Multi Comm Terminal

Hobbyist Terminal Program to communicate with Embedded systems

By Michael Roop

Table of Contents

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 ITEA Bluetooth shield with 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 see it.

Lists of Commands

The user can create, store, and retrieve lists of ASCII text based messages. The messages can be sent to the embedded device once connection is established. The embedded device will 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 asynchronous 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.

Languages

The App currently supports 9 languages in most areas of the UI. More will be added over time:

- English
- French
- German
- Italian
- Spanish
- Chinese (Simplified)
- Japanese
- Korean
- Russian

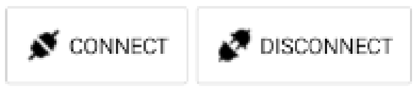
I followed an approach of using single word descriptions for headers and buttons. I used the Microsoft Technical Language Portal to get the translation. See: https://www.microsoft.com/en-us/language/Search?&searchTerm=Automatic&langID=303&Source=true&productid=0. This allows for a high precision in rendering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

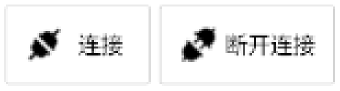
Settings

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, Terminator set, 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.

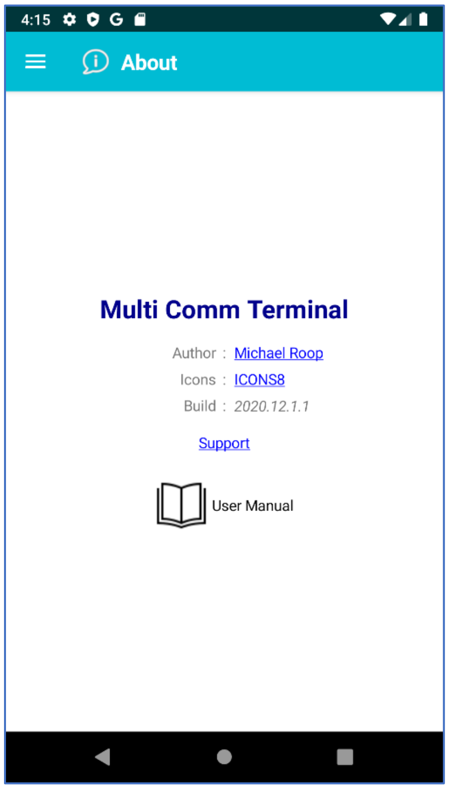
Wordless Buttons

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
✓	Select/Ok
🔍	Discover: Do a search for something

Main Page

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



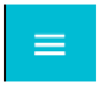
Multi Comm Terminal

Author : Michael Roop
Icons : ICONS8
Build : *2020.12.1.1*

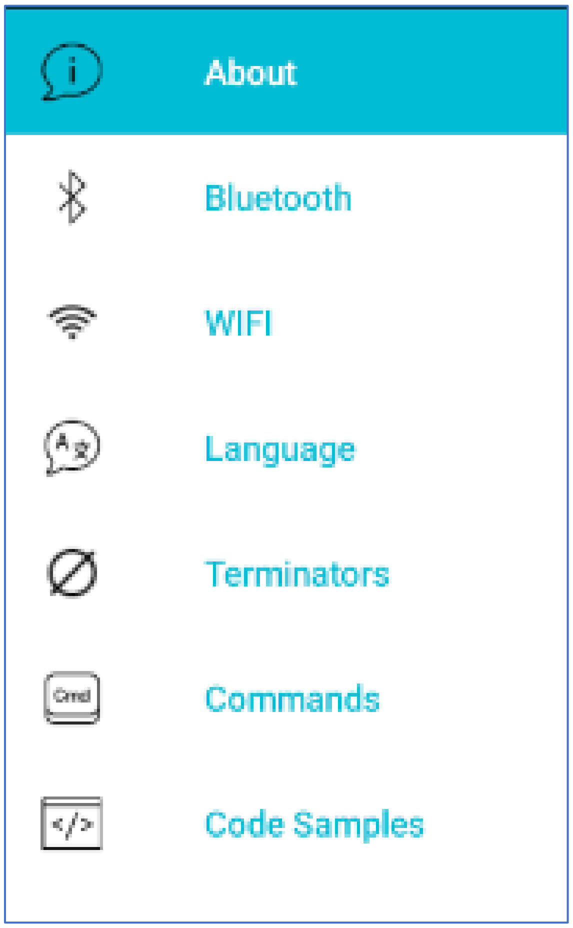
Support

User Manual

Drop Down Menu



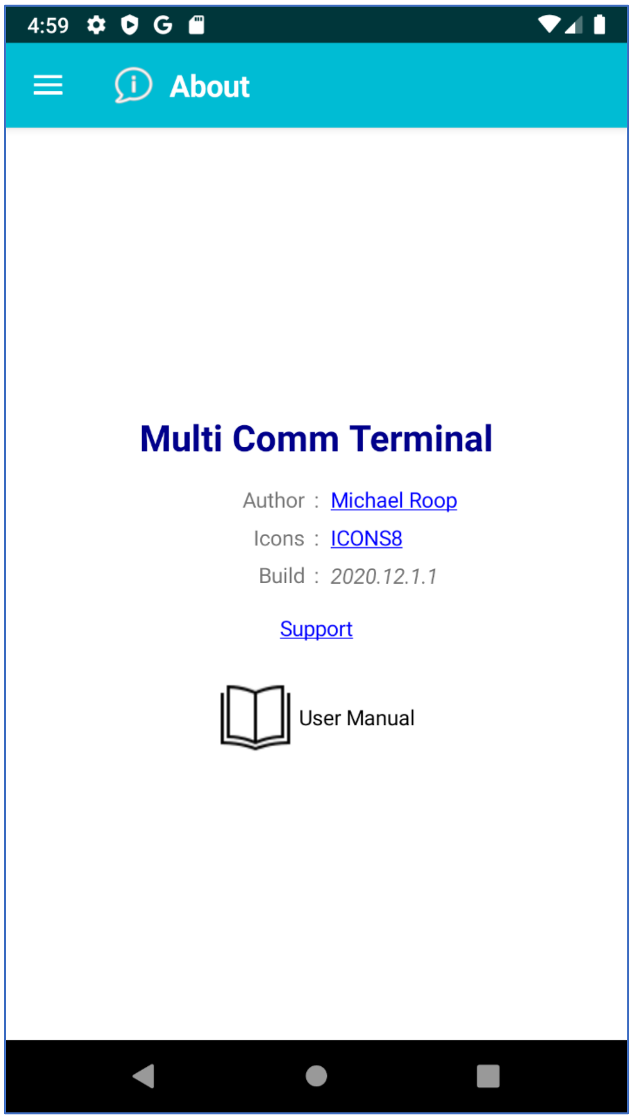
Tapping on the hamburger icon brings out the slide out menu



About

About

The App starts on the About page. You can return to it any time by tapping on the *About* menu item



Multi Comm Terminal

Author : Michael Roop

Icons : ICONS8

Build : 2020.12.1.1

Support

User Manual

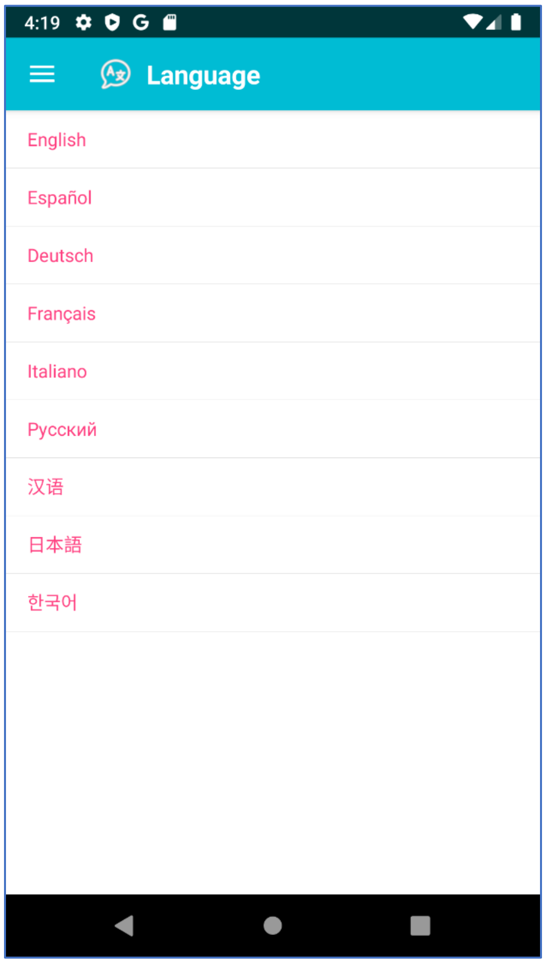
On this page you will find:

- 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
- Support: Link to open email with address for support
- User Manual: Link will open this user manual PDF file

Language Page

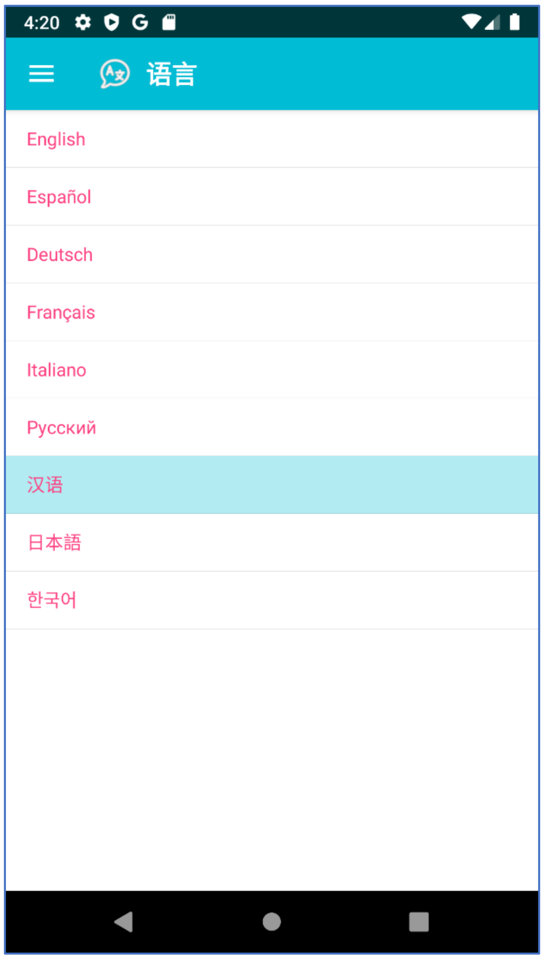
Language

Tapping on this will bring up the Language selector. There are currently 9 languages


4:19

Language

English
Español
Deutsch
Français
Italiano
Русский
汉语
日本語
한국어


4:20

语言

English
Español
Deutsch
Français
Italiano
Русский
汉语
日本語
한국어

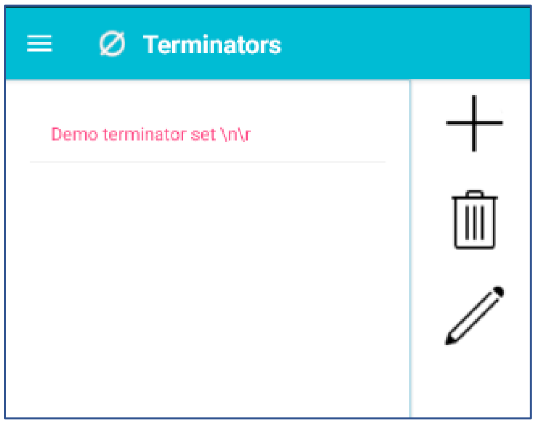
Highlight the requested language all buttons and other text will change. It will be saved as your preference for future sessions.

Terminators Page

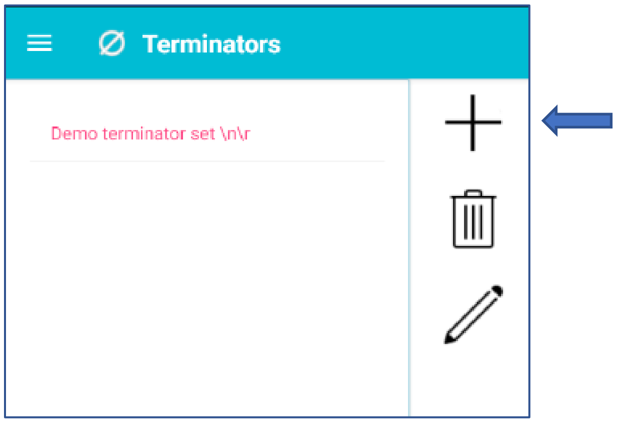
Ø Terminators

Tapping on this brings up a page with the currently stored terminator sequences.

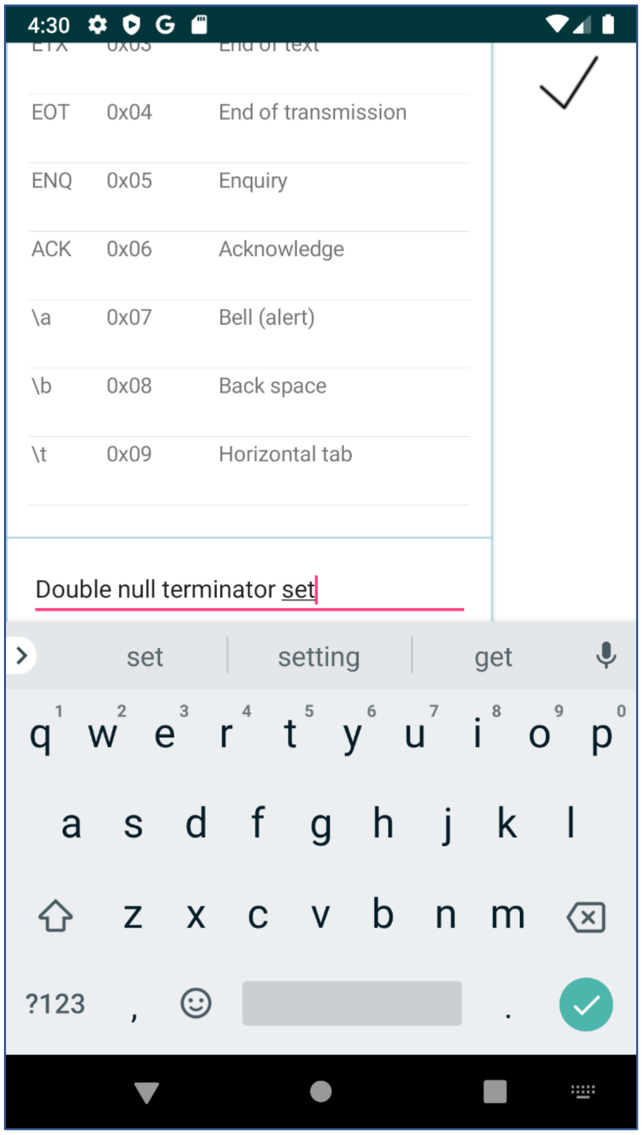
- A terminator sequence contains one or more nonprintable characters that are added to the end of the outgoing message.
- The embedded device must be expecting that sequence to determine when an incoming message is complete.
- The embedded device must also add those terminators to its outgoing messages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



Adding a terminator sequence set



In the Terminators Set Page, tap on the Add button and a Page appears to create a new set

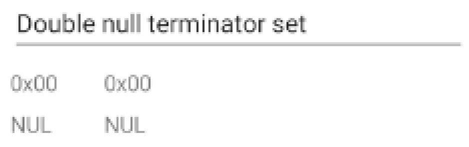


ETX	0x03	End of text
EOT	0x04	End of transmission
ENQ	0x05	Enquiry
ACK	0x06	Acknowledge
\a	0x07	Bell (alert)
\b	0x08	Back space
\t	0x09	Horizontal tab

Double null terminator set

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

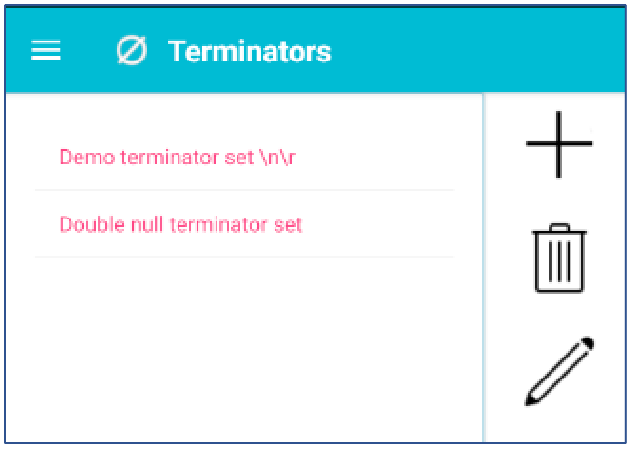
To add a terminator, tap on any one of the terminators in the list above. It will be added to the set below the set name.



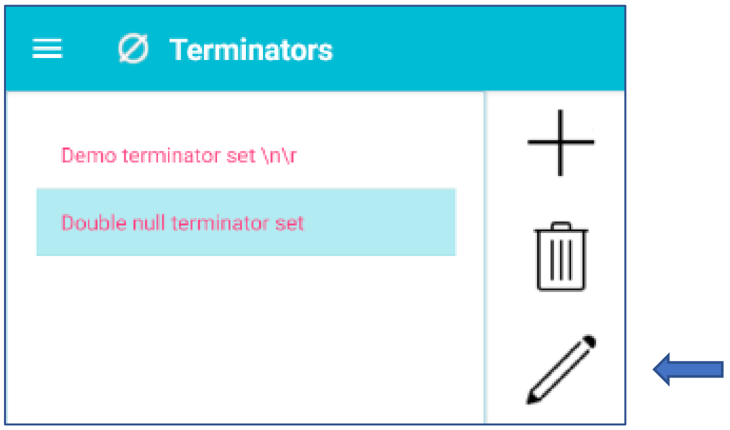
Double null terminator set

0x00	0x00
NUL	NUL

To remove the last terminator in the set, touch the delete icon. When you are satisfied, tap on the OK icon. The set will be saved, and you will be returned to the Terminators Page. Your new set will be posted in the list.



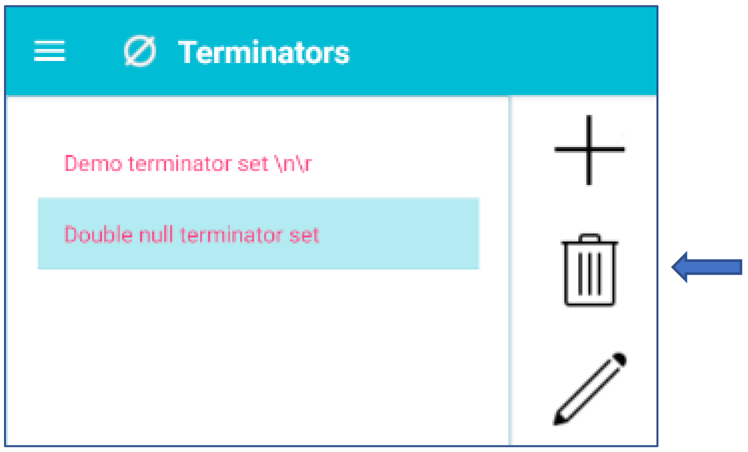
Edit Terminator set



Back at the Terminator Sets Page, tap on the Edit icon. You will be brought to the Terminators Edit Page where you can add or delete terminators and edit the set name.

Deleting a Terminator set

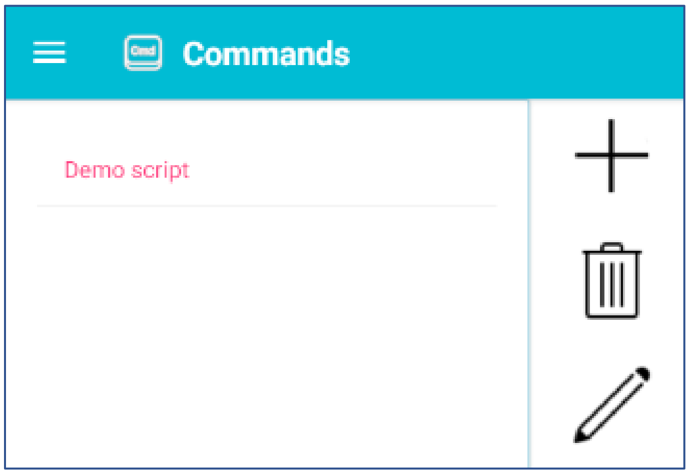
Back at the Terminator set list dialog, tap on the Delete button to delete the selected set



Commands

Commands

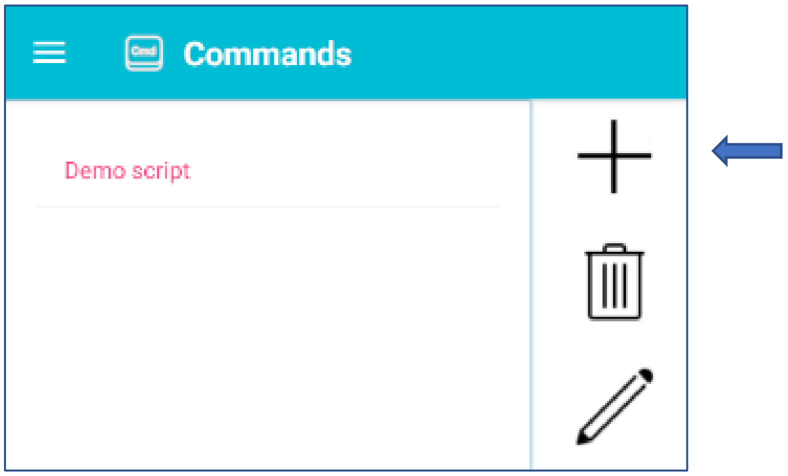
Tap *Commands* to bring up the Command Sets Page.



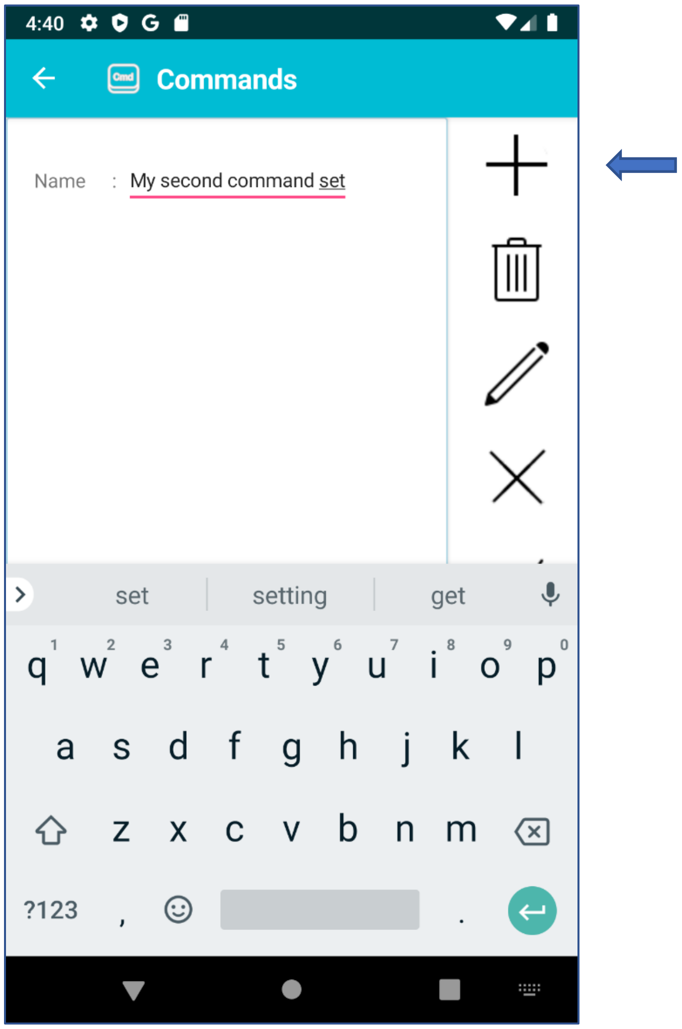
You can just tap on a Command set to select it as the current set and leave the page.

Adding a command list

Tap on the add icon to bring up the Command List Builder Page

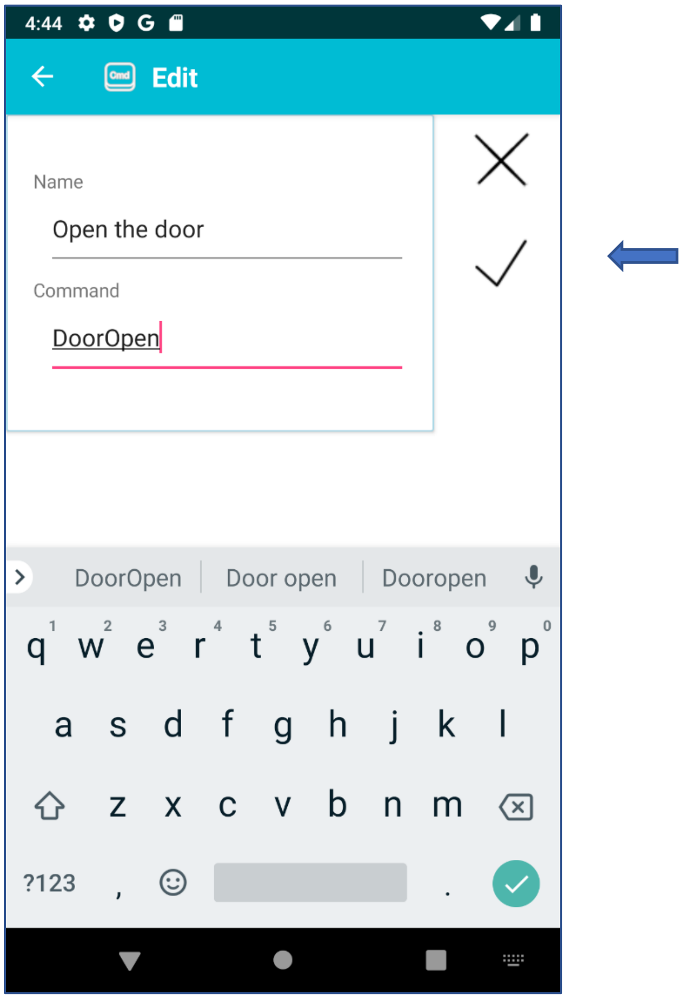


Edit the name in the Command Set Builder Page. To add a new Command to the set, tap the add icon



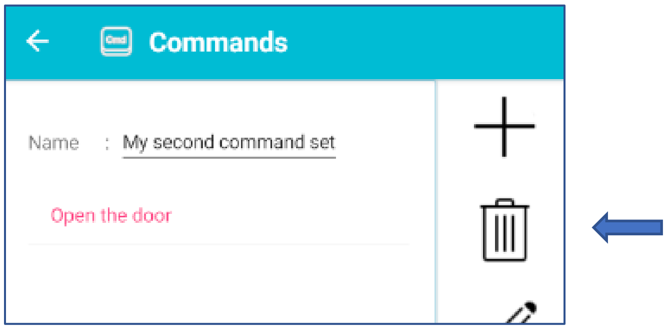
Name : My second command set

Adding a command



On the Command Builder Page, edit the command name and the actual command text that will be sent to the device. Tap the OK icon to save

On returning to the Command Set Builder page you will see your new command in the list. Add as many as you want

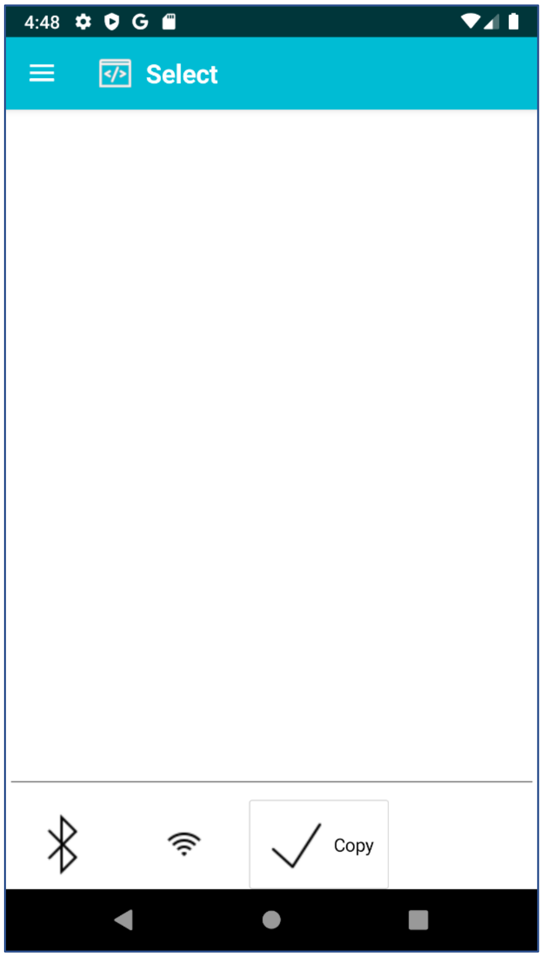


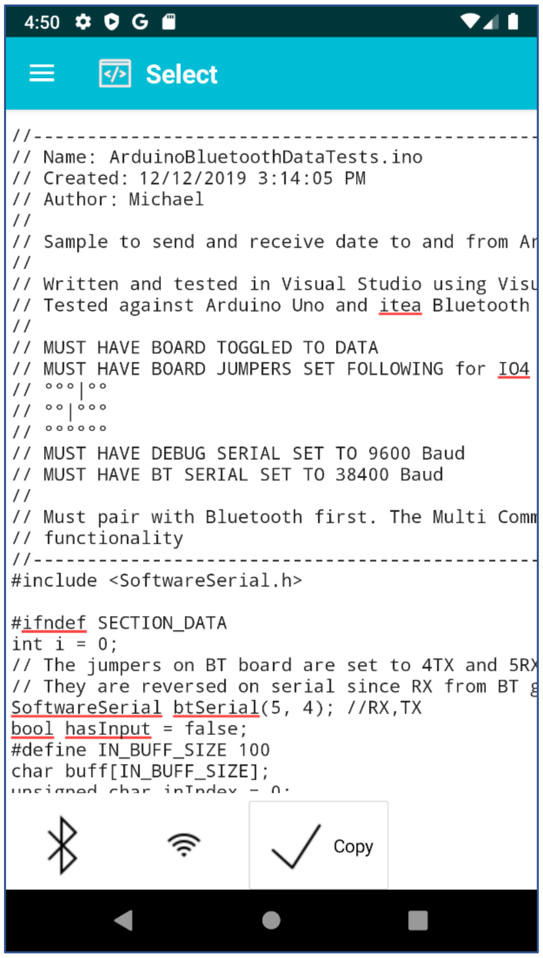
You can tap on the delete icon to remove the command from the list

Code Samples

Code Samples

Tap *Code Samples* to view Arduino code samples that receive commands and return responses. Tap on the Bluetooth or WIFI icons below to see the code.





```
//-----------------------------------------------
// Name: ArduinoBluetoothDataTests.ino
// Created: 12/12/2019 3:14:05 PM
// Author: Michael
//
// Sample to send and receive date to and from Ar
//
// Written and tested in Visual Studio using Visu
// Tested against Arduino Uno and itea Bluetooth
//
// MUST HAVE BOARD TOGGLED TO DATA
// MUST HAVE BOARD JUMPERS SET FOLLOWING for IO4
// °°°|°°
// °°|°°°
// °°°°°°
// MUST HAVE DEBUG SERIAL SET TO 9600 Baud
// MUST HAVE BT SERIAL SET TO 38400 Baud
//
// Must pair with Bluetooth first. The Multi Comm
// 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 g
SoftwareSerial btSerial(5, 4); //RX,TX
bool hasInput = false;
#define IN_BUFF_SIZE 100
char buff[IN_BUFF_SIZE];
unsigned char inIndex = 0;
```

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

Once you have the code listed, you can tap 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.

Bluetooth Classic Connections

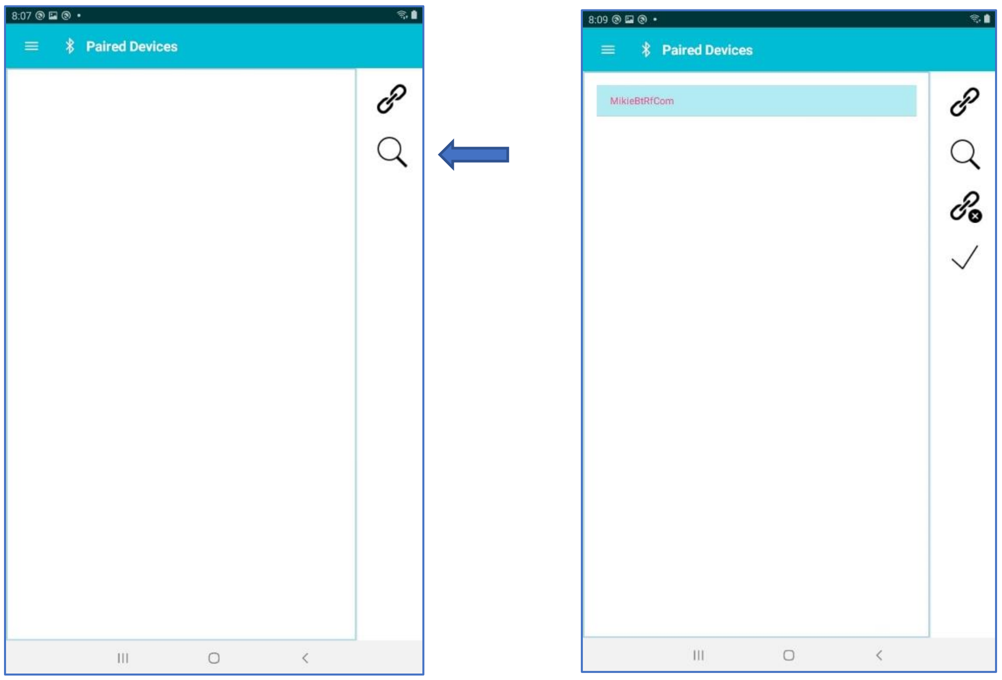
Tap the menu item to get started

Bluetooth

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 paired devices by tapping on the Discover icon. Afterwards select a device for more options



Open the page to pair Bluetooth Devices

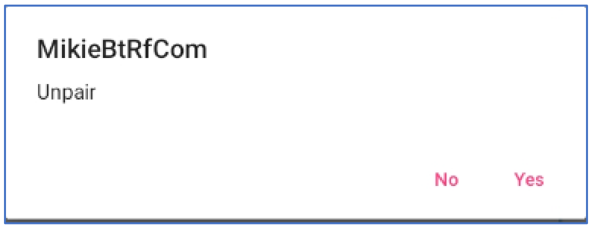
Discover paired devices to fill the list

Unpair selected device

Open the Run Window with selected device

Unpair Device

Tap on the UnPair icon to un-pair the selected device. You will have to confirm your choice

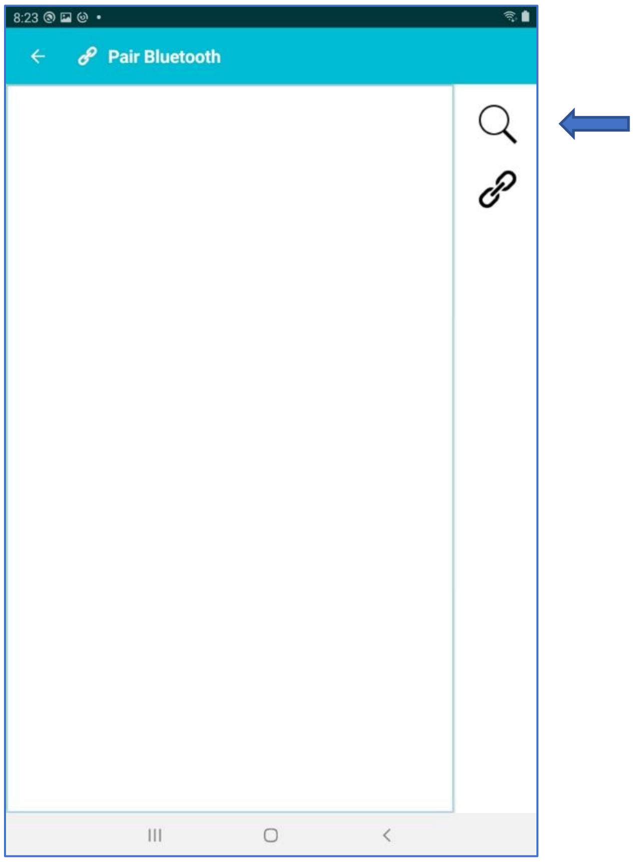

MikieBtRfCom

Unpair

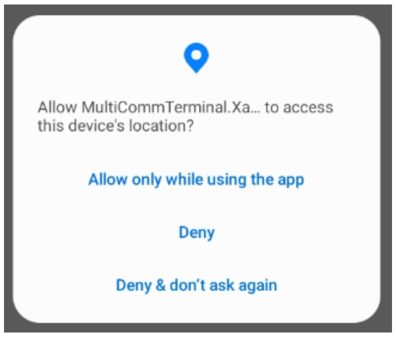
Tap Yes to un-pair the device. It will disappear from the list

Pair Bluetooth Devices Page

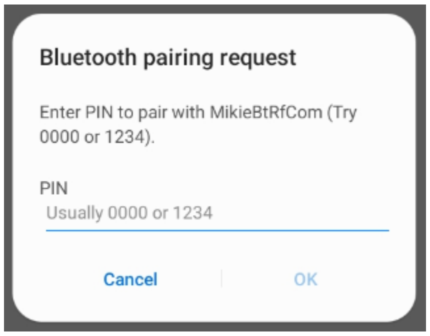
Tap on the Pair Icon to Open the pair page. Once the Pair Page is open, tap on the discover icon to get a list of non-paired devices



On first usage, the OS will bring up a dialog requesting permission to use the location. This is an OS requirement. You must tap *Allow…* to get a list. Otherwise, you cannot use this functionality. If you tap *Deny & don't ask again*, you will have to go into the App settings to allow it manually.


Allow MultiCommTerminal.Xa... to access this device's location?

Once you have non-paired devices in your list, select one and tap on the pair icon. The OS will put up a dialog to enter a PIN if it is required


Bluetooth pairing request

Enter PIN to pair with MikieBtRfCom (Try 0000 or 1234).

PIN

Usually 0000 or 1234

Enter the PIN and tap on OK

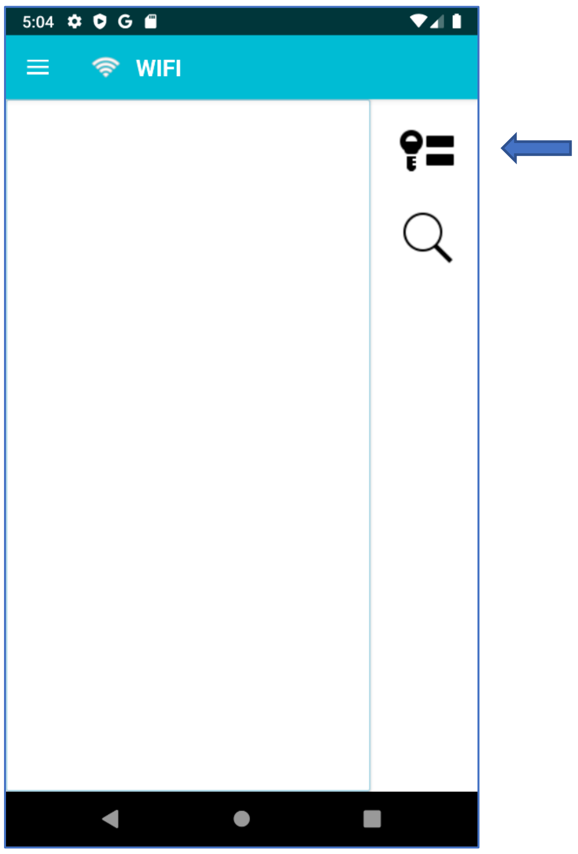
Select to Run

Back on the Paired Devices Page, select a device and tap on the Select icon to open the Run page. It is the same as the run page for WIFI, and it is described there in detail

WIFI Connections

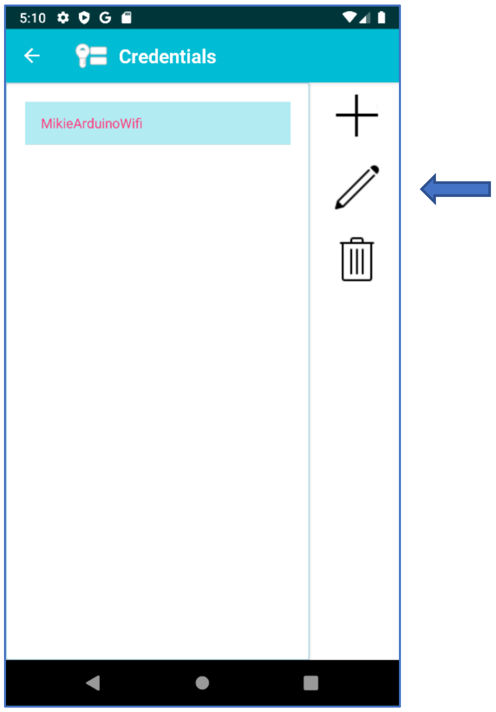
To start tap the WIFI menu item

WIFI

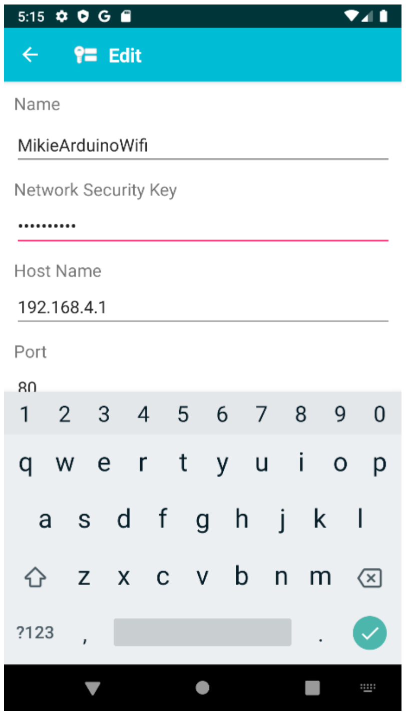


Credentials

To view or add connection credentials tap on the Credentials icon



The Credentials List Page provides a way to edit or delete credentials and connection parameters for previous WIFI connection. Tap on the credentials, then on the edit icon to view the contents

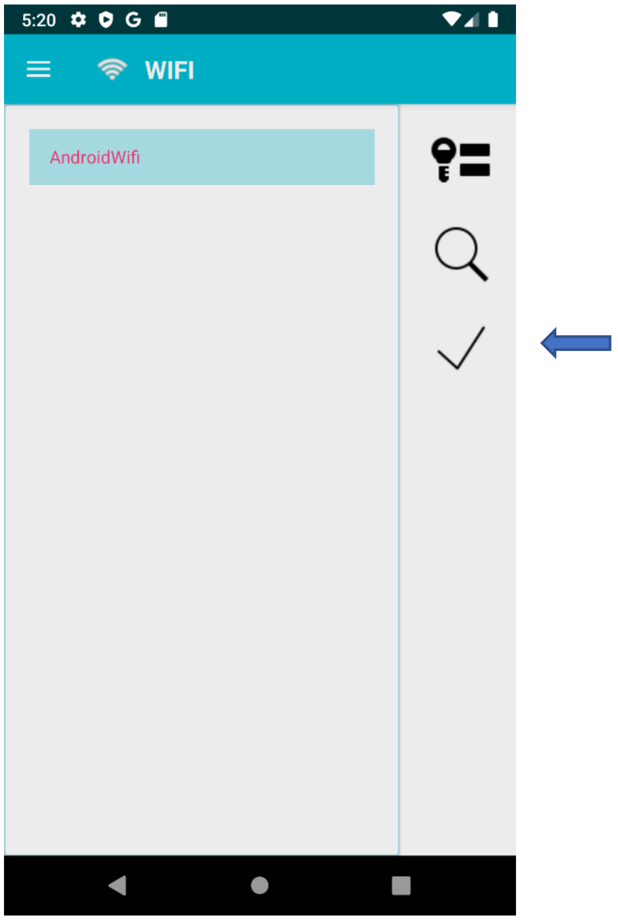


Name
MikieArduinoWifi
Network Security Key
••••••••••
Host Name
192.168.4.1
Port
80

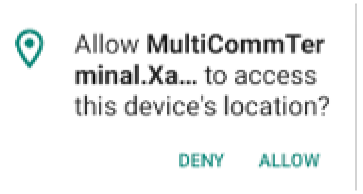
You can view or edit the credentials. Tap on the OK sign to save or the X to cancel edits. The security key is the password for the WIFI access point (like a router). The Host name/IP or those that you have exposed on your embedded device. See the Arduino WIFI code sample in the appendix.

Discovery

Tap on the search icon to view the WIFI networks within range.



On fist usage you will get an OS dialog requesting Location Permission. This is required by the OS.



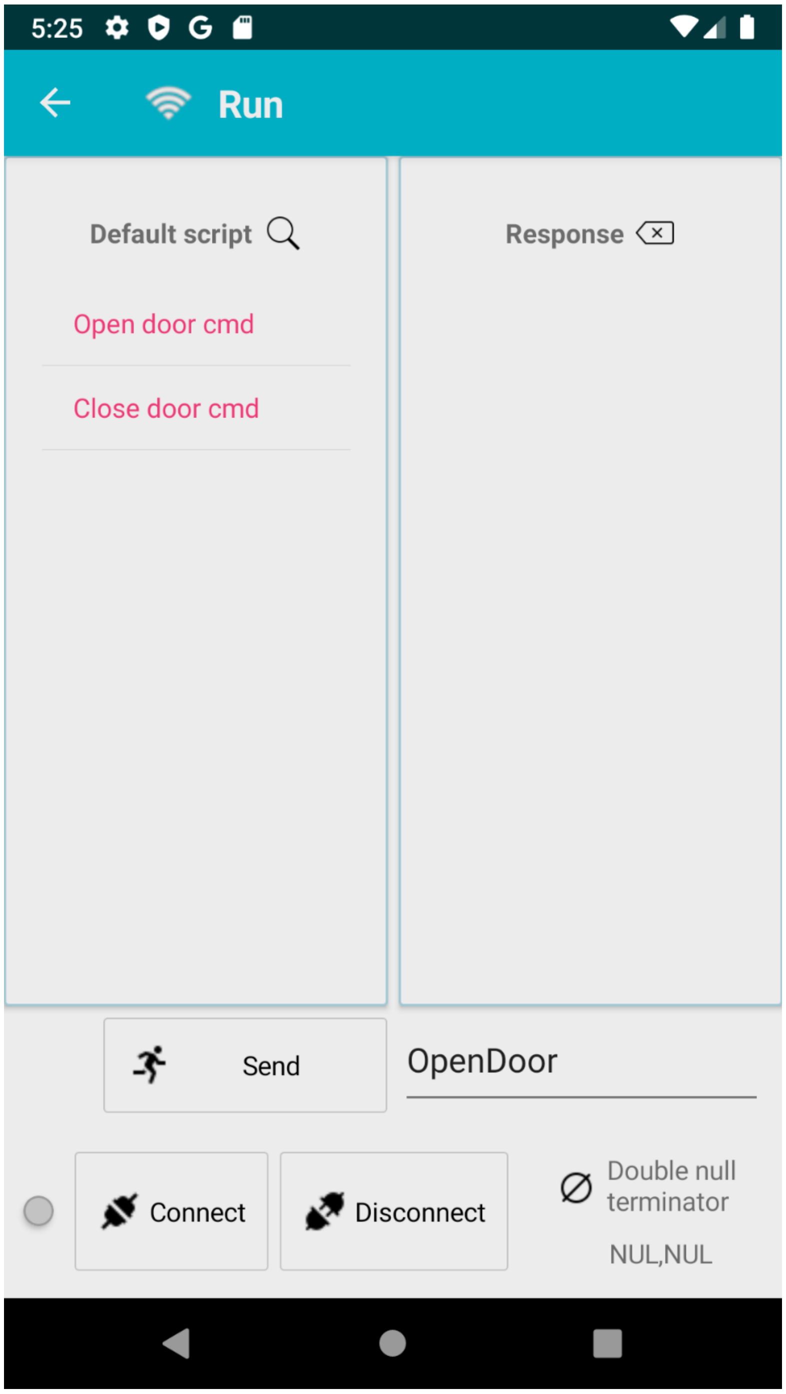
Allow **MultiCommTerminal.Xa...** to access this device's location?

Tap on Allow, otherwise you will not be able to scan a list of WIFI access points.

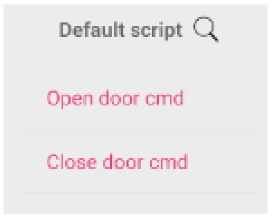
Select the Access point and tap on the OK/Select icon to bring up the run page

WIFI Run Page

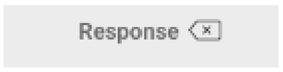
This is the heart of the App. Where you can establish a connection and run the commands you have built with your terminator set



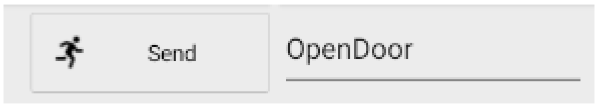
Run Page Elements



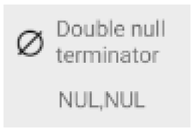
The currently selected command set. The name of the set is bold at the top. If you tap on it, a popup will allow you to select a different set. Each link below is a command. If you tap on it, it will be transferred to the send box



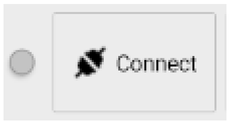
The header at the top of the responses list. This is where you will see the messages coming from the connected device. You can scroll the responses, or, If you tap on this header it will clear the list



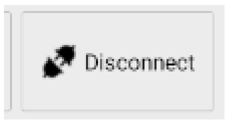
The send box and button, This is the actual text that will be sent to the device. Your current selection of terminators will be added to it. You can type in any text in the box and send it



The current terminator set that will be added to outgoing messages. Incomming messages will require those same terminators to be recognised. If you tap on this a popup will allow you to select another terminator set



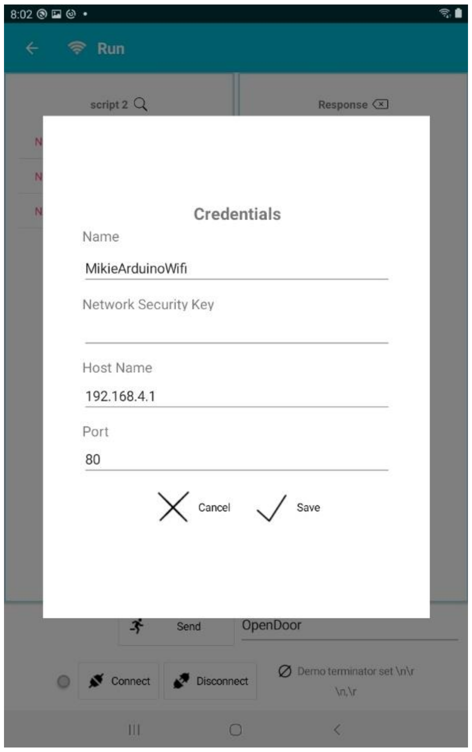
The connect button which establishes connection with the device. The light to the left will go green when connection is established.



The disconnect button to break the connection. The connection is also broken when you navigate away from the page

Connection

If you do not have credentials already entered, you will get a popup requesting the information. This will be saved for future use. If you enter the wrong information you will have to go back to the Credentials list and edit or delete them.



Credentials

Name
MikieArduinoWifi

Network Security Key

Host Name
192.168.4.1

Port
80

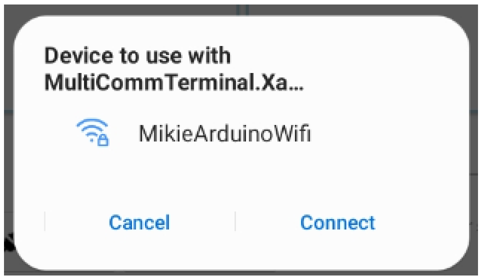
OpenDoor

- The first entry is the SSID of the network you selected. Filled in for you
- Next is the Network Security Key area (password).
- Then the host name (IP)
- Finally, the port

Tap on Save to continue connection. You can now send and receive messages.

Note: parts of the popup will disappear when the keyboard comes up

On Connection the OS puts up a dialog to confirm you want to connect to the device



Device to use with MultiCommTerminal.Xa...

MikieArduinoWifi

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* (0xoD) 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 date 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 termintor sequence is detected.
                            // You would then
                            //  - Look at the message
                            // - Determine operation requested
                            // - Do the operation
                            // - Optionaly, 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");
}
```