*. It will take quite some time. Eventually you will be presented with a list of all non-paired Classic Bluetooth devices within range.



The image shows a Bluetooth interface. On the left, there is a checkbox labeled 'Paired Devices' which is unchecked. Below it are three buttons: 'Discover' with a magnifying glass icon, 'Pair' with a link icon, and 'Info' with an eye icon. On the right, there is a list box containing the name 'MikieBtRfCom'.

In this case I had previously unpaired my Arduino device so it is the only one on the list. You can now click on Pair to Pair it. Or info for additional information. Clicking on Pair brings up a dialog if a PIN is requested by the device. In my example, the default PIN for the Arduino HC-05 module in the product's documentation is 1234. If you enter the wrong PIN you get a failure message



The image shows a dialog box titled 'Pair Bluetooth (MikieBtRfCom)'. Inside, there is a label 'Enter PIN' above a text input field. At the bottom, there are two buttons: 'OK' with a checkmark icon and 'Cancel' with an 'X' icon.

Enter the PIN and click on OK. It is stored by the computer's OS. You then must check the *Paired Devices* check box and click *Discover* to see the newly added device

WIFI Connections

Discovery

To start, select WIFI from the drop-down list.



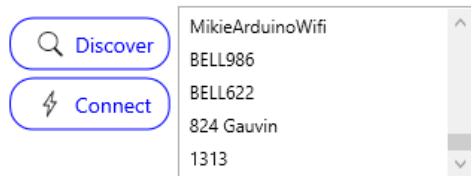
The image shows a dropdown menu with a Wi-Fi icon and the text 'Wifi'.

To the right you will get the Discover button and an empty list



The image shows a 'Discover' button with a magnifying glass icon next to an empty rectangular list box.

Click on *Discover*



You will get a list of WIFI access points. My Arduino UNO Wifi has been setup as an access point (see sample code in appendix)

Connection

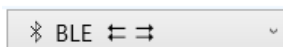
On first connection you will get a dialog to enter credentials and connection information

Enter the information the embedded device expects and click *OK* to connect. If the *Save* check box is checked (default) it will be stored for future connections if the current connection is successful. You will not need to enter it again. You can always edit it via the Menu entry

You can now send and receive messages. See the example in Bluetooth Classic above

Bluetooth Low Energy (BLE) Connections

To start, select BLE from the drop down list

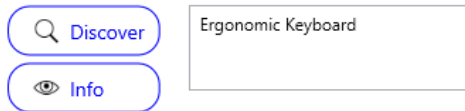


You will get the Discover button and an empty list



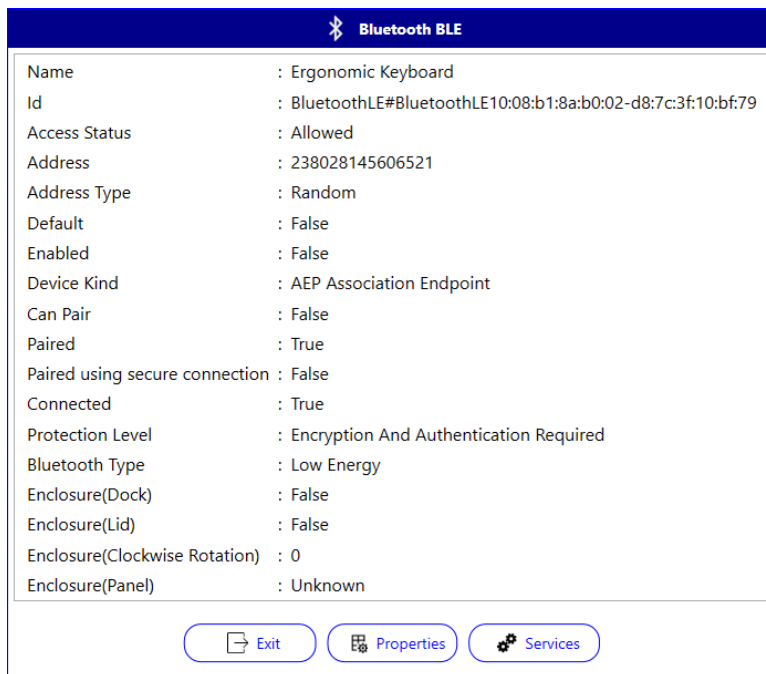
Discovery

Click on the *Discover* button to get a list of BLE devices in range




Detailed info Gather


Click on the *Info* button to get information on the device




Click on *Properties* to get additional information


Properties


System.ItemNameDisplay	: "Ergonomic Keyboard"	: String
{A35996AB-11CF-4935-8B61-A6761081ECDF} 19	: ""	: String
System.Devices.Aep.CanPair	: False	: Bool
System.Devices.Aep.IsPaired	: True	: Bool
{A35996AB-11CF-4935-8B61-A6761081ECDF} 18	: True	: Bool
{A35996AB-11CF-4935-8B61-A6761081ECDF} 13	: True	: Bool
System.Devices.Aep.ContainerId	: b324aa5b-aea2-5868-a9ef-4a51ad19fb8a	: Guid
System.Devices.Icon	: "C:\Windows\System32\DDORes.dll,-2210"	: String
System.Devices.GlyphIcon	: "C:\Windows\System32\DDORes.dll,-3073"	: String


Exit

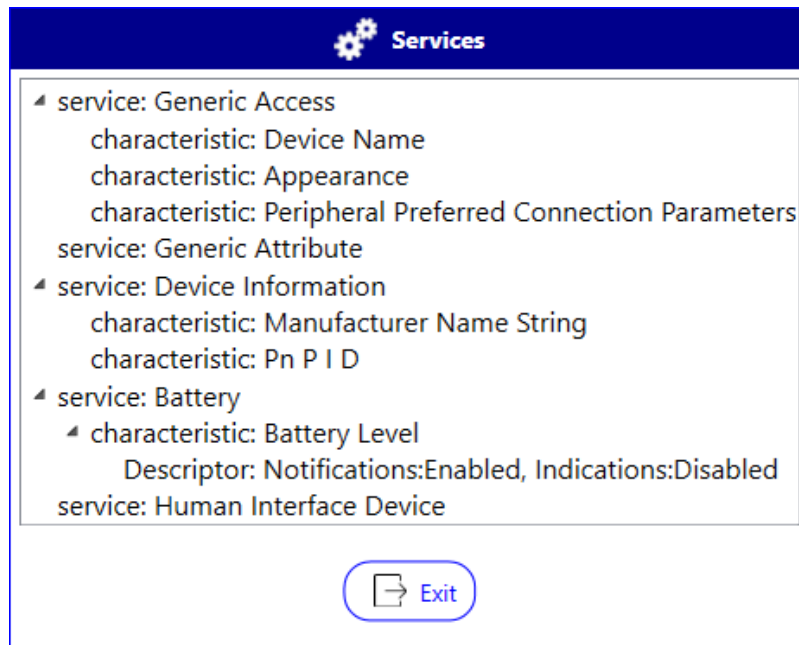
Click on *Services* to get a list of BLE services


Services

- service: Generic Access
 - service: Generic Attribute
- service: Device Information
- service: Battery
 - service: Human Interface Device


Exit

You can expand the tree to drill down from Service to Characteristics to Descriptors



Connection

Connection had been working but a change after a Microsoft update changed BLE to require pairing on connection. Since the Arduino does not yet support it I have removed this functionality until I can find a device to test against

Appendix

Introduction to Embedded Messaging

Your embedded device such as Arduinos may run code that can run without any outside input. However in some cases you may want to send commands remotely to, say, turn on an IO that trips a solenoid that starts to open a garage door.

On simple systems, messages are sent as a string made up of ASCII printable characters (0x41 to 0x7E). That would include punctuations and upper and lower characters. Non printable characters are reserved to specify message termination indicators (terminators)

In this application I have a list from 0x00 (NULL) to 0x1F (unit separator (down arrow)) that you can select as your terminators

So, a command sent to the embedded device to open the garage door might look like this:

OpenDoor\n\r.

The “*OpenDoor*” is the message, the “\n\r” is the terminator sequence. The \n (0x0A) is a new line character and the \r (0x0D) is a carriage return. The backslash is how those are typically inserted in C or C# string.

The App allows you to create a terminator sequence which it automatically adds to the message. You select that sequence based on how you programmed your device. On the flip side, the App expects that same terminator sequence on all message sent back to it.

The messages and terminator sequences you create are stored separately so you can mix and match at run time.

Arduino Bluetooth Sample

```
//-----  
// Name:           ArduinoBluetoothDataTests.ino  
// Created:    12/12/2019 3:14:05 PM  
// Author:      Michael  
//  
// Sample to send and receive data to and from Arduino Bluetooth shield  
//  
// Written and tested in Visual Studio using Visual Micro  
// Tested against Arduino Uno and itea Bluetooth shield with HC-05 module  
//  
// MUST HAVE BOARD TOGGLED TO DATA  
// MUST HAVE BOARD JUMPERS SET FOLLOWING for IO4 TX and IO5 RX
```

```

// °°°|°°
// °°|°°°
// °°°°°°

// MUST HAVE DEBUG SERIAL SET TO 9600 Baud
// MUST HAVE BT SERIAL SET TO 38400 Baud
//
// Must pair with Bluetooth first. The Multi Comm Terminal provides that
// functionality
//-----
#include <SoftwareSerial.h>

#ifndef SECTION_DATA
int i = 0;

// The jumpers on BT board are set to 4TX and 5RX.
// They are reversed on serial since RX from BT gets TX to serial
SoftwareSerial btSerial(5, 4); //RX,TX
bool hasInput = false;
#define IN_BUFF_SIZE 100
char buff[IN_BUFF_SIZE];
unsigned char inIndex = 0;
#endif // !SECTION_DATA

// the setup function runs once when you press reset or power the board
void setup() {
    // There is some strange behaviour when using different baud rates
    SetupCommunications(9600, 38400);
}

// the loop function runs over and over again until power down or reset
void loop() {
    ListenToBTData();
}

// Private helpers

```

```

void SetupCommunications(long dbgBaud, long btBaud) {
    btSerial.begin(btBaud);
    Serial.begin(dbgBaud);

    while (!Serial) {
        ; // wait for serial port to connect. Needed for Native USB only
    }

    Serial.println("Debug serial active");
    // example had pin 9 set low, then high but does not seem necessary
}

```

```

void ListenToBTData() {
    if (btSerial.available() > 0) {
        if (inIndex >= IN_BUFF_SIZE) {
            inIndex = 0;
            Serial.println("Corrupt BT input. Purge buffer");
        }

        buff[inIndex] = (char)btSerial.read();
        if (buff[inIndex] == '\r') {
            Serial.write("CR");
        }
        else if (buff[inIndex] == '\n') {
            Serial.write("LN");
        }
        else {
            Serial.write(buff[inIndex]);
        }

        // Doing \n\r
        if (buff[inIndex] == '\r') {
            Serial.println("Printing msg in buffer");
            hasInput = true;
            Serial.write(buff, inIndex + 1);
            btSerial.write(buff, inIndex + 1);
        }
    }
}

```

```

        memset(buff, 0, IN_BUFF_SIZE);
        inIndex = 0;
    }
    else {
        inIndex++;
        // Wipe out buffer if too long
        if (inIndex >= IN_BUFF_SIZE) {
            inIndex = 0;
            Serial.println("Corrupt BT input. Purge buffer");
        }
        else {
        }
    }
}

else {
    if (hasInput) {
        hasInput = false;
    }

    if (i % 50 == 0) {
        Serial.print("No BT msg # ");
        Serial.print((i / 10));
        Serial.println("");
    }
    i++;
    delay(100);
}
}

```

Arduino BLE Sample

```
// -----
// Name:      ArduinoBLESerial.ino
// Created:    4/28/2020 8:55:58 PM
// Author:     Michael
//
// Sets up the BLE to simulate a serial connection by having one characteristic
// configured as an incoming channel, and another one as the outgoing
//
// Tested on the Arduino UNO WIFI Rev2
//
// Was working but a subsequent update of Windows 10 put in BLE pairing on
// connect which is not yet supported in the Arduino library so it will fail
// on connection
//
// -----
#include <ArduinoBLE.h>
#include <string.h>

#ifndef SECTION_DATA

#define MAX_BLOCK_SIZE 20

// Create services that will act as a serial port. Max 20 bytes at a time
BLEService serialService("9999");

// Out channel 0x99,0x98 = 39,320 base 10. Caller reads or gets notifications from
this device
// Need a write in Setup with block of MAX_BLOCK_SIZE size OR it is not recognized by
caller
BLECharacteristic outputCharacteristic("9998", BLERead | BLENotify, MAX_BLOCK_SIZE);

// In channel 0x99,0x97 = 39,319 base 10. Caller writes to this device
BLECharacteristic inputCharacteristic("9997", BLEWrite, MAX_BLOCK_SIZE);

// 0x2901 is CharacteristicUserDescription data type
BLEDescriptor outputDescriptor("2901", "Serial Output");
BLEDescriptor inputDescriptor("2901", "Serial Input");

bool hasInput = false;
#define IN_BUFF_SIZE 500
char buff[IN_BUFF_SIZE];
unsigned char inIndex = 0;

#endif

// the setup function runs once when you press reset or power the board
void setup() {
    // TODO - This needs to be removed to work if not connected to USB serial port
    Serial.begin(9600);
    while (!Serial) {
        ; // wait for serial port to connect. Needed for Native USB only
    }
    Serial.println("Debug serial port active");

    pinMode(LED_BUILTIN, OUTPUT);
    if (!BLE.begin()) {
        Serial.println("Failed to start BLE");
        while (1);
    }
}
```

```

else {
    Serial.println("BLE begun");
}

// Name that users see in list for 'Add Device'
BLE.setLocalName("Test Arduino BLE serial device");
// The Device Name Characteristic
BLE.setDeviceName("Test BLE serial device");

// assign event handlers for connected, disconnected events
BLE.setEventHandler(BLEConnected, bleOnConnectHandler);
BLE.setEventHandler(BLEDisconnected, bleOnDisconnectHandler);

setupSerialDescriptor(serialService, outputCharacteristic, outputDescriptor);

setupSerialDescriptor(serialService, inputCharacteristic, inputDescriptor);
inputCharacteristic.setEventHandler(BLEWritten, inBytesWritten);

// Add service to BLE and start advertising
BLE.addService(serialService);
BLE.setAdvertisedService(serialService);
BLE.advertise();
Serial.println("Bluetooth device active, waiting for connections...");
}

// the loop function runs over and over again until power down or reset
void loop() {
    BLEDevice central = BLE.central();
    if (central) {
        // This will hold up the loop function as long as the connection persists
        while (central.connected()) {
            // The message processing is done via the event handler for incoming bytes
        }
        // Since the current connection has terminated wipe out accumulated in bytes
        ResetInBuffer();
    }
}

// Initialize the serial Characteristic, add Descriptor and connect to serial
void setupSerialDescriptor(BLEService& service, BLECharacteristic& characteristic,
BLEDescriptor& desc) {
    byte initBuffContents[MAX_BLOCK_SIZE];
    // If we initialize with 0x00 (null) creates weird output in future writes
    memset(initBuffContents, 0x20, MAX_BLOCK_SIZE);
    characteristic.writeValue(initBuffContents, MAX_BLOCK_SIZE);
    characteristic.addDescriptor(desc);
    service.addCharacteristic(characteristic);
}

void ProcessIncomingBuff() {
    // Doing \n\r terminator sequence
    if (buff[inIndex] == '\r') {
        Serial.println("Printing msg in buffer");
        hasInput = true;

        Serial.print("Sending back ");
        Serial.print(inIndex);
        Serial.println(" bytes to write");

        // This works by iterating through BLOCK_SIZE byte blocks.

```



```

    int count = inIndex / MAX_BLOCK_SIZE;
    int last = (inIndex % MAX_BLOCK_SIZE);
    int lastIndex = 0;
    for (int i = 0; i < count; i++) {
        lastIndex = i * MAX_BLOCK_SIZE;
        outputCharacteristic.writeValue(&buff[lastIndex], MAX_BLOCK_SIZE);
    }
    if (last > 0) {
        if (lastIndex > 0) {
            lastIndex += MAX_BLOCK_SIZE;
        }
        outputCharacteristic.writeValue(&buff[lastIndex], last);
    }

    ResetInBuffer();
}
else {
    //
    inIndex++;
    // Wipe out buffer if too long
    if (inIndex >= IN_BUFF_SIZE) {
        ResetInBuffer();
        Serial.println("Corrupt BT input. Buffer purged");
    }
}
}

```

```

void ResetInBuffer() {
    // Reset with spaces. Acts weird if you use 0x00 null)
    memset(buff, 0x20, IN_BUFF_SIZE);
    inIndex = 0;
}

```

```

byte inBuff[MAX_BLOCK_SIZE];

```

```

// Event handler for when data is written to the inByte characteristic
void inBytesWritten(BLEDevice device, BLECharacteristic byteCharacteristic) {
    digitalWrite(LED_BUILTIN, HIGH);

    int count = inputCharacteristic.readValue(inBuff, MAX_BLOCK_SIZE);
    ProcessIncomingBuff2(inBuff, count);
    digitalWrite(LED_BUILTIN, LOW);
}

```

```

void ProcessIncomingBuff2(uint8_t* inData, int length) {
    // index is pos of next write
    memcpy(&buff[inIndex], inData, length);

    int newIndex = inIndex + length;
    // This is where I would process the data. Remove complete command, etc
    inIndex = newIndex;

    Serial.write(inData, length); Serial.println("");
    Serial.write(buff, newIndex); Serial.println("");

    // Just bounce back whatever came in
    outputCharacteristic.writeValue(inData, length);
    // Check if there was a \n and simply obliterate main buffer
    for (int i = 0; i < length; i++) {
        if (inData[i] == '\n') {

```

```
        ResetInBuffer();
        break;
    }
}

// Event handler for on BLE connected
void bleOnConnectHandler(BLEDevice central) {
    Serial.print("CONNECTED, central: ");
    Serial.println(central.address());
}

// Event handler for on BLE disconnected
void bleOnDisconnectHandler(BLEDevice central) {
    Serial.print("DISCONNECTED, central: ");
    Serial.println(central.address());
}
```

Arduino WIFI Sample

```
// -----
// Name:          TestArduinoWifi.ino
// Created:   10/16/2020 1:22:40 PM
// Author:    Michael
//
// Tested on the Arduino UNO WIFI Rev2
//
// Sets up the board as a WIFI access point with a socket
//
// Initial example code found in:
// https://www.arduino.cc/en/Reference/WiFiNINABeginAP
//
// -----
#include <SPI.h>
#include <WiFiNINA.h>
#include "wifi_defines.h"
// -----
// The wifi_defines.h has the strings for the SSID and password
// Here are the contents
//      #pragma once
//
//      Must be 8 or more characters
//      #define MY_SSID "MikieArduinoWifi"
//
//      Must be 8 or more characters
//      #define MY_PASS "1234567890"
//
// -----

char ssid[] = MY_SSID;
char pwd[] = MY_PASS;
int keyIndex = 0;
int status = WL_IDLE_STATUS;
int led = LED_BUILTIN;

// Port 80 of socket usually used for HTTP - just using it as a entry
// The Multi Comm Terminal must enter whatever port number is set in
// the Arduino
WiFiServer server(80);

void setup() {
    // Serial is just to push data for debugging the Arduino code. Can be removed
    Serial.begin(57600);
    while(!Serial){}
    Serial.println("Serial started");

    pinMode(led, OUTPUT);

    // Check for the WiFi module:
    if (WiFi.status() == WL_NO_MODULE) {
        Serial.println("Communication with WiFi module failed!");
        // don't continue
        while (true);
    }

    String fv = WiFi.firmwareVersion();
    Serial.print("WIFI firmware version ");
    Serial.println(fv);
}
```

```

if (fv < WIFI_FIRMWARE_LATEST_VERSION) {
    Serial.print("Version below ");
    Serial.println(WIFI_FIRMWARE_LATEST_VERSION);
    Serial.println("Please upgrade the firmware");
}

// by default the local IP address of will be 192.168.4.1
// you can override it with the following:
// Whatever you choose will be the IP that you enter in
// the Multi Comm Connection parameters
// WiFi.config(IPAddress(10, 0, 0, 1));

// Print the network name (SSID);
Serial.print("Creating access point named: ");
Serial.println(ssid);

// Create access point
status = WiFi.beginAP(ssid, pwd);
if (status != WL_AP_LISTENING) {
    Serial.print("Status "); Serial.println(status);
    Serial.println("Access point creation failed");
    while (true) { }
}

delay(10000);

server.begin();
// you're connected now, so print out the status to the serial debug:
printWifiStatus();
}

// the loop function runs over and over again until power down or reset
void loop() {
    // Print a message to debug if a device has connected or disconnected
    if (status != WiFi.status()) {
        status = WiFi.status();
        if (status == WL_AP_CONNECTED) {
            Serial.println("Device connected to AP");
        }
        else {
            // Device has disconnected from AP. Back in listening mode
            Serial.println("Device disconnected from AP");
        }
    }
    ListenForClient();
}

// Determines if a client has connected a socket and sent a message
void ListenForClient() {
    //https://www.arduino.cc/en/Reference/WiFiNINABeginAP
    WiFiClient client = server.available();
    if (client) {
        Serial.println("Got a client connected new client");
        String currentLine = "";

        // Loop while the client's connected
        while (client.connected()) {
            if (client.available()) {
                // Read a byte
                char c = client.read();
                // Print character serial for debug
            }
        }
    }
}

```

```

        Serial.write(c);

        // This will bounce each character through the socket
        // The MultiCommMonitor will pick up the terminators and
        // Display it as a return message
        //
        // In the real world, you would accumulate the bytes until
        // the expected terminator sequence is detected.
        // You would then
        // - Look at the message
        // - Determine operation requested
        // - Do the operation
        // - Optionally, send back a response with expected
terminators
        //
        // See samples for BT Classic and BLE
        client.print(c);
    }

    // close the connection:
    client.stop();
    // Send a debug message
    Serial.println("client disconnected");
}
}

```

```

void printWifiStatus() {
    // print the SSID of the network you're attached to:
    Serial.print("SSID: ");
    Serial.println(WiFi.SSID());

    // print your board's socket IP address:
    IPAddress ip = WiFi.localIP();
    Serial.print("IP Address: ");
    Serial.println(ip);

    // print the received signal strength:
    long rssi = WiFi.RSSI();
    Serial.print("signal strength (RSSI):");
    Serial.print(rssi);
    Serial.println(" dBm");
}

```