ProtoControl *User Manual*

Revision 1 May 2024

## *Overview*

**Features**

1. Compatible with industry standard microcontrollers
2. UART port
3. 2.8” Capacitive TouchScreen Display
4. 4 I2C Ports for Hardware Peripherals
5. Magnetic Connectors for Hot-Swapability

**Applications**

1. Testbench for electrical components
2. Display and Controller for Embedded Systems
3. Prototyping device for hobbyists
4. Display for network of analog sensors

**Device Information**

| Parameter | Value |
| --- | --- |
| Voltage | 5V |
| Max Current Draw | 200mA |
| Main Panel Dimensions (LxWxH) | 108 x 82 x 22 mm |

**Description**

Protocontrol is a plug and play panel with a start-to-finish user interface development experience. Protocontrol consists of a touchscreen display, four I2C ports for hardware peripherals, and a UART channel to communicate with the User's device. Customizing the touchscreen's Profile is as simple as using the website to drag and drop icons onto a grid that maps to the touch screen. Both the digital and hardware peripherals can be used as controllers for the user system and the touchscreen can also be used to display values from the user system. Protocontrol is compatible with commonly used microcontrollers such as Arduino. This device is ideal for engineers who need to quickly develop a user interface to test their product and for hobbyists who need a system that is easy to use and can help manage complexity.

**

*Figure 1: ProtoControl Primary Panel*

## Table of Contents

[**Overview 2**](#_g6txs2b9ubik)

[**Table of Contents 3**](#_9es30rx6e2v3)

[**Section I: Out of the Box Setup 4**](#_4hzf52vbxop8)

[A. Your Kit Includes: 4](#_lh7npa5hjzx1)

[B. Power Requirements 4](#_p8bacwcy705p)

[**Section II: Initially Connecting to Wi-Fi 5**](#_k01zulor16sg)

[**Section III: Configuring the Touchscreen 8**](#_6nozjb6mizw7)

[A. Accessing the Website 8](#_7kb81shfnfdt)

[B. Digital Peripherals 8](#_13on9jr6y6jo)

[C. Device ID 9](#_wamkui21b5ey)

[D. Profile Number 9](#_t4ynfbbzara7)

[E. Component ID 10](#_xscdnh8z2rpk)

[F. Customizing a Profile 11](#_4d9avjma5ry7)

[G. Generate Touchscreen Profile and Upload to Protocontrol 11](#_abv6vydht8rw)

[H. Updating Profile on Website 12](#_b6s2ux4ylgor)

[I. Refreshing/Cycling Through Multiple Profiles on Device 12](#_i7rko0rfc7w2)

[J. Working with Multiple Devices with Multiple Profiles 13](#_5wzle4i1v7n4)

[**Section IV: Configuring and Debugging I2C Peripherals 13**](#_tyiaxqjj08xg)

[A. Live Editing Hardware Peripherals 14](#_fom593f6a2sj)

[B. I2C Port Layout 14](#_jyuvm91dnmlb)

[**Section V: UART Basics 16**](#_i0d2l0gk9rth)

[**Section VI: Coding on the User Side 20**](#_n8tkayalqe7t)

[Section VII: User System Example 20](#_alw2ds8dlbl3)

[A.User System Without Protocontrol 20](#_lgetpl99n91v)

[B.User System + ProtoControl 23](#_w3xlrhmmhgvl)

[Step 1: Create a string to store messages from Protocontrol 24](#_1i8sqc8kj2d1)

[Step 2: Set up communication with serial port (UART) 24](#_2mnmqgsbsj08)

[Step 3: Create an outline of a case statement in void loop 25](#_heqbbajj8dan)

[Step 4: Parse out Component ID 26](#_t01kk12i3478)

[Step 5: Use data from Protocontrol in your system 26](#_xuxn3cz6l77)

[Step 6: Final Script 28](#_ws0qdd5od4na)

[**Section VIII: Debugging 31**](#_mx0p4f212kiz)

[A. Using Serial Monitor as a Debugging Tool 31](#_je13j65ekvzl)

[**Section IX: Terminology 31**](#_2xlx3uj9vrkq)

[**Appendix 31**](#_dil9hkmk0u5g)

[Appendix A: Arduino Background 31](#_xra737ve1w5n)

[A. Download Arduino IDE 2 for your device 31](#_mhc324fnosfh)

[**References 32**](#_kvhopcn6dnu5)

## Section I: Out of the Box Setup

Figure 2: Protocontrol Kit

### A. Your Kit Includes:

1. Main Board
2. Keypad Board (0x39)
3. Rotary Encoder Board (0x3A)
4. Joystick (0x48)
5. UART Cable
6. USB A - USB C
7. USB A - Barrel Jack
8. Barrel Jack Port
9. Serial (UART) Port

Figure 2: ProtoControl Kit Components

### B. Power Requirements

Protocontrol needs 5V and can be powered via the Barrel Jack or via the Vin and Ground pins near the Rx and Tx pins (see figure 3).

### 

*Figure 3: Protocontrol Main Panel and Ports. The UART Port consists of Vin, TX, RX, Gnd (in that order). There are 4 I2C ports, one on each side of the board.*

## Section II: Initially Connecting to Wi-Fi

1. When you power ProtoControl for the first time, it will not be connected to a wi-fi network. A textbox will pop-up indicating that we must complete the wi-fi setup process.
2. A new access point will be opened called “ProtoAP”. The password for the access point is “password”
3. Use your laptop/tablet/phone to connect to the ProtoAP wi-fi network.
4. An IP address will display on touchscreen once you connect to the Access Point



Figure 4: Information displayed on ProtoControl after connecting to the ProtoAP wi-fi network.

1. Go to http://[IP address]
2. You will be prompted for SSID and password (enter your network of choice)
3. There will also be an empty field for Device ID. Choose a Device ID of your choice. It is recommended that you select a Device ID with only numbers and letters.

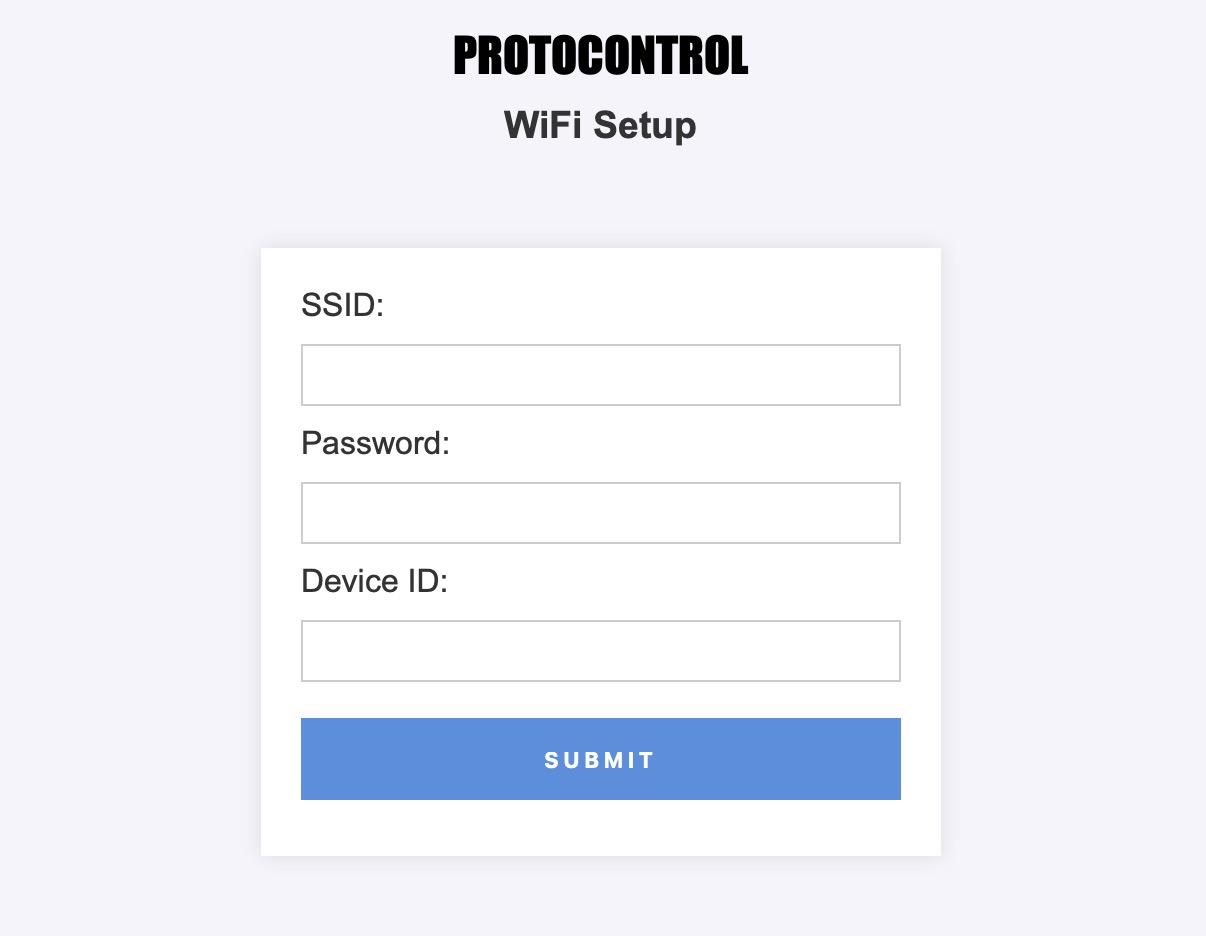


Figure 5: *WiFi Setup site*

1. Click Submit.
2. Once we are successfully connected to wi-fi, the Access Point closes. When you look at available networks, “ProtoAP” should no longer be visible.
3. The touchscreen populates with the last Profile associated with the Device ID or if no Profiles have been created, the touchscreen populates with a blank Profile.



Figure 7: ProtoControl screen after successfully connecting to wi-fi.

1. If you have not successfully connected to a wi-fi network, the Access Point will remain open. Try again until you connect to a network.

## Section III: Configuring the Touchscreen

### A. Accessing the Website

The website can be accessed [here](https://pages.github.ncsu.edu/ECE-Senior-Design-Fa2023-Sp2024/38-Protocontrol/).

### B. Digital Peripherals

| **Icon** | **Settings** |
| --- | --- |
|  | **Button**   * **Occupies 1 Cell** * **Send Data to User System**   **Editable Properties**   * **Button Color** * **Text Color** * **Text (4 char. Max)** |
|  | **Slider**   * **Occupies 3 Cells horizontally** * **The slider can only be oriented horizontally** * **Send Data to User System**   **Editable Properties**   * **Bar Color** * **Slider Color** * **Min. Value** * **Max. Value** |
|  | **Toggle**   * **Occupies 1 Cell** * **Send Data to User System**   **Editable Properties**   * **Text Color** |
|  | **Increment/Decrement Button**   * **Occupies 1 Cell** * **Send Data to User System**   **Editable Properties**   * **Text Color** * **Min Value** * **Max Value** |
|  | **Terminal**   * **Occupies 1 Cell** * **Display Data from User System**   **No Editable Properties** |

### C. Device ID

Each ProtoControl Panel has a unique, user-selected Device ID (see Section II). Protocontrol makes a request from the server looking for items in the database that correspond to it’s device idea and the current profile. If the Panel does not see its own Device ID, then it ignores the incoming data. When you generate a Profile, enter the Device ID for the ProtoControl you want that Profile to be visible on. Be careful of typos (blank spaces, capitalization, etc.).

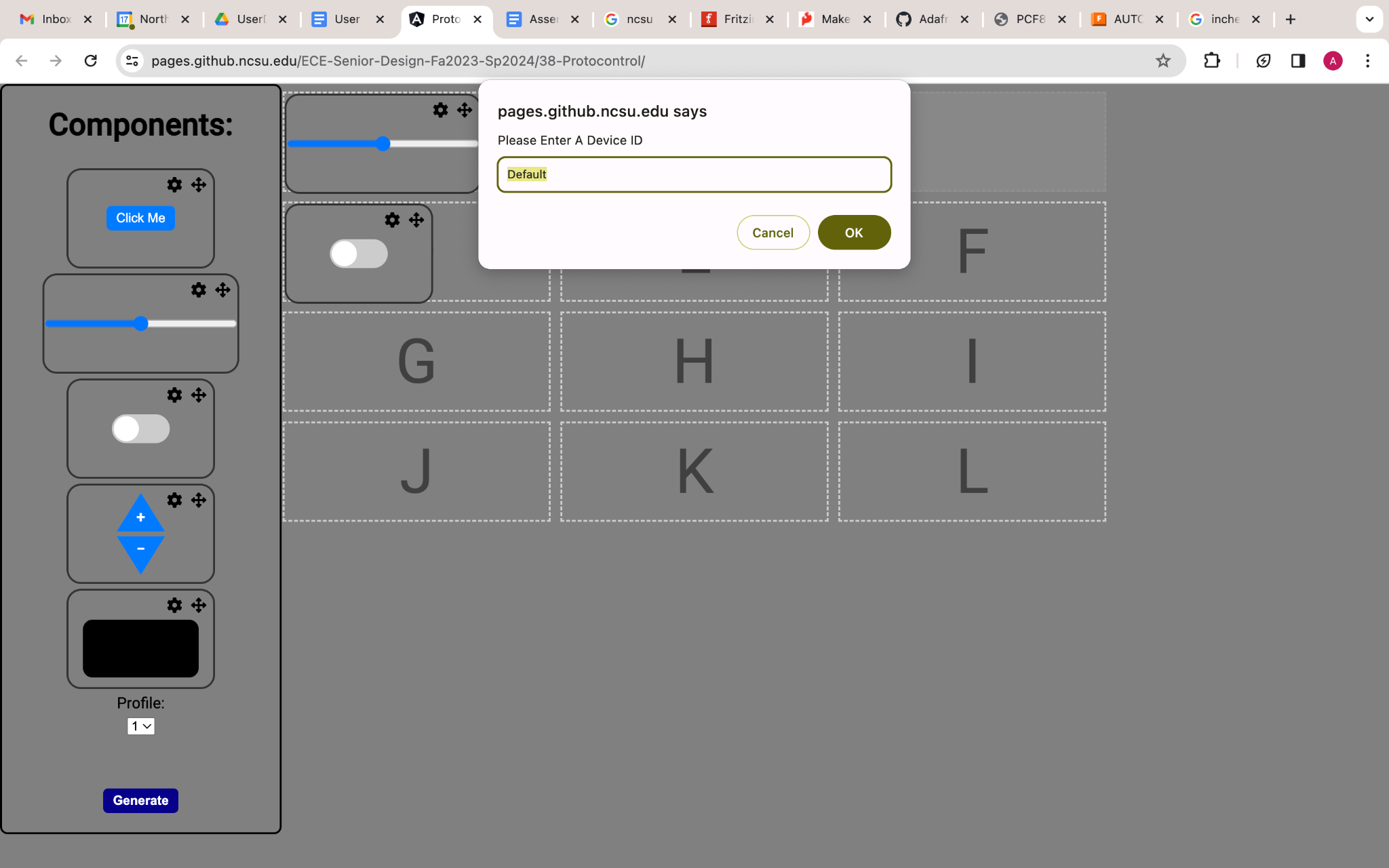
****

Figure 8: Choosing your Device ID on the website

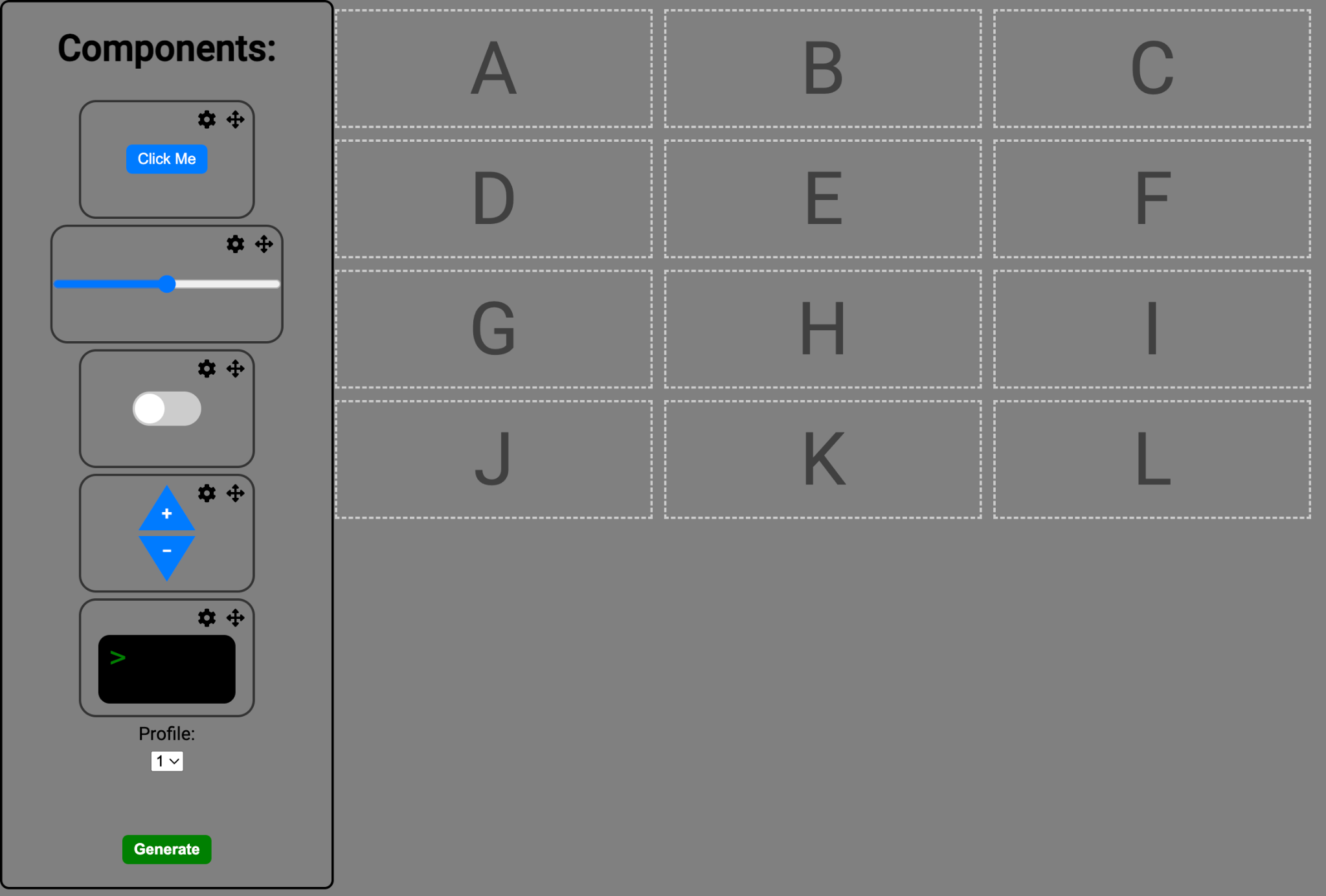
### D. Profile Number

Each ProtoControl Panel can have up to 5 Profiles saved on the device. You can change which Profile you are working on by clicking the dropdown arrow and selecting your Profile of choice.



Figure 9: Dropdown to select which profile you want to edit

### E. Component ID

****

*Figure 10: Component ID Map and Touchscreen Profile.*

The touchscreen display is divided into a 4x3 grid. Each cell has a Component ID (A-L) Each drag-and-drop component on the website takes up *at least one* of these cells. It is not possible to use a fraction of a location. Some components (ie slider) take up more than one cell. For example, if a slider is placed on cell D, cell E & F are disabled and it is not possible to place a component on cell E or F. Although the slider visually takes up 3 cells, its Component ID is G (see Figure 11 below).

### F. Customizing a Profile

1. Drag and Drop Components to your desired cell.
   1. Components that take up more than one cell will disable surrounding cells 

Figure 11: Slider placed in location G

1. Click the ⚙ Icon to modify component settings
2. After modifying component settings, press the green save button



Figure 12: Slider properties such as color and data ranges can be edited.

1. Currently, it is not possible to replace a component selection without clearing the whole grid. Clearing the whole grid can be accomplished by refreshing the website.

### G. Generate Touchscreen Profile and Upload to Protocontrol

Once you are happy with your Profile, click the green “generate” button. This converts the Profile information into a string. The string/Profile is identified by a Profile Number (between 1-5), and a Device ID associated with it. After clicking “generate” it should take no more than 30 seconds for the database to update with your string and identifier.

The string is sent over wifi to Protocontrol. After initially setting up the wi-fi connection, Protocontrol will automatically connect to that network upon startup (unless that network is not detected).

### H. Updating Profile on Website

To modify an existing Profile:

1. Select a Profile [see section D].
2. Make the edits [See section F].
3. Click Generate.
4. Enter the Device ID of your ProtoControl Panel.

By clicking on “generate”, a new string (including unique identifier, Profile number, and Device ID) is created and saved in the database. Although the new Profile is saved to the database, it hasn’t been updated on the device itself (see next section).

### I. Refreshing/Cycling Through Multiple Profiles on Device

If you want to cycle through the five Profiles saved to your Device, press the lower right button. As you click through each Profile, Protocontrol pulls the latest version of that Profile.

Suppose you have created 3 Profiles for a ProtoControl Panel with a Device ID of “Device1”.



Figure 13: Three different profiles all connected and viewable on the same Panel.

You can press the button and view all three Profiles in a loop.

Now, if you edit Profile 2 of “Device 1” on the website and click “generate”, the update will be sent to the database. If you refresh the website you can start on a blank slate.



Figure 14: Editing profile 2

If you press the lower right button on ProtoControl to go from Profile 1 to 2, Protocontrol reads the original Profile 1 and 3 associated with “Device 1”. However, instead of reading the original Profile 2 of “Device 1”, it reads the latest “Device 1” Profile 2 and updates the touchscreen display.

In short, Profiles are updated via the website and the touchscreen is refreshed by pressing the lower right button on the Panel.

### J. Working with Multiple Devices with Multiple Profiles

When you are working with multiple ProtoControl Panels, updating a Profile for one Panel (“Device 1”) will not interfere with the other (“Device 2”).

## 

## 

## 

## 

## 

## 

## Section IV: Configuring and Debugging I2C Peripherals

Digital hardware peripherals have addresses ranging from 0x39-0x3F. Analog hardware peripherals have addresses ranging from 0x48-0x4B. Upon initialization, Protocontrol attempts to begin I2C communication with the specified address (see Table 1. If a peripheral *is not* connected to Protocontrol, the function returns false and Protocontrol knows that the peripheral is not connected. If a peripheral *is* connected to Protocontrol, the function returns true and the Protocontrol knows that the peripheral is connected.

// Example code to check if component is connected

// flag to indicate if the rotary encoder is connected

bool encoder = false;

// create an instance of the gpio pin expander with I2C address of 0x38

PCF8574 pcf8574(0x38);

// pcf8574.begin () attempts to initialize communication with the pin expander

if (pcf8574.begin() == false) {

encoder = false;

}



| Hardware Peripheral | Component ID | Address | Data Type |
| --- | --- | --- | --- |
| Rotary Encoder | M | 0x39 | No limit |
| Keypad | N | 0x3A | 0-9 (discrete) |
| Joystick X | O | 0x3B | (continuous) |
| Joystick Y | P | 0x48 | (continuous) |
| Slider | Q | 0x49 | (continuous) |

Table 1: Hardware Peripherals and their respective A=addresses

### A. Live Editing Hardware Peripherals

How to Swap Out Hardware Peripherals

1. Remove old Hardware Peripheral (connected to the main board via magnetic panel)
2. Snap on new Hardware Peripheral
3. Power off and on ProtoControl (either disconnect power supply or press the reset button on the microcontroller)

### B. I2C Port Layout

See Figure 1 in the Overview Section for the I2C Port Layout on the main ProtoControl Panel.

## 

Figure 15: I2C Port Layout for Upright Component Peripheral

## 

## Section V: UART Basics

UART (Universal Asynchronous Receive Transmit) is a communication protocol that uses two wires (Receive and Transmit) to communicate data. UART can support a maximum of one Controller and one Peripheral. Data is transferred asynchronously (without a clock).

UART Transmits data in the form of packets, and each packet follows a set structure:

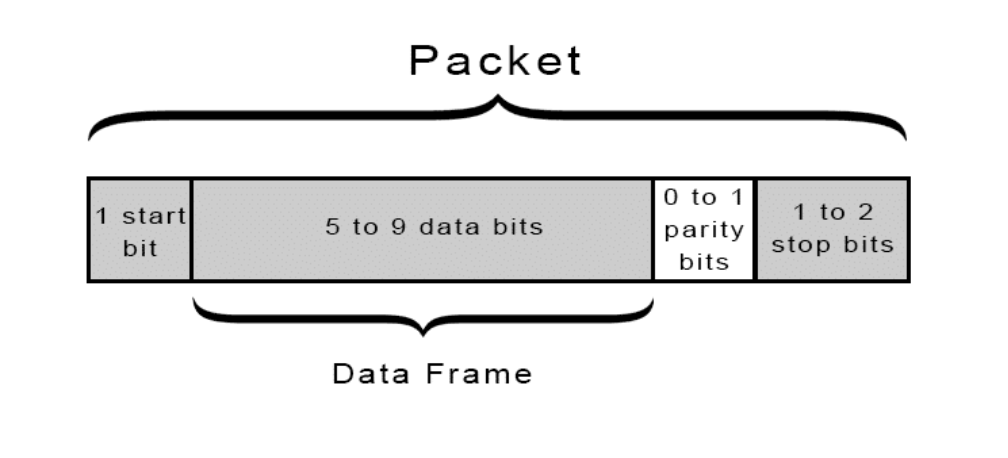


Figure 16: Data Packet Format [1]

**START:** The UART transmission line is held High when data is not being transmitted. The Start bit of the data packet pulls the line Low to indicate that transmission has begun. The receiving UART detects the voltage change and starts reading data at the specified baud rate.

**DATAFRAME:** The data frame can hold 5-9 bits of data.

**PARITY:** The parity bit is an optional bit in the packet that is used for error detection that works by comparing the data on the transmit and receive side to see if any data was corrupted during transfer.

Once the UART receives the data frame, it tallies the number of bits set to 1 and determines whether the sum is even or odd. In case of even parity (where the parity bit is 0), the count of 1 bits in the data frame should be even. Conversely, for odd parity (where the parity bit is 1), the count of 1 bits in the data frame should be odd. Matching the parity bit with the data ensures error-free transmission. However, if the parity bit and the calculated sum do not align - such as when the parity bit is 0 but the sum is odd, or vice versa - the UART detects changes in the data frame [1,2].

**STOP:** The sending UART signals the end of transmission by pulling the TX line low for one or two bits duration.

Start and Stop bits are used to synchronize data Transmitted from one UART device and data Received by the other UART device.

Baud Rate

The Baud Rate is the frequency with which bits are transmitted. The baud rate of the transmitter and receiver must be within 10% of each other otherwise the timing will deviate too much.

| Advantages | Disadvantages |
| --- | --- |
| 2 Wires | Data frame limited to 9 bits |
| Error Checking (Parity Bit) | Baud Rate of Transmitter and Receiver must be within 10% of each other |
| Ideal for Debugging and Verification of Products | Limited to a single controller and peripheral |

Table 2: Advantages and Disadvantages of UART Protocol

### A.Set Up UART for Arduino



Serial.begin(9600); //init communication over USB

2

Serial1.begin(9600); //communication over RX/TX pins



### B.Set Up UART for Raspberry PI



import serial

import time

data\_var = 0

# Function to send data saved in data\_var

def send\_data():

ser.write((data\_var + '\n').encode())

# Open serial connection to the printer

# baud rate = 115200

ser = serial.Serial('/dev/ttyUSB0', 115200, timeout=1)

# Change COM3 to your printer's COM port

com = serial.Serial('/dev/ttyS0', 115200, timeout=1) # Change COM3 to your printer's COM port

# Integrating Raspberry Pi with ProtoControl

# User System receives data from digital peripheral in location 'G'

while(1):

if(com.inWaiting() > 0):

data = com.readline().decode().strip()

print(len(data))

if(data.startswith('G')):

data = data[1:]

send\_data(int(data),50)

# Close serial connection when you are done with UART

ser.close()



### C.Integrating MSP 430 with ProtoControl

// User System receives data from digital peripheral in location 'C' (data used to control LED) and 'G'(data used to control Servo)

void processMessage(void)

{

switch (msg[0])

{

case 'G': // Left Forward

LEFT\_FORWARD\_SPEED = ((int)msg[1]) \* 5000;

HEXtoBCD(msg[1]);

adc\_line(3, 5);

break;

case 'C': // Red On

if (msg[1] == '1')

{

P1OUT |= RED\_LED;

strcpy(display\_line[1], "REDLED ON ");

}

else if (msg[1] == '0')

{

P1OUT &= ~RED\_LED;

strcpy(display\_line[1], "REDLED OFF");

}

break;

default:

break;

}

int i;

for(i=0;i<sizeof(msg);i++){

msg[i] = ' ';

}

}



## 

## 

## Section VI: Coding on the User Side

Data from the digital and hardware peripherals is sent over UART as a string of characters in the following format:

**+[component ID]|[value]**

**Component ID**

- Each component is assigned a unique *component identifier* assigned by the website that is **known to the user**

**Digital Component**

Tag assigned by grid location on website

**Hardware Component**

Tag is associated with Hex Address of each peripheral

## Section VII: User System Example

### A.User System Without Protocontrol

The following user system involves a servo and an RGB led. The RGB led iteratively cycles through colors every 200ms. The servo motor is swept back and forth from 0-180\* in increments of 5 degrees every 200ms.The changes in the rgb led color and the servo position are based on time elapsed rather than user control. The user system also includes a light sensitive resistor whose value is printed on the serial monitor. The sketch has an LED initialized for debugging purposes though not used explicitly.



//example user system using Arduino Uno

//this sketch is compatible with any Arduino IDE enabled device

#include <Servo.h>

int sensorPin = A0; // Analog pin 0 for light sensitive resistor

int sensorValue = 0; // var stores new sensor value

int lastValue = 0; // var stores previous sensor value

int ledPin = 13; // Analog pin 13 on Arduino Uno

// Initialize an instance of "Servo" called "arm"

Servo arm;

int pos = 0;

bool clockwise = 1;

bool counterclockwise = 0;

// RGB led init variables

int redPin= A5; // Analog pin 5 on Arduino Uno

int greenPin = A6; // Analog pin 6 on Arduino Uno

int bluePin = A3; // Analog pin 3 on Arduino Uno

int red = 0;

int green = 0;

int blue = 255;

// function to change color of LED based on input vars

void setColor(int redValue, int greenValue, int blueValue) {

analogWrite(redPin, redValue);

analogWrite(greenPin, greenValue);

analogWrite(bluePin, blueValue);

}

void setup() {

Serial.begin(115200);

// LED Setup

pinMode(ledPin, OUTPUT); // declare LED pin as output

digitalWrite(ledPin, LOW);

// Servo Setup (Analog or Digital 9?)

arm.attach(9);

// RGB LED setup

pinMode(redPin, OUTPUT);

pinMode(greenPin, OUTPUT);

pinMode(bluePin, OUTPUT);

// Debugging LED setup

pinMode(4,OUTPUT);

}

void loop() {

// set the color with local ints

setColor(red, green, blue);

// sweep color from blue to cyan

if (green < 255) {

green++;

}

// sweep servo from 0 to 180

if (clockwise && pos < 180) {

pos++;

arm.write(pos);

} else if (clockwise && pos == 180) {

clockwise = false;

counterclockwise = true;

} else if (counterclockwise && pos > 0) {

pos--;

arm.write(pos);

} else if (counterclockwise && pos == 0) {

clockwise = true;

counterclockwise = false;

}

// read the value from the light sensor

sensorValue = analogRead(sensorPin);

sensorValue = map(sensorValue, 0, 800, 0, 100);

// if it's changed, call sendData to update display

if (abs(lastValue - sensorValue) >= 4) {

lastValue = sensorValue;

Serial.print("sensor value: ");

Serial.println(sensorValue);

}

delay(200);

}



### B.User System + ProtoControl

Let’s modify our original script to include Protocontrol!

| Features of the Original User System | Features of the User System with Protocontrol |
| --- | --- |
| RGB LED sweeps from Blue to Cyan automatically | User control of RGB LED color with a slider on the touchscreen |
| Servo position is automatically swept | Use keypad and/or Rotary Encoder to control servo |
| Display the light sensitive resistor value on the serial monitor | Display the light sensitive resistor value on the touchscreen |

Table 3: User System with and without ProtoControl

In this case, Protocontrol is being used to introduce user control of the system via digital and hardware peripherals. While it is possible to have user control without a separate microcontroller, the necessary pins might not be available (keypad takes up 7 pins, touchscreen takes up 40 pins). Additionally, Protocontrol makes the final script more concise and easy to read.

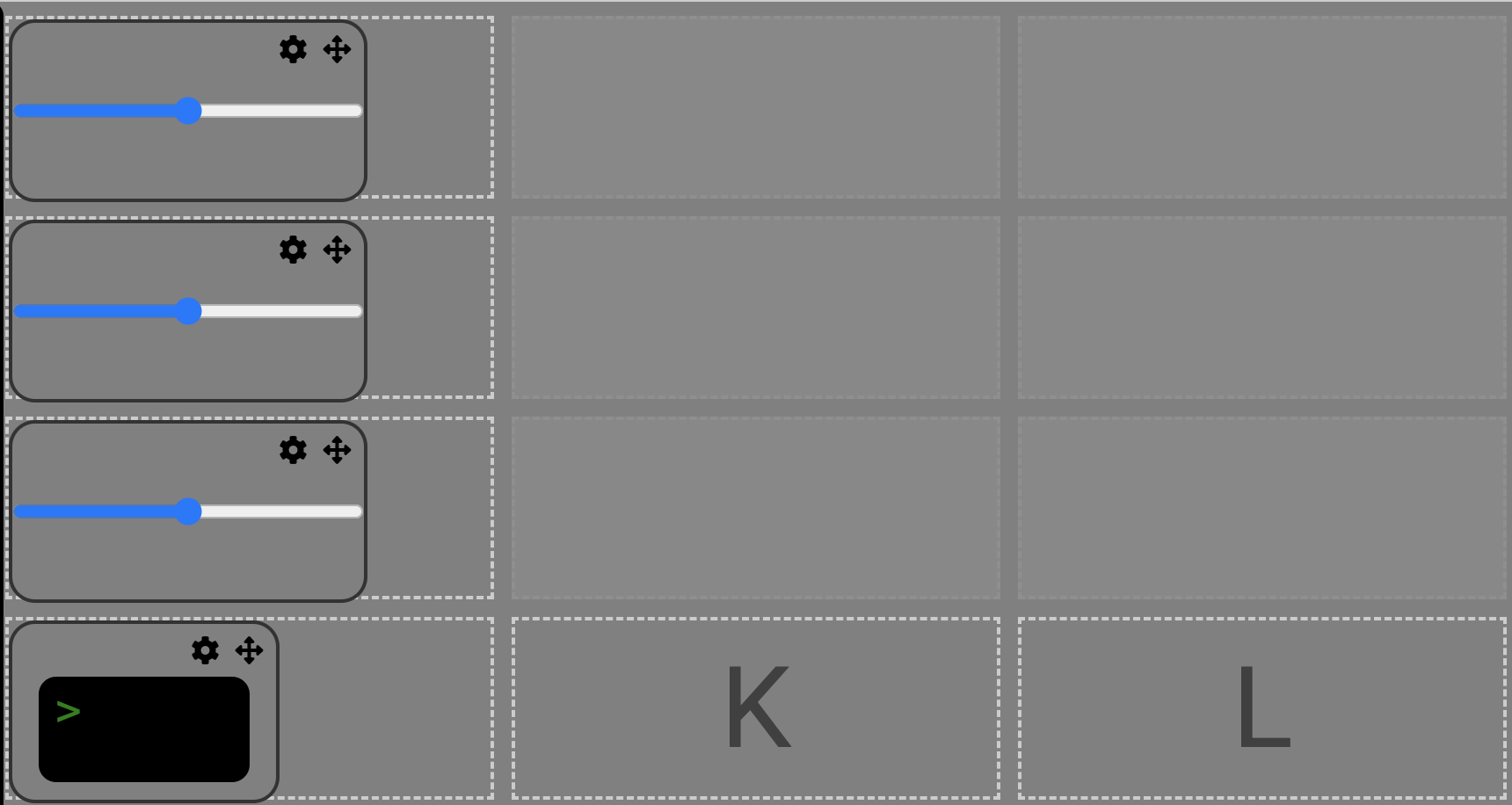


Figure 17: Profile associated with this User System

#### Step 1: Create a string to store messages from Protocontrol

int ledPin = 13;

// Insert the following:

String communication;

// the string will hold parsed data received over UART

// ...



#### Step 2: Set up communication with serial port (UART)

void loop () {

// ... Insert the following:

if(Serial.available()){

// parse out null character

communication = Serial.readStringUntil('\n');

// print out command from Protocontrol onto serial monitor

Serial.println(communication);

}

//..

}



#### Step 3: Create an outline of a case statement in void loop

Create a “case” (if-statement) for each expected command from Protocontrol.

|  | Component ID | Command  +[component tag]|[value] |
| --- | --- | --- |
| Slider 1 (Red) | A | +A255 |
| Slider 2 (Green) | D | +D50 |
| Slider 3 (Blue) | G | +G137 |
| Rotary Encoder | M | +M-53 |
| Keypad | N | +N6 |

The component ID of physical peripherals is fixed based on each component’s I2C address. The component ID of the digital peripherals is based on how you chose your Profile.

void loop () {

if(Serial.available()){

communication = Serial.readStringUntil('\n');

Serial.println(communication);

//arm.write(communication.toInt()\*10);

//.... Insert the Following:

if(communication[0] == 'A'){

}

if(communication[0] == 'D'){

}

if(communication[0] == 'G'){

}

if(communication[0] == 'J'){

}

if(communication[0] == 'M'){

}

if(communication[0] == 'N'){

}

}

//..

}



#### Step 4: Parse out Component ID

Once we know the component ID, we need to parse out the first character so that the only value remaining in the string is raw data.

// add this line to every if-statement in the case statement block

communication.remove(0,1);

#### 

#### Step 5: Use data from Protocontrol in your system

|  | Component ID | Expected Outcome |
| --- | --- | --- |
| Slider 1 (Red) | A | Data from the slider is between 0 to 255 (range configured on the website). Set Red variable equal to this. |
| Slider 2 (Green) | D | Data from the slider is between 0 to 255 (range configured on the website). Set Green variable equal to this. |
| Slider 3 (Blue) | G | Data from the slider is between 0 to 255 (range configured on the website). Set Blue variable equal to this. |
| Terminal | J | Display the photosensitive resistor value on the console |
| Rotary Encoder | M | Rotary encoder values are not restricted. Use modulus operation to scale the keypad values to 0-180 and set the Servo Motor’s position to this. |
| Keypad | N | Keypad data is between 0-9. Multiply the data by 10 and set the Servo Motor’s position to this. |

void loop () {

//...

if(Serial.available()){

communication = Serial.readStringUntil('\n');

Serial.println(communication);

//arm.write(communication.toInt()\*10);

if(communication[0] == 'A'){

communication.remove(0,1);

red = communication.toInt();

}

if(communication[0] == 'D'){

communication.remove(0,1);

green = communication.toInt();

}

if(communication[0] == 'G'){

communication.remove(0,1);

blue = communication.toInt();

}

if(communication[0] == 'M'){

communication.remove(0,1);

arm.write(communication.toInt()\*10);

}

if(communication[0] == 'N'){

communication.remove(0,1);

arm.write(communication.toInt());

}

if(communication[0] == 'J'){

communication.remove(0,1);

digitalWrite(4,communication.toInt());

}

}

//...

}



#### Step 6: Final Script

#include <Servo.h>

int sensorPin = A0; // select the input pin for the potentiometer

int ledPin = 13; // select the pin for the LED

int sensorValue = 0; // variable to store the value coming from the sensor

int lastValue;

Servo arm;

String communication;

//RGB led init variables

int redPin= 5;

int greenPin = 6;

int bluePin = 3;

int red = 0;

int green = 0;

int blue = 0;

//function to change color of LED based on input vars

void setColor(int redValue, int greenValue, int blueValue) {

analogWrite(redPin, redValue);

analogWrite(greenPin, greenValue);

analogWrite(bluePin, blueValue);

}

void setup() {

// declare the ledPin as an OUTPUT:

Serial.begin(115200);

pinMode(ledPin, OUTPUT);

digitalWrite(ledPin, LOW);

arm.attach(9);

//led setup

pinMode(redPin, OUTPUT);

pinMode(greenPin, OUTPUT);

pinMode(bluePin, OUTPUT);

pinMode(4,OUTPUT);

}

void loop() {

// set the color with local ints

setColor(red,green,blue);

//read the value from the light sensor

sensorValue = analogRead(sensorPin); //max 715 min 0

sensorValue = map(sensorValue,0,800,0,100);

//if its changed, call send data to update display

if(abs(lastValue - sensorValue) >= 4){

lastValue = sensorValue;

sendData(sensorValue);

delay(200);

}

if(Serial.available()){

communication = Serial.readStringUntil('\n');

Serial.println(communication);

//arm.write(communication.toInt()\*10);

if(communication[0] == 'A'){

communication.remove(0,1);

red = communication.toInt();

}

if(communication[0] == 'D'){

communication.remove(0,1);

green = communication.toInt();

}

if(communication[0] == 'G'){

communication.remove(0,1);

blue = communication.toInt();

}

if(communication[0] == 'M'){

communication.remove(0,1);

arm.write(communication.toInt()\*10);

}

if(communication[0] == 'N'){

communication.remove(0,1);

arm.write(communication.toInt());

}

if(communication[0] == 'J'){

communication.remove(0,1);

digitalWrite(4,communication.toInt());

}

}

}

void sendData(int data){

Serial.println("+Light Level: "+(String)data);

//Serial.println(data);

}

## Section VII: Terminology

Panel: refers to the main board of ProtoControl (see Figure 3)

Profile: refers to a unique layout of components on the website that gets uploaded to the main ProtoControl Panel/Touchscreen

Sketch: Term used frequently in reference to Arduino to describe a short script

## Appendix

### Appendix A: Arduino Background

#### Download Arduino IDE 2 for your device

Click [here](https://www.arduino.cc/en/software) to download the latest version of Arduino IDE for your machine.

1. General Structure of Arduino Sketch

// The first section is used for Constants and Defines

int sensorPin = A0;

// Setup (this section of code runs once at the beginning, especially to setup Serial Applications and pinMode (whether a pin is an input or output)

void setup() {

}

// Loop (this section of code continuously runs, like main() in C/C++)

void loop () {

// The analog Sensor is begin continuously read

sensorValue = analogRead(sensorPin); //max 715 min 0

}

// User Defined Functions

// Writing functions in Arduino is very similar to writing funcitons in C

// Function can be called in other functions or in void loop();

## 

## 

## References

[1] Analog Devices. "UART – A Hardware Communication Protocol." *Analog Dialogue*. [Online]. Available: <https://www.analog.com/en/resources/analog-dialogue/articles/uart-a-hardware-communication-protocol.html>. [Accessed: April 18, 2024].

[2] Circuit Basics. "Basics of UART Communication." *Circuit Basics*. [Online]. Available:<https://www.circuitbasics.com/basics-uart-communication/>. [Accessed: April 18, 2024].