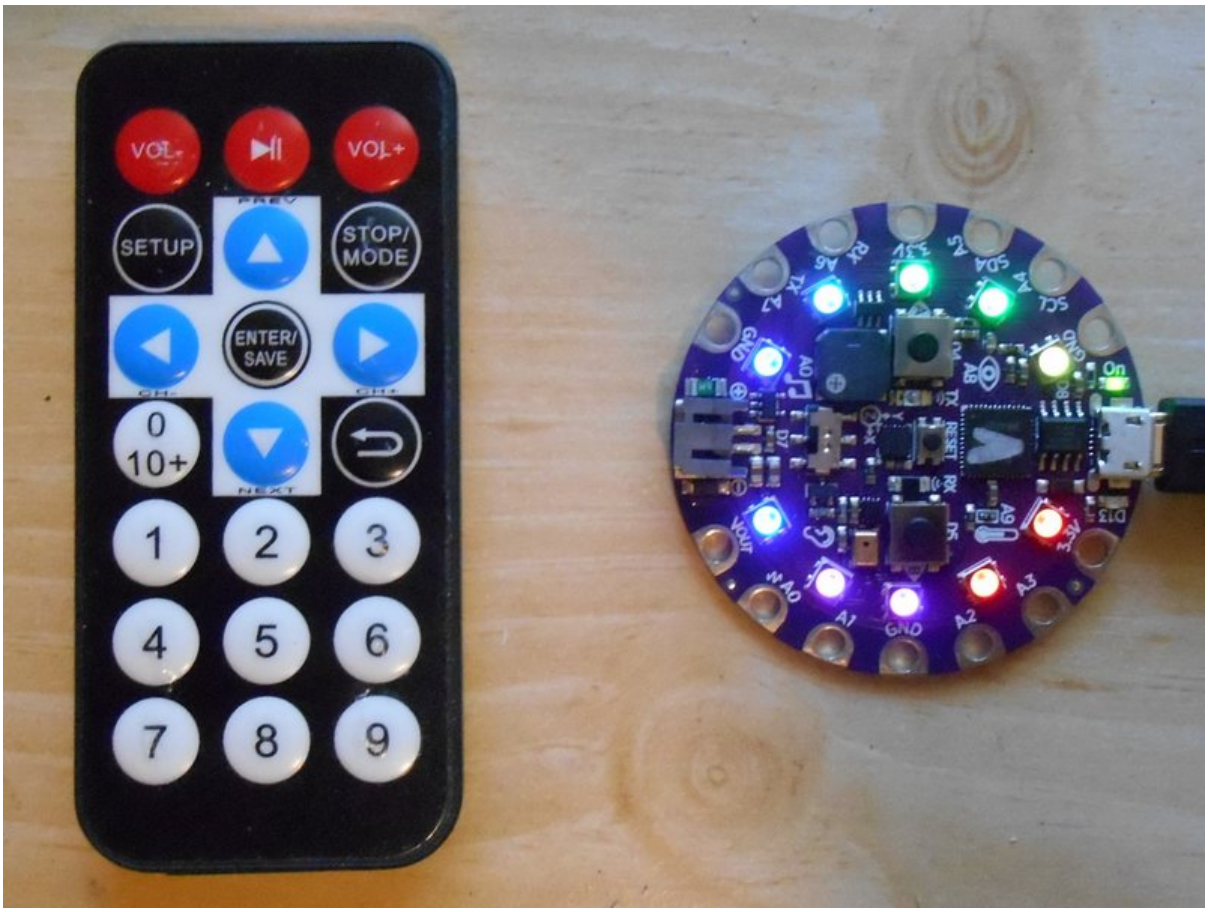
Infrared Transmit and Receive on Circuit Playground Express in C++

Created by Chris Young



https://learn.adafruit.com/infrared-transmit-and-receive-on-circuit-playground-express-in-c-plus-plus-2

Last updated on 2024-06-03 02:09:23 PM EDT

Table of Contents

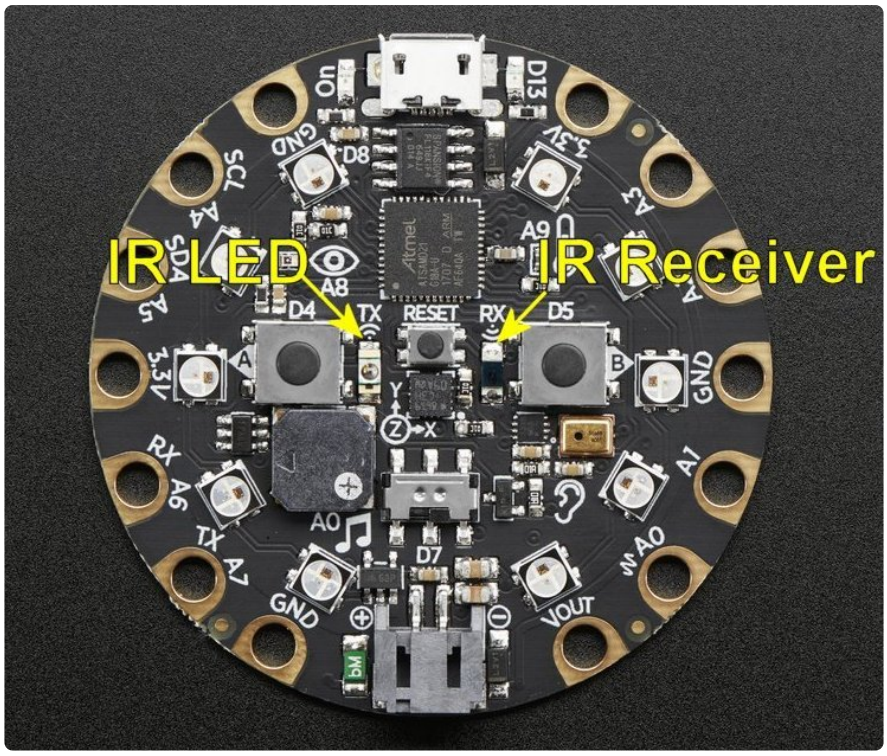
Overview

One of the great new features of the Circuit Playground Express is the addition of an infrared LED and infrared receiver. This allows you to use your Circuit Playground Express as an infrared remote control or to send signals to the Circuit Playground Express to make it perform various functions such as changing the pattern or colors of the NeoPixels.

The **Adafruit_CircuitPlayground** library allows you to control various functions on the Circuit Playground Express and in the latest version it has incorporated a specialized version of IRLib2. IRLib2 is a powerful library that allows you to transmit, receive, and decode IR signals using an Arduino or compatible microcontroller. It supports a large variety of protocols. These protocols are used by different manufacturers of consumer electronics products such as TVs, DVD/Blu-ray, and cable boxes. The specialized version of IRLib2 is incorporated into the **Adafruit_CircuitPlayground** library has been preconfigured for use with the Circuit Playground Express.

We have another Tutorial here in the learning system (https://adafru.it/xCP) that teaches you how to use IRLib2 on other Arduino platforms. If you are interested in more details you can get IRLib2 directly on GitHub (https://adafru.it/vwF). That repository includes an extensive users manual (https://adafru.it/xCR) with lots of details about the supported protocols and extensive information about how IR signals are transmitted and received.

In this tutorial we will give you a brief introduction to IR signals and show you how to use the IR features specifically of the Circuit Playground Express.

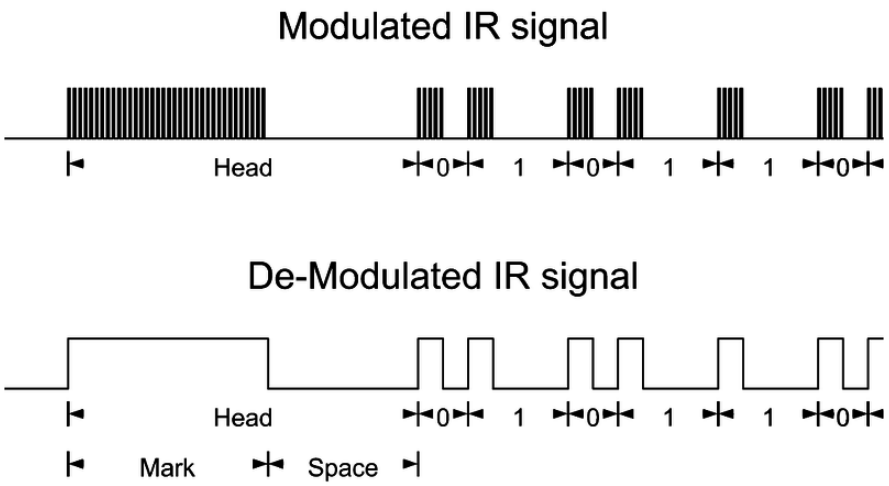


Understanding Infrared Signals

Remote controls such as those used for your TV, DVD/Blu-ray or Cable box use infrared transmitters and receivers. The transmitter is basically nothing more than flashlight that emits a light beam at a frequency that is lower than the human eye can detect. The remote control sends out a specially timed sequence of pulses to transmit data to your TV or other device. Each consumer electronics device manufacturer has its own set of protocols for these signals so that the signals intended for your TV for example do not interfere with those intended for your Blu-ray or cable box. Even within a single manufacturer there are special codes that make sure you only control the device you want to control.

When the IR LED is on for a brief period of time we call that a "Mark" and the period of time when it is off we call a "Space". A complete sequence of Marks and Spaces is referred to as a "frame" of data. The timing of these marks the spaces encodes the data we want to transmit. However the marks themselves are actually a modulated signal. Typically this is that about 38 kHz however some protocols use frequencies ranging from 36 kHz to 57 kHz. Modulating the signal ensures that other sources of infrared emission such as sunlight, light bulbs, or just the normal heat coming off of objects is not interpreted as part of the signal. The receiver must demodulate the signal and produce a clean square wave so that we can time the length of the off and on pulses. Your Circuit Playground Express includes a circuit that will demodulate the 38 kHz signals and turn them into a clean square wave that we can sample to understand the data.

Here is an illustration of how these signals are actually transmitted and how they look after they have been demodulated.



The signal usually starts with a very long mark and space. This header information helps identify the protocol and it allows the automatic gain control circuit in the receiver to adjust to the level of signal it is receiving. Then comes the data bits. In this example all of the marks are the same length but the variation of the length of spaces tell you whether it is a logical "1" bit or a logical "0" bit. Some protocols vary the length of marks and use fixed length spaces. Others vary the length of both marks and spaces. And still others use the transition from low to high versus high to low to designate a "0" or "1" bit.

Fortunately for the end user like you, the IRLib2 library takes care of all these details for you. If you are using one of our supported protocols, you don't need to know anything about the timing of the signals or how to interpret the patterns. The software takes the stream of pulses, decodes it, tell you what protocol it is using, how many bits of data in the signal, and then gives you an unassigned integer value up to 32 bits long that uniquely identifies the signal that was sent. You can also use these 32-bit integers in the sending portion of the software and it will re-create the string of marks and spaces in the proper protocol.

If you encounter a protocol that is not supported by IRLib2 there is an extensive section in the users manual that tells you how to analyze received data so that you can write your own decoder and encoder to support a new protocol. IRLib2 currently supports 12 protocols and many of those have variations within them. We are constantly adding support for new protocols and they will be incorporated into the library once they are tested and stable.

In the next section we will show you how to install the software and receive and decode an IR signal on your Circuit Playground Express.

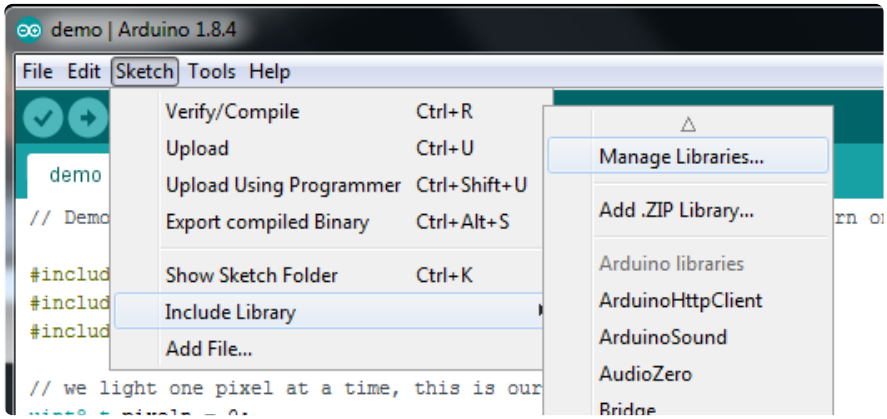
Library Installation

The IRLib2 library has been incorporated into the Adafruit_CircuitPlayground library so if you've already installed that library you can skip to the next page which describes the tutorial programs. If you have not already installed library here is how to do it.

Installing Via Library Manager

The Circuit Playground library is available on the Adafruit GitHub website (https://adafru.it/naF). The library is installed in versions of the Arduino IDE greater than 1.6.7 as follows:

Open up the Arduino library manager:



Search for **Adafruit circuit playground** library and install it. Make sure you put a space in-between circuit and playground as it will not show up otherwise.

Run the Demo

Plug in your Circuit Playground. Under Windows you'll need to make sure you have installed the driver package (https://adafru.it/naC). Do that if you haven't!

Next, you'll also need to make sure you have installed the Adafruit Board Support package. That's required no matter what computer you're using.

Select the Circuit Playground Express Board

Under the **Tools -> Board** submenu, scroll down to **Adafruit Boards** and pick **Adafruit Circuit Playground Express**

Select the matching Port

Under **Tools->Port** select the port that is labed (Circuit Plaground)

Load the Demo Program

OK you're now ready to load the demo. Under **File->Examples** locate **Adafruit CircuitPlayground** and then select the **demo** program.

Compile/Verify the Demo

Click the **Verify** button (also the **Sketch->Verify** menu item) to compile/verify the demo. Make sure you get "**Done compiling."** and no errors

Upload Demo

Click the Upload button to upload the code

You should get a **Done uploading.** message in the blue statusbar

You can now run the serial console to get data output:

You'll get information such as:

- "Capacitive touch" readings for all 8 outer pads (under 50 means not touched, over 100 usually means the pads are touched)
- Slide switch location (left or right)
- If the Right and Left buttons are pressed
- Light sensor readings, higher values mean more light
- Sound sensor readings
- X, Y and Z accelerometer readings
- Temperature in Celcius

Sample Programs

There are four tutorial programs included with the Adafruit_CircuitPlayground library. You can find them in the "examples" folder in that library. The programs are called:

- **Infrared_Read** -- reads the signal from infrared remote and attempts to decode it.
- **Infrared_NeoPixel** -- control the pattern and color of the NeoPixels on your Circuit Playground Express using a remote control.
- **Infrared_Send** -- sends an IR signal when you push the user buttons
- **Infrared_Record** -- reads an IR signal and remembers it. Then when you push a button on the Circuit Playground Express it will retransmit that signal.
- **Infrared_Testpattern** -- send a test signal to another Circuit Playground Express or Arduino to test all supported protocols.

To use these programs you will need an infrared remote. Some of the examples were written using the Adafruit Mini Remote shown below but you can use any IR remote by substituting the proper codes. We will show you how to do that.

Mini Remote Control
This little remote control would be handy for controlling a robot or other project from across the room. It has 21 buttons and a layout we thought was handy: directional buttons and...
https://www.adafruit.com/product/389

Infrared_Read

In this example you will point an infrared remote at your Circuit Playground Express and we will decode the signal. It will tell you the name and number of the protocol, the number of bits in the protocol, and a hex number up to 32 bits long that represents the received data.

You can find the program in the Adafruit_CircuitPlayground library in the "examples" subfolder. Or here is a listing of the program that you can cut and paste and we will use for reference in our discussion.

```
/* Infraread_Read.ino Example sketch for IRLib2 and Circuit Playground Express
   Illustrates how to receive an IR signal, decode it and print
   information about it to the serial monitor.
*/
#include &lt;Adafruit_CircuitPlayground.h&gt;

#if !defined(ADAFRUIT_CIRCUITPLAYGROUND_M0)
  #error "Infrared support is only for the Circuit Playground Express, it doesn't
work with the Classic version"
#endif


void setup() {
  CircuitPlayground.begin();
  Serial.begin(9600);

  while (!Serial); // Wait until serial console is opened

  CircuitPlayground.irReceiver.enableIRIn(); // Start the receiver
  Serial.println("Ready to receive IR signals");
}

void loop() {
  //Continue looping until you get a complete signal received
  if (CircuitPlayground.irReceiver.getResults()) {
    CircuitPlayground.irDecoder.decode();           //Decode it
    CircuitPlayground.irDecoder.dumpResults(false);  //Now print results. Use false
for less detail
    CircuitPlayground.irReceiver.enableIRIn();      //Restart receiver
  }
}
```

You should upload the sketch to your Circuit Playground Express using the Arduino IDE. Once it has successfully uploaded, you should open your serial monitor. It will prompt you to send an IR signal. Take any TV or DVD or cable box remote or the recommended Adafruit Mini Remote and point it at your Circuit Playground Express and press a button.

Here is some sample output.

```
Ready to receive IR signals
Decoded NEC(1): Value:FD807F Adrs:0 (32 bits)
```

This was a result after pressing the play button on the Adafruit Mini Remote. It tells us that we successfully decoded NEC protocol. That is protocol number 1. The hex value returned was "0xFD807F". This is a 32-bit value. It so happens that all NEC protocols are always 32 bit values but some protocols such as Sony might have a variable number of bits of data so that will be useful information. The "Address" value is also used by some protocols to provide additional information. For example the Samsung36 protocol is 32 bits long and is separated into a 16-bit address and a 20 bit value. For the NEC protocol "Address" is and used.

If the protocol is not recognized you will see...

```
Decoded Unknown(0): Value:0 Adrs:0 (0 bits)
```

In the next example we will show you how to use the received data from the IR receiver to control the pattern of lights on the NeoPixels.

How it works

Let's look at the specifics of the program to see how it works.

At the top we include the Adafruit_CircuitPlayground library. Then in the setup function we initialize the CircuitPlayground object and the serial monitor. We wait for the serial monitor to initialize. Then we call...

```
CircuitPlayground.irReceiver.enableIRIn()
```

This tells the IR receiver portion of IRLib2 to begin monitoring the Circuit Playground Express IR receiver circuitry using a pin change interrupt. Every time it detects an IR signal going from low to high or high to low it generates interrupt and starts a timer. It measures the intervals of Marks and Spaces received.

In the "loop()" portion of the program we have an if statement that checks...

```
if (CircuitPlayground.irReceiver.getResults()) {
```

this function checks to see if a full frame of our data has been received and if it has it will return true. In this example we do nothing but wait for that signal to turn true but in other applications you can go off and do something else in the loop as long as you periodically check "`getResults()`" to determine when the data has been received and then to do something with it.

Once a full frame of data has been received, the receiver stops listening for more data until you tell it to start up again.

Inside the if statement we call the `CircuitPlayground.irDecoder.decode()` method. It looks at the timing data and determines what protocol was used and decodes it. To see the results we call the `CircuitPlayground.irDecoder.dumpResults(false)` method. It will print the information on the serial monitor. If you change the parameter to "true" it will give you a more verbose listing of the information received.

Finally now that we have decoded the information and printed it out, we want to resume receiving so we have to call `CircuitPlayground.irReceiver.enableIRIn()` again to restart the receiver.

Details of the signal

For most applications, once you know the protocol number, the number of bits, and the data value that is all you need to to know in order to do whatever you want to do with the information. But if you're trying to analyze an unknown protocol or are just curious about what goes on behind the scenes we can use the verbose option on the dumpResults method to get more information. Edit the line in your sketch that says

```
CircuitPlayground.irDecoder.dumpResults(false)
```

and change the parameter to "true". Here is the output you would see by pressing the play button on the Adafruit Mini Remote.

```
Ready to receive IR signals
Decoded NEC(1): Value:FD807F Adrs:0 (32 bits)
Raw samples(68): Gap:52026
Head: m9014 s4560
0:m538 s602 1:m567 s574 2:m565 s577 3:m563 s577
4:m533 s607 5:m542 s597 6:m543 s598 7:m541 s599
8:m560 s1711 9:m539 s1734 10:m564 s1706 11:m564 s1707
12:m562 s1710 13:m569 s1702 14:m567 s573 15:m566 s1704

16:m536 s1737 17:m562 s578 18:m542 s598 19:m561 s580
20:m559 s581 21:m559 s581 22:m539 s602 23:m538 s602
24:m537 s604 25:m536 s1735 26:m534 s1737 27:m562 s1709
28:m541 s1731 29:m568 s1702 30:m567 s1704 31:m536 s1736

32:m534
Extent=67563
Mark min:533 max:569
Space min:573 max:1737
```

The first line of the output is identical to the non-verbose version of the output. It tells us this is the NEC protocol, the hex value, and the number of bits. It then goes on to give us other information.

There were 68 samples of marks and spaces. The gap between the time we called enableIRIn() and the beginning of the signal was 52026 microseconds. All of the values in this dump are given in microseconds.

The header of the signal consisted of a mark that was 9014 and space of 4500. This is followed by the mark and space values for each of the 32 bits. The marks are pretty much consistent around the middle 500 range. Some of the spaces are about that same value while others are are three times that value. The bits with short spaces are interpreted as a "0" and the bits with a long space are interpreted as a "1".

Other information at the end of the dump includes "Extent" which is the total length of the signal. It also tell you the minimum and maximum length of marks and spaces. This information can be used to identify unsupported protocols.

For more information about all the protocols supported by IRLib2 and how to write your own protocol decoders for protocols we do not yet support there is extensive information in the IRLib2 manual available on GitHub at https://github.com/cyborg5/IRLib2/tree/master/IRLib2/manuals (https://adafru.it/xCR)

The manual is available in Microsoft Word .docx, Adobe acrobat PDF, or e-book .epub formats.

Infrared_NeoPixel

In this section we will show you how to control a pattern of lights on the Circuit Playground Express ring of NeoPixels using an IR remote. You can find the source code in the Adafruit_CircuitPlayground library in the "examples" subfolder. We will not reproduce the entire source listing here but we will make some references to particular portions.

This example has been programmed to use the Adafruit Mini Remote Control which uses NEC protocol. If you do not have that remote available you will have to modify the sketch to use the protocol and codes for your particular remote. We will give you some tips on how to do that later but for now let's look at how the program works as originally written.

You should upload the scanned to your Circuit Playground Express using the Arduino IDE. When the upload has completed the NeoPixels will display a red dot that is moving counterclockwise. This is the default pattern we have programmed.

Try pushing the right arrow button on the remote and the dot will begin to spin clockwise. Pressing the left arrow will make it spin counterclockwise again. If you push the up arrow or down arrow it will go faster or slower. If you press the volume plus or volume minus the brightness will go up or down. If you press the "0" button the pattern will shift to a rotating rainbow pattern. Pressing "1" gives you a red and white "candy cane" pattern. Pressing "2" gives uses a default one red pixel spinning. Pressing "3" or "4" will display a solid color on all pixels and will cycle up and down the rainbow on repeated presses.

How It Works

In the "setup" function which begins at line 65 we initialize the Circuit Playground and enable the IR receiver as well as initialize various variables that control the pattern of the NeoPixels. In the loop function beginning at line 72 we first display the pattern using "Show_Pattern()" followed by a brief delay and then we increment the phase of the pattern wrapping around at 10. Now we check to see if there is a new IR signal. If there is not we return and the loop continues. We then attempt to decode the signal. If that fails we return and the loop executes again. Note that we could get a perfectly valid IR signal but if we can't decode it then we just ignore it, reinitialize the receiver and go on.

If it was decoded, we print out the results on the serial monitor just so we can see what's going on for the fun of it. Next we check to see if this really was NEC protocol. If that succeeds we then go into a "switch" statement online number 107. The switch is based on the decoded value. Depending on what value was received we change the pattern type, direction of spinning, brightness, or speed. Once we made those changes we reinitialize the receiver. On the next execution of the loop when we get "Show_Pattern()" the changes we have made will take effect.

Using a Different Remote

If you are not using the Adafruit Mini Remote you have to modify this program to use the codes of your particular remote. There are 10 functions that you have to support so you will need to pick 10 different buttons on your remote to assign to the various functions. We used the volume up and down, the 4 arrow keys, and the numbers 0-4 but you can pick any buttons that you want on your remote.

You need to load the Infrared_Read sketch from the previous page of this tutorial. Point your remote at the Circuit Playground Express and push one of the buttons that you want to assign to one of the functions of this program. Note the protocol name

and number and the hex value that is decoded. Write down that information for each of the 10 buttons you're going to use.

At line 100 of the program we check to see if we received and decoded the proper protocol number. The statement currently says...

```
if (! CircuitPlayground.irDecoder.protocolNum == NEC) {
```

If your remote uses something other than NEC protocol you will have to substitute the protocol number for your remote. Here is a list of the supported protocols. This list is defined in "Adafruit_CircuitPlayground/utility/IRLibProtocols.h"

```
#define UNKNOWN 0
#define NEC 1
#define SONY 2
#define RC5 3
#define RC6 4
#define PANASONIC_OLD 5
#define JVC 6
#define NECX 7
#define SAMSUNG36 8
#define GICABLE 9
#define DIRECTV 10
#define RCMM 11
#define CYKM 12
```

Look at the "switch" statement at line 107. There are 10 cases corresponding to the 10 different functions in this program. You will see values such as "ADAF_MINI_1" or "ADAF_MINI_2" on each case statement. These values are defined in a file called "adafruit_mini_codes.h" that is included with the program. At the top of your Arduino IDE you can click on the second tab to see this file. It contains the hex codes for all of the buttons on that remote... even the ones we are not using on this project. If you plan to use a remote for more than one project it might be a good idea to create such a file for your particular remote and then just copy that information into whatever program you need.

For example if I was going to use the remote for my Samsung TV it uses NECx protocol (not straight NEC but NECx which is protocol number 7) The code for volume up is "0xE0E0F01F". So I have a choice of either creating a header file with the definition such as

```
#define MY_TV_VOLUME_UP 0xE0E0F01F
```

And then I would modify the case statement online 144 which currently reads

```
case ADAF_MINI_VOLUME_UP:
```

to instead read

`case MY_TV_VOLUME_UP:`

or I could just plug in the value directly by saying...

`case 0xE0E0F01F:`

You will need to make the modifications for all 10 case statements.

Some protocols use special repeat codes and other strange features that may affect the way you implement your code. For complete details about all the protocols supported by IRLib2 and how to write your own protocol decoders for protocols we do not yet support there is extensive information in the IRLib2 manual available on GitHub at https://github.com/cyborg5/IRLib2/tree/master/IRLib2/manuals (https://adafru.it/xCR)

The manual is available in Microsoft Word .docx, Adobe acrobat PDF, or e-book .epub formats.

Infrared_Send

So far we have only used the receiver portion of the IR capabilities of the Circuit Playground Express. But we also have the ability to transmit IR codes using the built-in IR LED. Here is an extremely simple example that sends a signal whenever you push one of the buttons on the Circuit Playground Express. This example is set up for my Samsung TV using NECx protocol but you can substitute protocol and code values for your remote. Use the Infrared_Read example to figure out the codes for your TV remote.

```
/* Infrared_Send.ino Example sketch for IRLib2 and Circuit Playground Express
   Illustrates how to transmit an IR signal whenever you do push one of the
   built-in pushbuttons.
*/
#include &lt;Adafruit_CircuitPlayground.h&gt;

#if !defined(ADAFRUIT_CIRCUITPLAYGROUND_M0)
  #error "Infrared support is only for the Circuit Playground Express, it doesn't
work with the Classic version"
#endif

void setup() {
  CircuitPlayground.begin();
}

//Defines for a Samsung TV using NECx protocol
#define MY_PROTOCOL NECX
#define MY_BITS 32
```

```
#define MY_MUTE 0xE0E0F00F
#define MY_POWER 0xE0E040BF

void loop() {
  // If the left button is pressed send a mute code.
  if (CircuitPlayground.leftButton()) {
    CircuitPlayground.irSend.send(MY_PROTOCOL,MY_MUTE,MY_BITS);
    while (CircuitPlayground.leftButton()) {}//wait until button released
  }
  // If the right button is pressed send a power code.
  if (CircuitPlayground.rightButton()) {
    CircuitPlayground.irSend.send(MY_PROTOCOL,MY_POWER,MY_BITS);
    while (CircuitPlayground.rightButton()) {}//wait until button released
  }
}
```

Upload the sketch using the Arduino IDE. Point the front of your Circuit Playground Express at your TV. When you press the left button it should mute your TV. When you press the right button it will toggle the power. Again we remind you you may have to substitute different protocols and codes for your TV.

Note that after we send the data, we wait until the button is released so that it doesn't repeatedly send the code over and over again.

You could expand this example to add other functions perhaps using the capacitive touch features of the Circuit Playground Express.

Infrared_Record

In this example we will both receive and send IR signals in the same program. You will point your remote at the Circuit Playground Express and press a button. It will record the signal and then when you press the left button on the Circuit Playground Express it will re-transmit it. You can find example program in Adafruit_CircuitPlayground/examples/Infrared_Record. Here is the listing.

```
/* Infrared_Record.ino Example sketch for IRLib2 and Circuit Playground Express
   Illustrates how to receive an IR signal, decode and save it.
   Then retransmit it whenever you push the left pushbutton.
*/
#include &lt;Adafruit_CircuitPlayground.h&gt;

#if !defined(ADAFRUIT_CIRCUITPLAYGROUND_M0)
  #error "Infrared support is only for the Circuit Playground Express, it doesn't
work with the Classic version"
#endif


/* IR signals consist of a protocol number, a value, and a number of bits.
 *  Store all of these values for future use.
 */
uint8_t IR_protocol;
uint32_t IR_value;
uint16_t IR_bits;

void setup() {
```

```
  Serial.begin(9600);
  while (!Serial);
  Serial.println("Send an IR signal and I will record it.");
  Serial.println("Press the left button and we will retransmit it.");
  CircuitPlayground.begin();

  CircuitPlayground.irReceiver.enableIRIn(); // Start the receiver
  IR_protocol=0; //  Indicates we've not received a code yet
}

void loop() {
  /*  Receiver will look for a signal and when wa complete frame of data
   *   has been received, getResults() returns true. Once that happens,
   *   the receiver stops reccording  so you will need to restart it
   *   after you have processed the data.
   */
  if(CircuitPlayground.irReceiver.getResults()) {
    //attempt to decode it
    if(CircuitPlayground.irDecoder.decode()) {
        Serial.println("IR decoded");
        //Print the results.  Change parameter to "true" for verbose output.
        CircuitPlayground.irDecoder.dumpResults(false);
        Serial.println("Saving results. Press left button to retransmit.");
        IR_protocol=CircuitPlayground.irDecoder.protocolNum;
        IR_value= CircuitPlayground.irDecoder.value;
        IR_bits= CircuitPlayground.irDecoder.bits;
    }
    CircuitPlayground.irReceiver.enableIRIn();      //Restart receiver
  }

  /* If the left button is pressed and we have received a code
   * retransmit it using the sender.
   */
  if (CircuitPlayground.leftButton()) {
    Serial.println("Left button pressed!");
    if(IR_protocol) {
      CircuitPlayground.irSend.send(IR_protocol, IR_value, IR_bits);
      Serial.println("Sending recorded IR signal");
      Serial.print("Protocol:"); Serial.print(IR_protocol,DEC);
      Serial.print("  Value:0x");  Serial.print(IR_value,HEX);
      Serial.print("  Bits:"); Serial.println(IR_bits,DEC);
    } else {
      Serial.println("No signal saved yet.");
    }
  }
}
```

The program loop looks for an IR signal and when it finds one that it can decode it records the protocol, data value, and number of bits. Then if you push the left button it checks to see if a code has been received and if it has it uses the send method to retransmit it.

Infrared_Testpattern

This particular program is admittedly not a much practical use. It allows you to transmit a sample signal using any of the 12 supported protocols and their various variants. We use it to debug the software. It illustrates how each of the 12 protocols works. Additional details on the protocols can be found in the IRLib2 users manual

available on GitHub at https://github.com/cyborg5/IRLib2/tree/master/IRLib2/manuals (https://adafru.it/xCR)

You can find this program in the "Adafruit_CircuitPlayground/examples" folder. We will not bother to reproduce a listing here. You should upload the program using the Arduino IDE and open the serial monitor. You will be prompted to type a number from 1 to 12 to send out test signals for any of the 12 supported protocols. Or you can enter "-1" and it will cycle through all 12 protocols. Note that the signals we are sending are not necessarily valid bit patterns for any particular function in that protocol. Rather they have been chosen to illustrate some of the features of those protocols.

You need another Arduino set up to receive IR signals using IRLib2 running the "dump" example or Circuit Playground Express using the Infrared_Read program. It will receive the signals transmitted by this program and you can verify that the transmission is working properly. Here is the output from this program.

```
Type the protocol number: 1-12 Or '-1' for all
Protocol #1 NEC data = 0x12345678
Protocol #1 NEC data = 0xFFFFFFFF
Protocol #1 NEC data = 0x12345678 data2 = 0x28 40 dec
Protocol #2 Sony data = 0x12345678 data2 = 0xC 12 dec
Protocol #2 Sony data = 0x12345678 data2 = 0xF 15 dec
Protocol #2 Sony data = 0x12345678 data2 = 0x14 20 dec
Protocol #3 RC5 data = 0xFFFFFFFF data2 = 0xD 13 dec
Protocol #3 RC5 data = 0xFFFFFFFF data2 = 0xE 14 dec
Protocol #3 RC5 data = 0xFFFFFFFF data2 = 0xE 14 dec khz=57
Protocol #4 RC6 data = 0xFFFF data2 = 0x14 20 dec
Protocol #4 RC6 data = 0xCFFFFF data2 = 0x18 24 dec
Protocol #4 RC6 data = 0xCFFFFF data2 = 0x1C 28 dec
Protocol #4 RC6 data = 0xFFFFFFFF data2 = 0x20 32 dec
Protocol #5 Panasonic Old data = 0xFFFFFFFF
Protocol #6 JVC data = 0x12345678 data2 = 0x1 1 dec
Protocol #6 JVC data = 0x12345678
Protocol #7 NECx data = 0x12345678
Protocol #7 NECx data = 0xFFFFFFFF
Protocol #8 Samsung36 data = 0x12345 data2 = 0x1234 4660 dec
Protocol #9 G.I.Cable data = 0x12345678
Protocol #9 G.I.Cable data = 0xFFFFFFFF
Protocol #10 DirecTV data = 0x12345678
Protocol #10 DirecTV data = 0x12345678 data2 = 0x1 1 dec
Protocol #10 DirecTV data = 0x12345678 khz=40
Protocol #11 rcmm data = 0xFFFFFFFF data2 = 0xC 12 dec
```

```
Protocol #11 rcmm data = 0xFFFFFFFF data2 = 0x18 24 dec
Protocol #11 rcmm data = 0xFFFFFFFF data2 = 0x20 32 dec
Protocol #12 CYKM data = 0x2001
Protocol #12 CYKM data = 0x2004
Protocol #12 CYKM data = 0x2002
Protocol #12 CYKM data = 0x2008
```

Here is the output you would see on another Arduino receiving these signals.

```
Ready to receive IR signals
Decoded NEC(1): Value:12345678 Adrs:0 (32 bits)
Decoded NEC(1): Value:FFFFFFFF Adrs:0 (0 bits)
Decoded NEC(1): Value:12345678 Adrs:0 (32 bits)
Decoded Sony(2): Value:678 Adrs:0 (12 bits)
Decoded Sony(2): Value:678 Adrs:0 (12 bits)
Decoded Sony(2): Value:678 Adrs:0 (12 bits)
Decoded Sony(2): Value:5678 Adrs:0 (15 bits)
Decoded Sony(2): Value:5678 Adrs:0 (15 bits)
Decoded Sony(2): Value:5678 Adrs:0 (15 bits)
Decoded Sony(2): Value:45678 Adrs:0 (20 bits)
Decoded Sony(2): Value:45678 Adrs:0 (20 bits)
Decoded Sony(2): Value:45678 Adrs:0 (20 bits)
Decoded RC5(3): Value:1FFF Adrs:0 (13 bits)
Decoded RC5(3): Value:3FFF Adrs:0 (14 bits)
Decoded RC5(3): Value:3FFF Adrs:0 (14 bits)
Decoded RC6(4): Value:FFFF Adrs:0 (20 bits)
Decoded RC6(4): Value:CFFFFF Adrs:0 (24 bits)
Decoded RC6(4): Value:CFFFFF Adrs:0 (28 bits)
Decoded RC6(4): Value:FFFFFFFF Adrs:0 (32 bits)
Decoded Panasonic Old(5): Value:3FFFFF Adrs:0 (22 bits)
Decoded JVC(6): Value:5678 Adrs:1 (16 bits)
Decoded JVC(6): Value:5678 Adrs:0 (16 bits)
Decoded JVC(6): Value:5678 Adrs:0 (16 bits)
Decoded NECx(7): Value:12345678 Adrs:0 (32 bits)
Decoded NECx(7): Value:FFFFFFFF Adrs:0 (0 bits)
Decoded Samsung36(8): Value:12345 Adrs:1234 (36 bits)
Decoded G.I.Cable(9): Value:5678 Adrs:0 (16 bits)
Decoded NEC(1): Value:FFFFFFFF Adrs:0 (0 bits)
Decoded DirecTV(10): Value:5678 Adrs:0 (16 bits)
Decoded DirecTV(10): Value:5678 Adrs:1 (16 bits)
Decoded DirecTV(10): Value:5678 Adrs:0 (16 bits)
Decoded rcmm(11): Value:FFF Adrs:0 (12 bits)
Decoded rcmm(11): Value:FFFFFF Adrs:0 (24 bits)
```

```
Decoded rcmm(11): Value:FFFFFFFF Adrs:0 (32 bits)
Gap=38622 CYKM decoded:2001 Mouse move right 0
Gap=37282 CYKM decoded:2004 Mouse move up 0
Gap=37442 CYKM decoded:2002 Mouse move left 0
Gap=37562 CYKM decoded:2008 Mouse move down 0
```

Customizing Supported Protocols

The original IRLib2 was designed to run on 8-bit AVR style microprocessors such as Arduino Uno, Arduino Leonardo, etc. These processors have very limited space and in most applications you only need to use one or two protocols and not all 12. IRLib2 as a clever system of using include files to write an application with only the protocols you need to use. Because the Circuit Playground Express has an ARM 32 bit processor with plenty of memory we have included all 12 protocols by default. But should you have an extensive application and find yourself running short on program memory, you can customize the IRLib2 portion of the Adafruit_CircuitPlayground library to use only a limited set of protocols.

Here is a partial listing of "Adafruit_CircuitProtocol/utility/IRLibCPE.h".

```
#include "IRLibDecodeBase.h"
#include "IRLibSendBase.h"
#include "IRLib_P01_NEC.h"
#include "IRLib_P02_Sony.h"
#include "IRLib_P03_RC5.h"
#include "IRLib_P04_RC6.h"
#include "IRLib_P05_Panasonic_Old.h"
#include "IRLib_P06_JVC.h"
#include "IRLib_P07_NECx.h"
#include "IRLib_P08_Samsung36.h"
#include "IRLib_P09_GICable.h"
#include "IRLib_P10_DirecTV.h"
#include "IRLib_P11_RCMM.h"
#include "IRLib_P12_CYKM.h"
//include additional protocols here
#include "IRLibCombo.h"
#include "IRLibRecvPCI.h"
```

This file includes the base code for sending and receiving followed by 12 include statements for each of the supported protocols. If you are not going to use one of these protocols you can edit the file to comment out any of the unused protocols. Do not rearrange the order that the protocols are included. They have to remain in numerical order but you can comment out as many as you want as long as at least one is left.

Furthermore if you're only receiving and not sending you can comment out the

```
#include "IRLibBase.h"
```

Similarly if you are only sending but not receiving and decoding you can comment out

```
#include "IRLibDecodeBase.h"
```

Again note that you probably have plenty of available program memory on this platform and you need not make these optimizations but we wanted to tell you how to do so if your application was running out of program memory.

Final Thoughts

These sample programs are just to illustrate capabilities of IR transmitting and receiving. There are lots of other applications you can come up with. As we mentioned you can use the capacitive touch capabilities to trigger the sending of other IR signals. You could use the IR receiver to do more than just blink the NeoPixels. Perhaps playback a sound sample using the onboard speaker. Use 2 Circuit Playground Express boards to talk to one another perhaps transmit secret codes or identify friend or foe. Kids will come up with lots of creative uses for these features. The possibilities are limited only by your imagination.