

# Multi Comm Terminal

Hobbyist Terminal Program to communicate with Embedded systems

By Michael Roop

## Table of Contents

Introduction .....	4
Supported Mediums .....	4
Data Storage.....	4
Lists of Commands .....	4
Terminators.....	4
WIFI Credentials.....	5
Languages .....	5
Settings.....	5
User Interface .....	5
Buttons.....	5
Wordless Buttons.....	6
Main Window.....	6
Drop Down Menu.....	7
Language .....	7
Terminators.....	8
Commands .....	11
Code Samples.....	12
Credentials .....	14
USB Settings .....	15
About.....	17
Bluetooth Classic Connections.....	17
Discovery.....	17
Detailed Info.....	18
Unparing.....	19
Connection .....	19
Pairing .....	20
WIFI Connections .....	20
Discovery.....	20
Connection.....	23
Connection.....	<b>Error! Bookmark not defined.</b>
Appendix .....	24
Introduction to Embedded Messaging .....	24

Arduino Bluetooth Sample..... 24

Arduino BLE Sample..... **Error! Bookmark not defined.**

Arduino WIFI Sample ..... 28

Arduino USB Sample ..... **Error! Bookmark not defined.**

## 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.
  - Future release
- USB:
  - Connection via hard wired USB port
  - Currently evaluating whether it would be useful

## Data Storage

Data storage is hidden from the user. The Android device would need to be routed to be seen

## 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 established. 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 8 languages in most areas of the UI:

- English
- French
- German
- Italian
- 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 device saves, 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 could be added over time

## User Interface

### Buttons



Some buttons have both an icon and text. So, if you inadvertently switch to a language you do not understand you can still navigate back to change it.

## Wordless Buttons

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



Add: Add a new Item



Edit: Edit and existing item



Delete: Remove an item

## Main Window

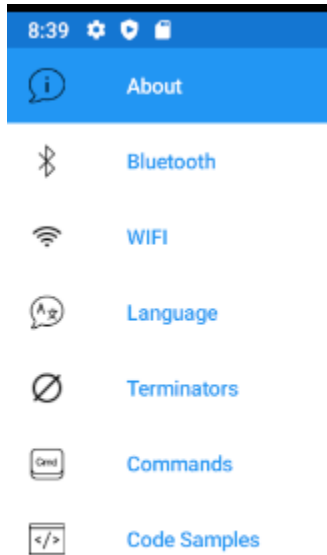
The App opens with the About page which will be described later



## Drop Down Menu



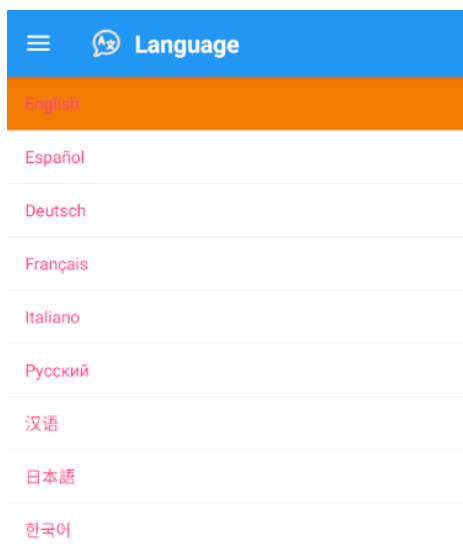
Touching on the hamburger symbol brings out the slide out menu



## Language



Clicking on this will bring up the Language selector. Currently there are 8 languages supported for UI elements



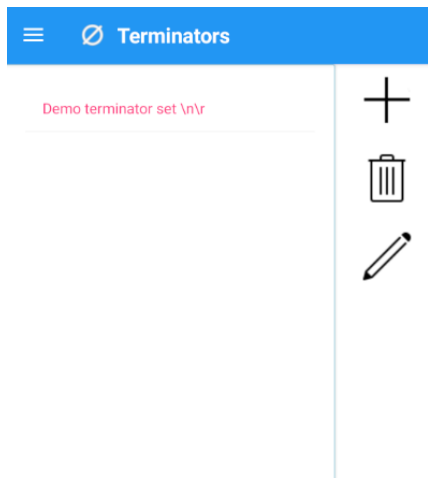
Highlight the requested language all buttons and other text will change. It will be saved as your preference for future sessions.

## Terminators

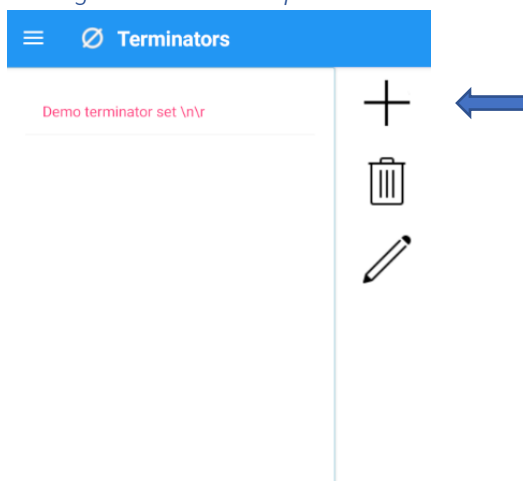


Clicking on this brings up a dialog box with currently stored terminator sequence.

- 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



### *Adding a terminator sequence set*



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



←

Ø

Edit

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

NewSet

←

Enter the name of the set so you can identify it in the future.

To add a terminator, click on any one of the terminators in the list above. It will be added to the set below the set name.

My Terminator Set \n\r

0x0a    0x0d  
 \n    \r

To remove the last terminator in the set touch the delete symbol. When you are satisfied, click on the OK symbol. The set will be save and you will be returned to the Terminators page. Your new set will be posted in the list.

≡

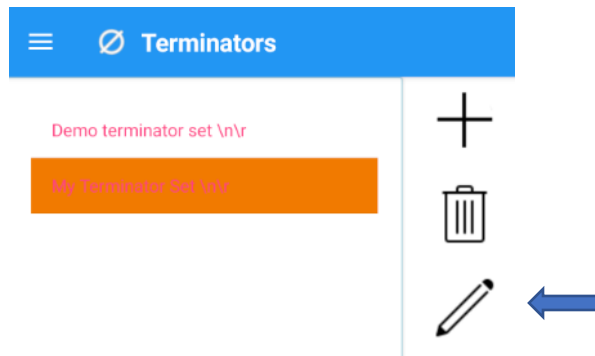
Ø

Terminators

Demo terminator set \n\r

My Terminator Set \n\r

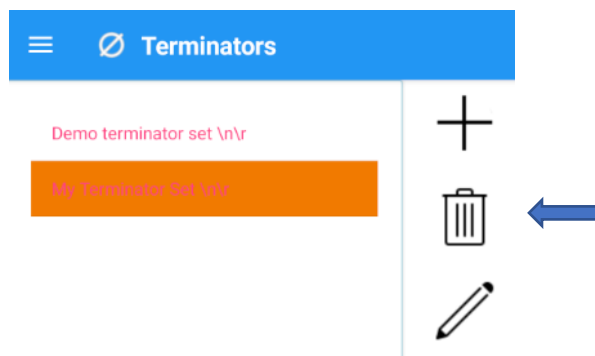
### *Edit Terminator set*



Back at the Terminator set list dialog, click on the Edit button for the editor dialog. You will be brought back to the edit screen where you can add or delete terminators and edit the name.

### *Deleting a Terminator set*

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

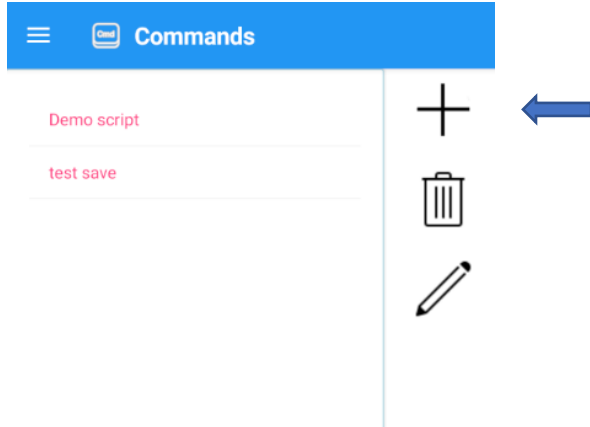


## Commands



### Commands

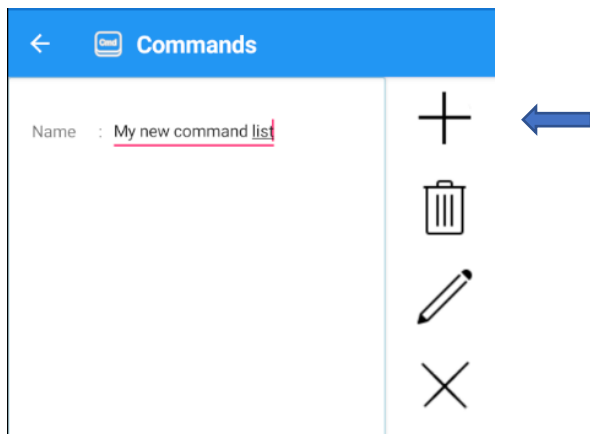
Touch the Commands to bring up the Command set builder page.



You can just click on a Command set to select it as the current.

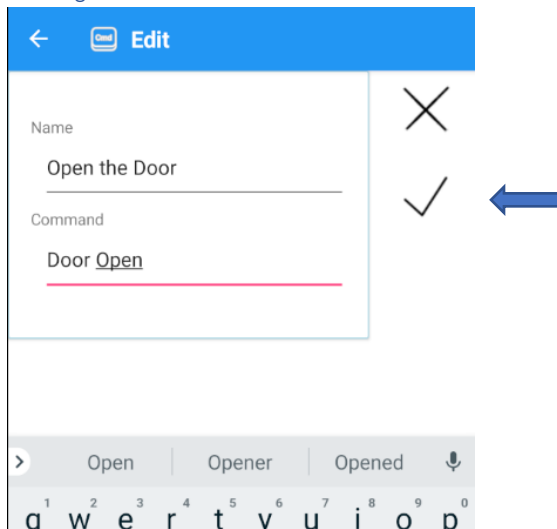
### *Adding a command list*

The command set builder allows you to edit the name. To add a new Command set touch the add symbol



You will be presented with the command builder page.

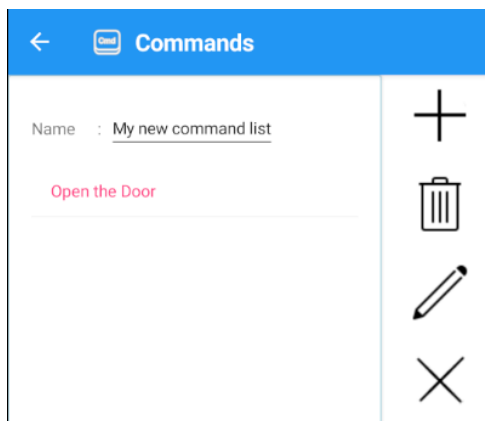
### Adding a command



The screenshot shows the 'Edit' screen for a command. The top bar is blue with a back arrow and the word 'Edit'. Below, there are two input fields: 'Name' with the text 'Open the Door' and 'Command' with the text 'Door Open'. To the right of these fields are two icons: a large 'X' and a checkmark. A blue arrow points to the checkmark icon. At the bottom, there is a keyboard with a microphone icon and a list of letters: a, w, e, r, t, v, u, i, o, p.

On the command builder page, edit the command name and the actual command that will be sent to the device. Touch the OK symbol to save

On returning to the command set builder page you will see you new command in the list



The screenshot shows the 'Commands' screen. The top bar is blue with a back arrow and the word 'Commands'. Below, there is a list of commands. The first command is 'My new command list' with the text 'Open the Door' below it. To the right of the list are four icons: a plus sign, a trash can, a pencil, and a large 'X'.

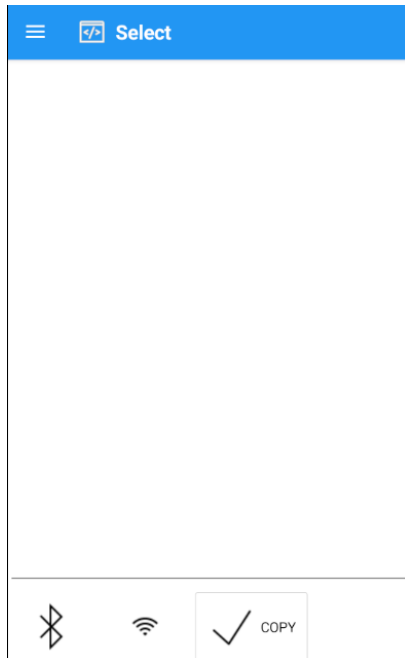
You can click on the delete symbol to remove it from the list

### Code Samples

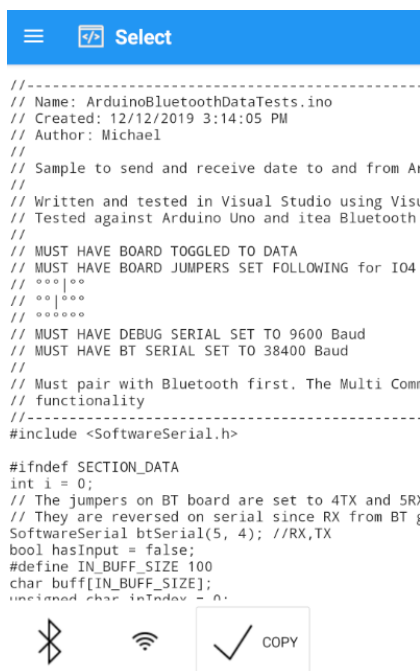


[Code Samples](#)

This will allow you to view Arduino code samples that receive commands and return responses. Touch the menu item to bring up the samples page

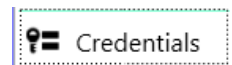


At the bottom you will see Bluetooth and WIFI symbols. Touch one of these to bring up the sample.

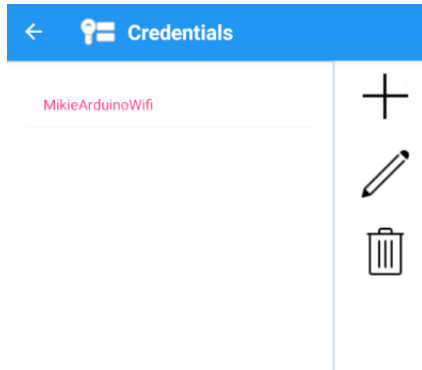


Once you have the code listed, you can touch the Copy button to save it to the clipboard. You can then paste it into an email to send it to yourself to load it onto an Arduino board.

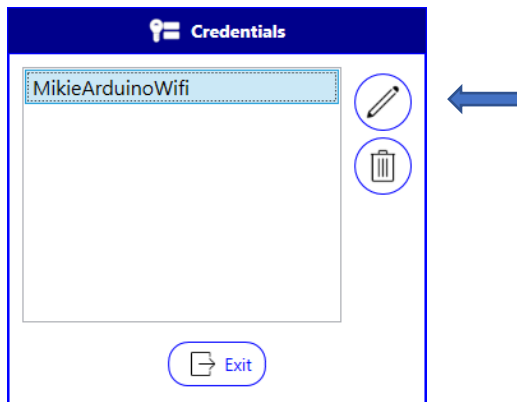
## Credentials



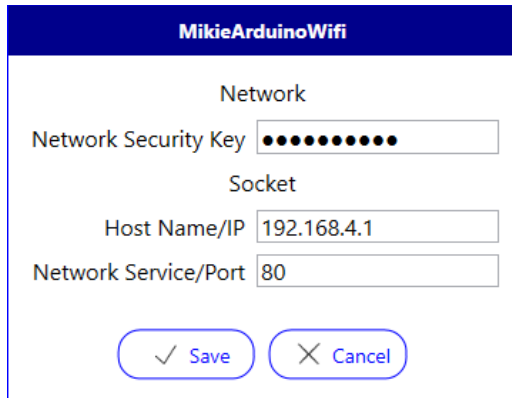
The Credentials dialog provides a way to edit or delete credentials and connection parameters for a previous WIFI connection. Clicking on this will open a dialog with a list of stored credentials



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



**MikieArduinoWifi**

Network

Network Security Key

Socket

Host Name/IP

Network Service/Port

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.

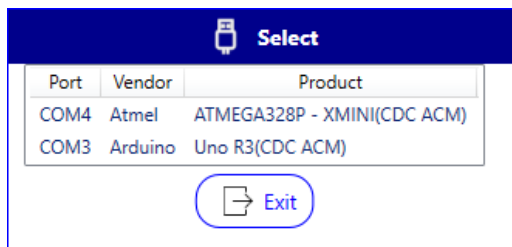
Click on Save to store your changes, or cancel to discard

## USB Settings



### USB Settings

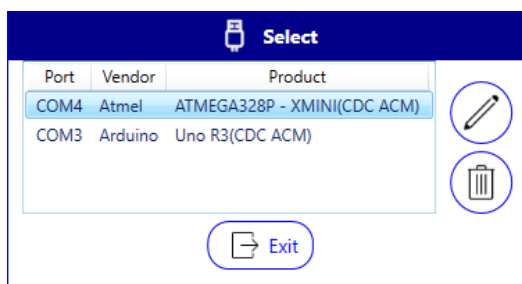
Open the USB Select dialog



**Select**

Port	Vendor	Product
COM4	Atmel	ATMEGA328P - XMINI(CDC ACM)
COM3	Arduino	Uno R3(CDC ACM)

Each line is a stored configuration for a USB serial connection

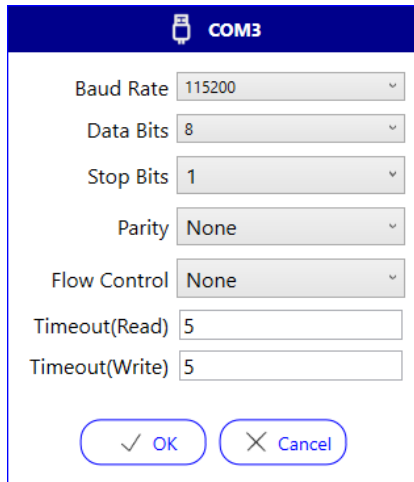


**Select**

Port	Vendor	Product
COM4	Atmel	ATMEGA328P - XMINI(CDC ACM)
COM3	Arduino	Uno R3(CDC ACM)

When you select one of the configurations, the edit and delete buttons appear. You cannot create a new configuration here. It must be done on first connection since it grabs information from the device.

If you click on the edit button you are presented with a dialog

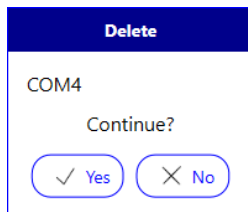


A screenshot of a 'Serial Port Configuration' dialog box for COM3. The dialog has a dark blue header with a USB icon and the text 'COM3'. Below the header, there are several settings: 'Baud Rate' set to 115200, 'Data Bits' set to 8, 'Stop Bits' set to 1, 'Parity' set to None, and 'Flow Control' set to None. These are all dropdown menus. Below these are two text input fields: 'Timeout(Read)' and 'Timeout(Write)', both containing the value 5. At the bottom, there are two buttons: 'OK' with a checkmark icon and 'Cancel' with an 'X' icon.

Setting	Value
Baud Rate	115200
Data Bits	8
Stop Bits	1
Parity	None
Flow Control	None
Timeout(Read)	5
Timeout(Write)	5

This will have the configuration information based on what was harvested from the serial connection and any changes you made on first connection. Since there is no feedback if you have wrong settings, you may need to come back here to correct them. You must enter the settings that the embedded device is expecting. Look at the Arduino code sample for serial.

Back at the USB Select dialog, if you click on the delete button you are presented with a confirmation dialog.



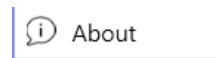
A screenshot of a 'Delete' confirmation dialog box. The dialog has a dark blue header with the text 'Delete'. Below the header, it says 'COM4' and 'Continue?'. At the bottom, there are two buttons: 'Yes' with a checkmark icon and 'No' with an 'X' icon.

Text
COM4
Continue?

Click on *Yes* to delete it, *No* to exit without changes.



## About



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



- Author: Link directs you to my Linked in Profile in case you want to send feedback.
- Icons: Link is to the site that provided the excellent icons
- Build: Information refers to my current source control system. It would be helpful to send this information if you are reporting an error
- User Manual: Link will open the user manual PDF file in the browser

## Bluetooth Classic Connections

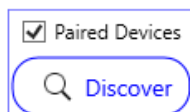
Touch the menu item to get started



The Bluetooth medium connects via an RFComm connection to the embedded device. This is provided on Arduinos with modules such as HC-05.

## Discovery

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



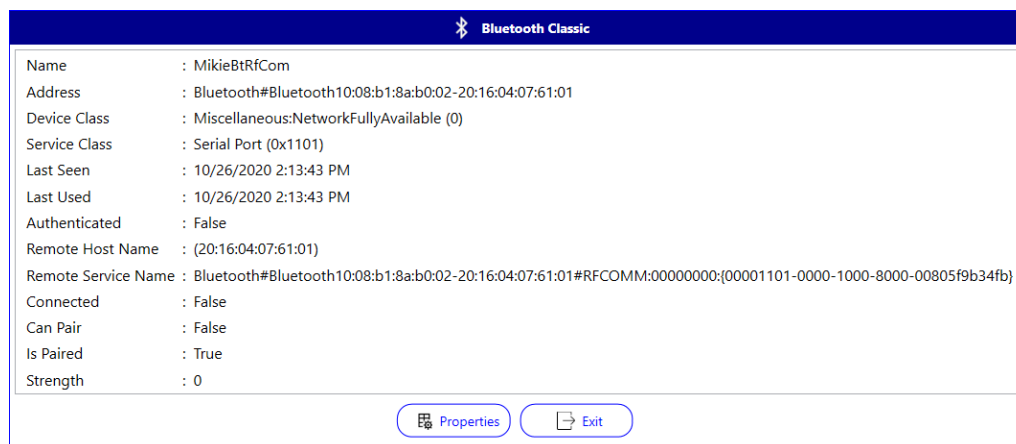
If the Paired Devices is checked, you will get a list of previously paired devices. This is very fast but the device itself may not be available. The list is populated from information stored the computer's OS. The paired devices will show up in the list box to the right



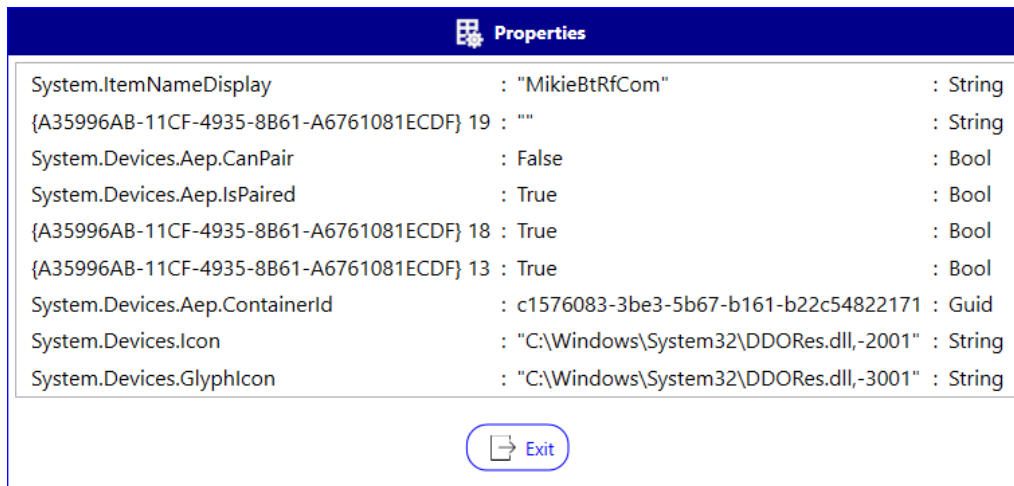
Extra buttons will appear. The *Unpair* button allows you to Unpair the device, *Info* will show a dialog with information, *Connect* will allow you to establish a connection

### Detailed Info

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



Clicking on the Properties buttons brings up additional information



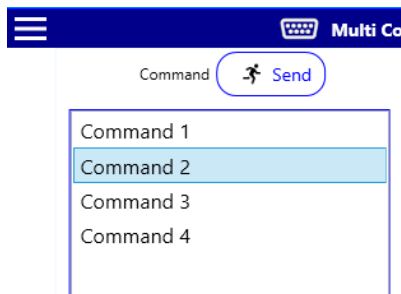
This information is highly technical. Lookup the Microsoft documents for more information.

## Unpairing

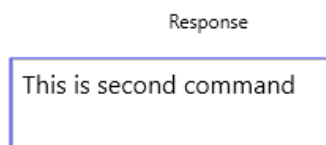
If the device shows up on the paired list, you can click on the *Unpair* button to unpair it.

## 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



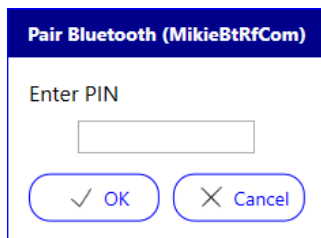
In the real world, the Arduino would interpret the command, execute the appropriate action and maybe send back one or more response messages

## 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.



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



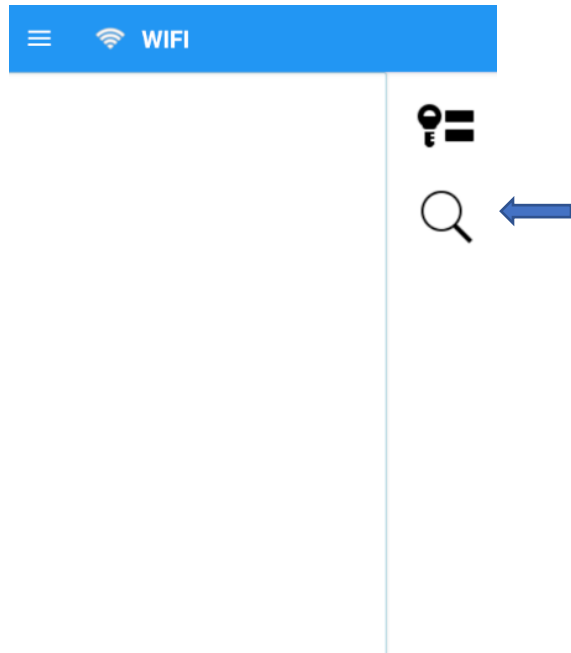
Enter the PIN and click on OK. It is stored by the computer's OS. You then must check the *Paired Devices* check box and click *Discover* to see the newly added device

## WIFI Connections

To start touch the WIFI menu item

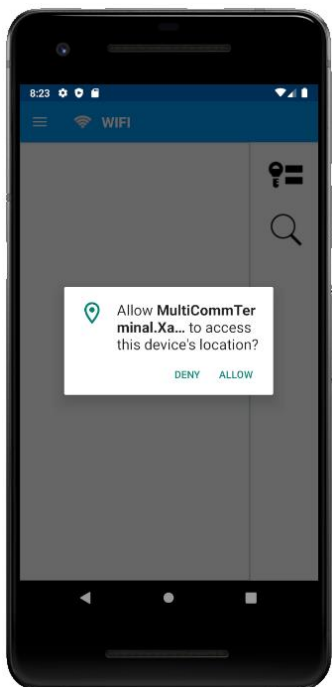


## Discovery

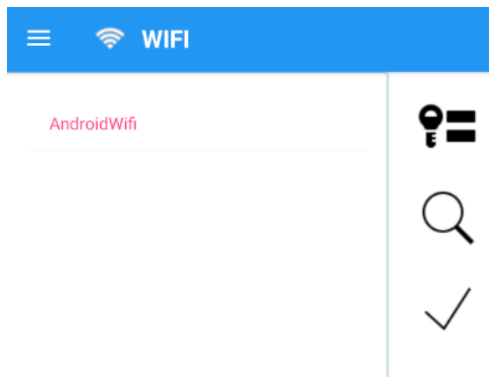


On entry you will have an empty list of paired devices. To fill the list, touch the discovery symbol.

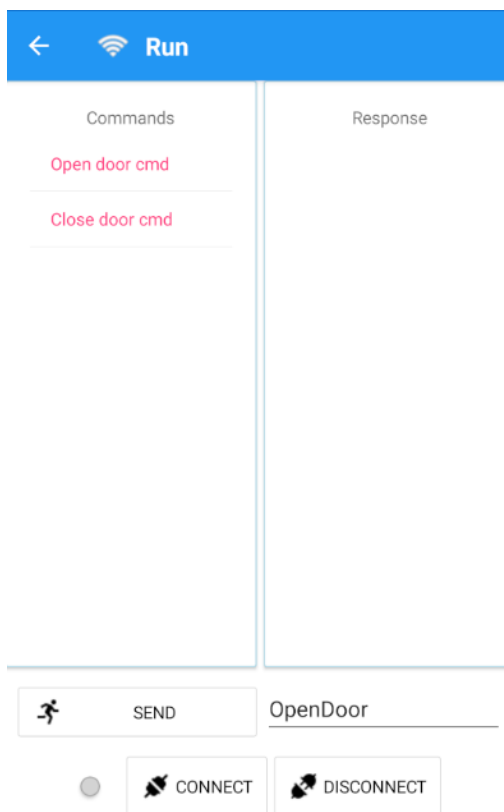
On the first connection, Android will request permission to use your location. There is a required permission to do a network connection



You will get a list of WIFI access points.



Select the desired wireless SSID and touch on the OK symbol. The Run page will open



To the left you will see the command set that you had previously selected. The right pane will show responses from the connected device

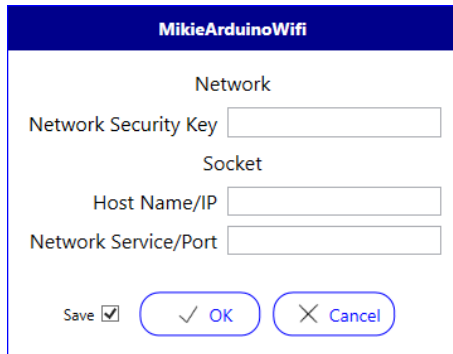
To connect, touch the connect button. The busy symbol comes up until connection is complete. At that point, the connection light goes green.

Touch any command in the list and it will appear in the send entry box. Click send to push the command towards the device. Or just enter the command directly and touch send.

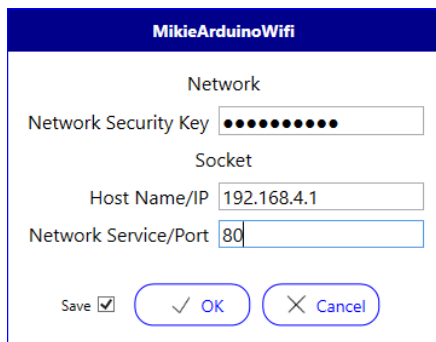
To disconnect, touch the disconnect button

## Connection

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



The dialog box has a dark blue header with the text "MikieArduinoWifi". Below the header, the word "Network" is centered. There are three input fields: "Network Security Key", "Host Name/IP", and "Network Service/Port". The "Host Name/IP" field is under the heading "Socket". At the bottom left is a "Save" checkbox which is checked. To the right of the checkbox are two buttons: "OK" with a checkmark icon and "Cancel" with an X icon.



This dialog box is identical in layout to the first one, but with values entered in the fields. The "Network Security Key" field contains ten black dots. The "Host Name/IP" field contains the text "192.168.4.1". The "Network Service/Port" field contains the text "80". The "Save" checkbox is checked, and the "OK" and "Cancel" buttons are at the bottom right.

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

## 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 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");
}

```