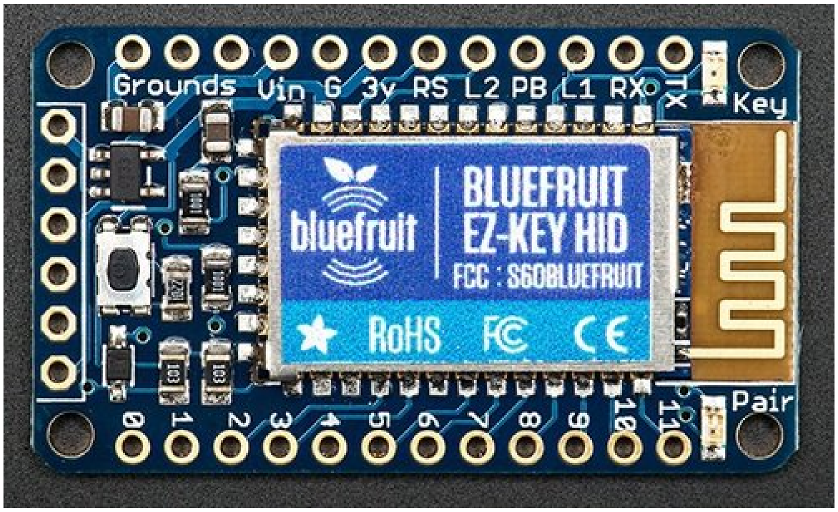
Introducing Bluefruit EZ-Key

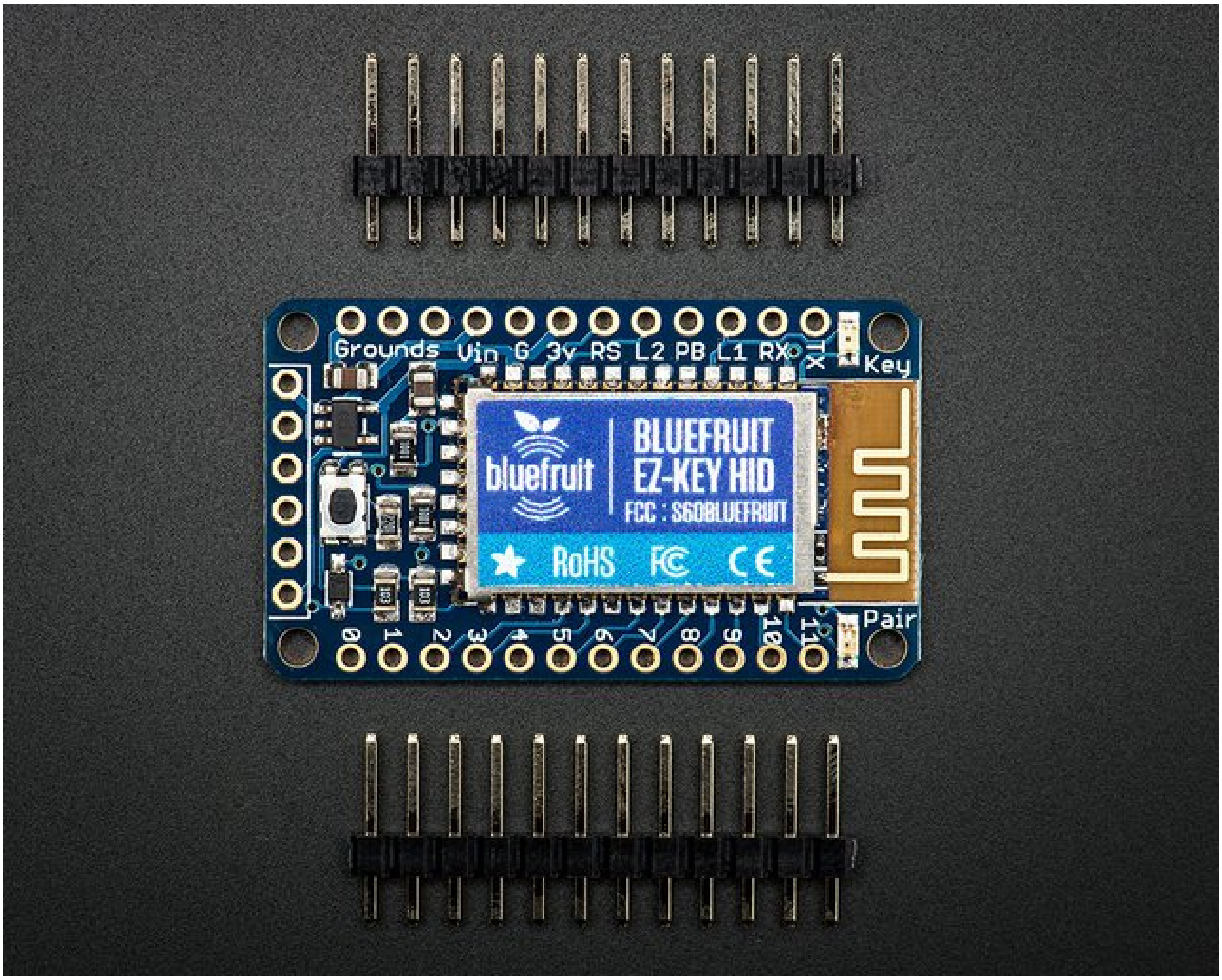
Created by lady ada



Last updated on 2016-09-22 06:30:31 PM UTC

Guide Contents

Overview



Create your own wireless Bluetooth keyboard controller in an hour with the Bluefruit EZ-Key: it's the fastest, easiest and bestest Bluetooth controller. We spent years learning how to develop our own custom Bluetooth firmware, and coupled with our own BT module hardware, we've created the most Maker-friendly wireless you can get!

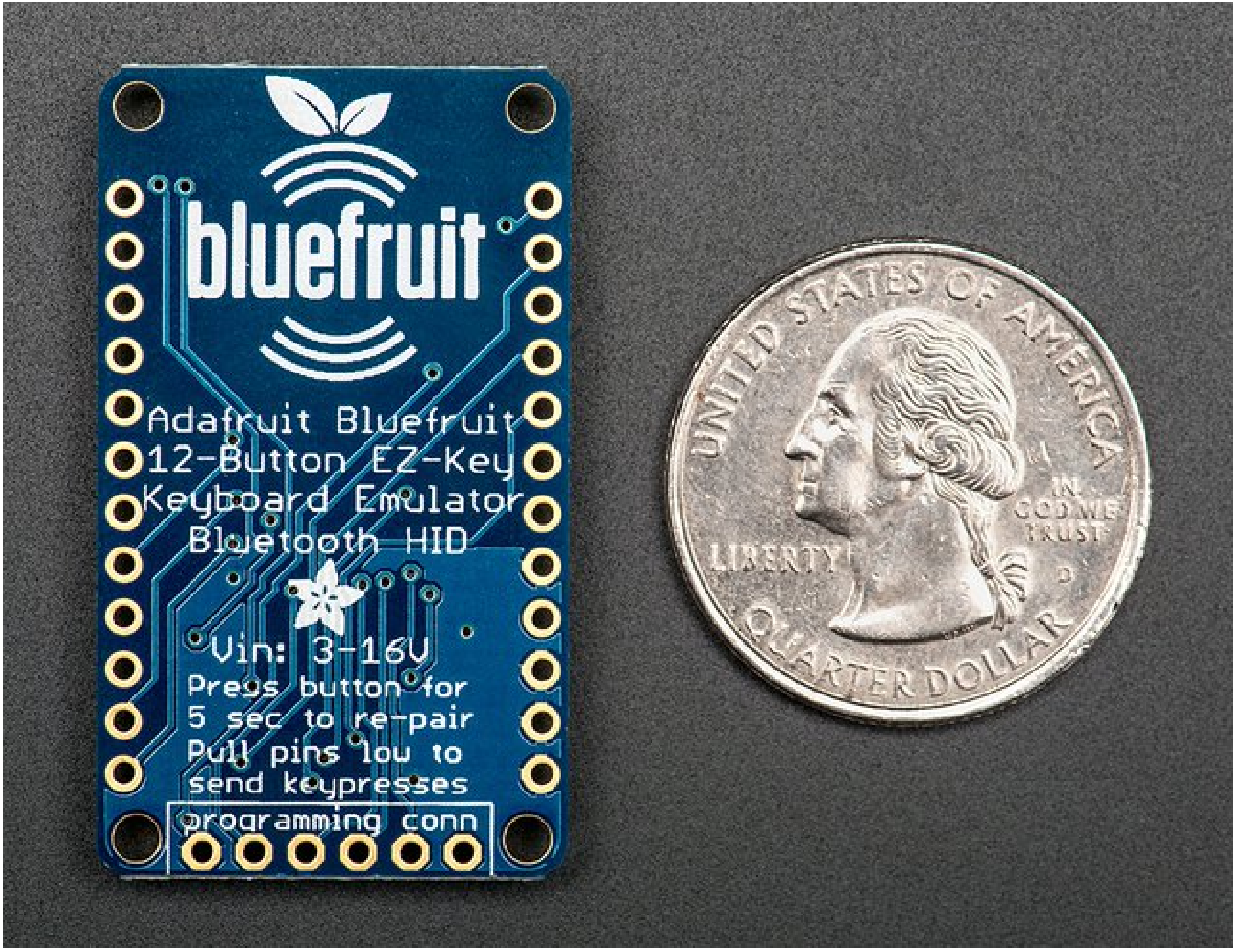
This breakout acts just like a BT keyboard, and works great with any BT-capable device: Mac, Windows, Linux, iOS, and Android. Power the module with 3-16VDC, and pair it to the computer, tablet or phone just as you would any other BT device. Now you can connect buttons from the 12 input pins, when a button is pressed, it sends a keypress to the computer. We pre-program the module to send the 4 arrow keys, return, space, 'w', 'a', 's', 'd', '1' and '2' by default. Advanced users can reprogram the module's keys using an FTDI or other Serial console cable, for any HID key report they desire.

You can pair multiple Bluefruit's to a single device, each one has a unique identifier. These modules are FCC & CE certified and are RoHS-compliant so they are easy to integrate into your project.

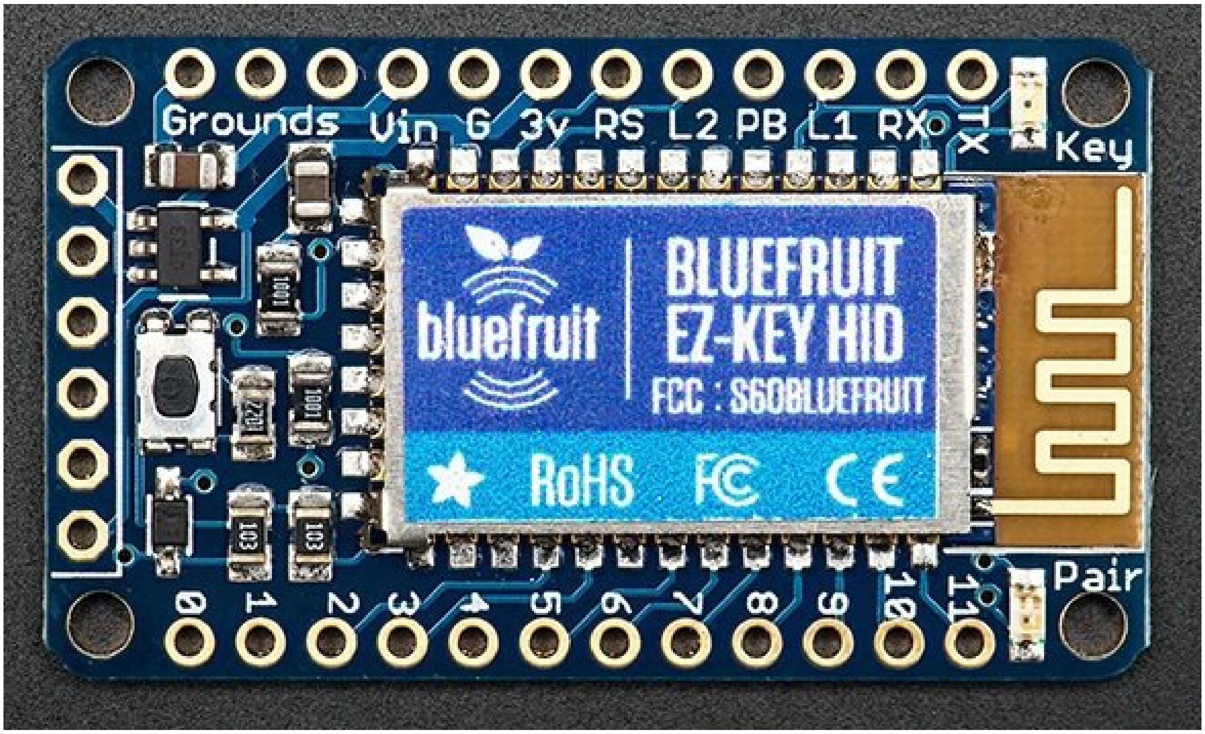
If you want to have better control over the data sent, connect a microcontroller to the RX pin at 3-5V logic level, 9600 baud, and send ASCII data: it will be 'typed out' character by character. We also have support for various non-printable characters such as ESC, Shift, F1-F12, etc. as well as toggling the virtual keyboard on iOS.

New in v1.1 (shipping as of Oct 22 2013) - We've made Bluefruit EZ-Key even better, you can now map keys to mouse button clicks and mouse movement (up/down/left/right) as well as send mouse commands over the UART. We also now have 'over the air' remapping, no serial cable required to re-map the pins!

New in v1.2 (shipping as of Nov 7 2013) - We have added support for mapping buttons to some "Consumer Report" keys, also known as Multimedia buttons.



Pinouts



This is a tour of all the pins available on the EZ-Key module. The module consists of a CSR BlueCore with custom firmware on a breakout PCB. The PCB makes it easy to use and hard to break. Every pin and connection you want is available on the breakout board, and there's even some handy mounting holes.

Let's take a tour of the pins! Starting with the top control & power pins...

Top Row

Grounds and Vin

To use this module, you will need to at least power it. Powering it is easy though, you can give it anywhere between 3-16VDC and the power input is reverse-polarity protected. Connect the positive wire from your battery to **Vin** and ground to the **Grounds** or **G** pin. There is an output from the onboard 3.3V voltage regulator on pin **3v** that will let you snag ~100mA of current for other sensors, microcontrollers or whatever.

Control and LEDs

To the right of the power pins, there are some control pins

- **RS** - this is the reset pin. To reset the module, pull this pin to ground. It does not affect pairing.
- **L2 -** this is the same output that is connected to the**Pair** LED. If you want to put this in a box and have an external Pair indicator LED, wire an LED from this pin, through a 1K resistor, to ground.
- **PB** - this is the pair button pin. It is connected to the button onboard that is used to reset the pairing. If you want to make another, external pair button, connect a switch from this key to **3V** (not ground!)
- **L1 -** this is the same output that is connected to the**Key** LED. If you want to put this in a box and have an external Key-press indicator LED, wire an LED from this pin, through a 1K resistor, to ground.
- **RX** - this is the UART input, used if you want to send UART->Keypress data, or re-map the buttons. It is 5V compliant, use 3V-5V TTL logic, 9600 baud.
- **TX** - this is the UART output, used for watching debug data or re-mapping the buttons. It is 3V logic level output.

Bottom Row

This row is easy, it is 12 individual pins that connect to a switch that will trigger a keypress. Each pin has a pullup resistor internally to 3V. To activate a keypress, connect the pin to ground. When it is connect to ground, a KEYDOWN is sent, when it is disconnected, a KEYUP is sent.

Do not inject 5V into these pins! They connect directly to the BT module which runs at 3V.

Left Port (6-pins)

Despite looking *a lot* like an FTDI connector, this is the programming/test port. We use this at the Adafruit factory to get your modules tested. It is *not* field reprogrammable. Do not connect anything to these pins, it could damage or permanently brick the EZ-Key!

Pairing to Bluefruit

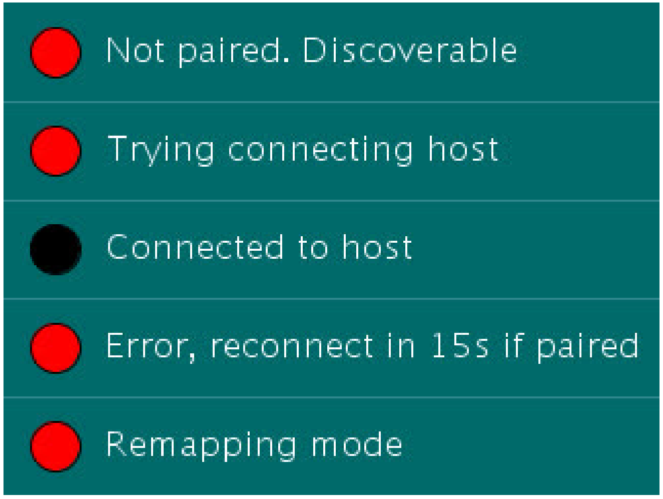
Before you can use the EZ-Key you have to **pair** it to your computer, laptop, tablet or phone. It's pretty easy to do this because EZ-Key acts just like a Bluetooth keyboard.

We have detailed walkthroughs for Windows, Mac and Linux (Raspberry Pi)

We have also paired it without difficulty to iOS devices such as iPhone and iPad (any version) and an Android tablet but don't have a detailed walkthrough. Check your device's documentation on how to pair a keyboard. It's usually really easy and just requires turning on BT and then scanning for the powered up module.

You only have to pair once to your device. After that, it will auto-connect.

If you ever have difficulties with auto-connecting, especially if you do a system update or upgrade, just follow the pairing procedure from the beginning. It only takes a few minutes. The following GIF shows meaning of the red LED on board.

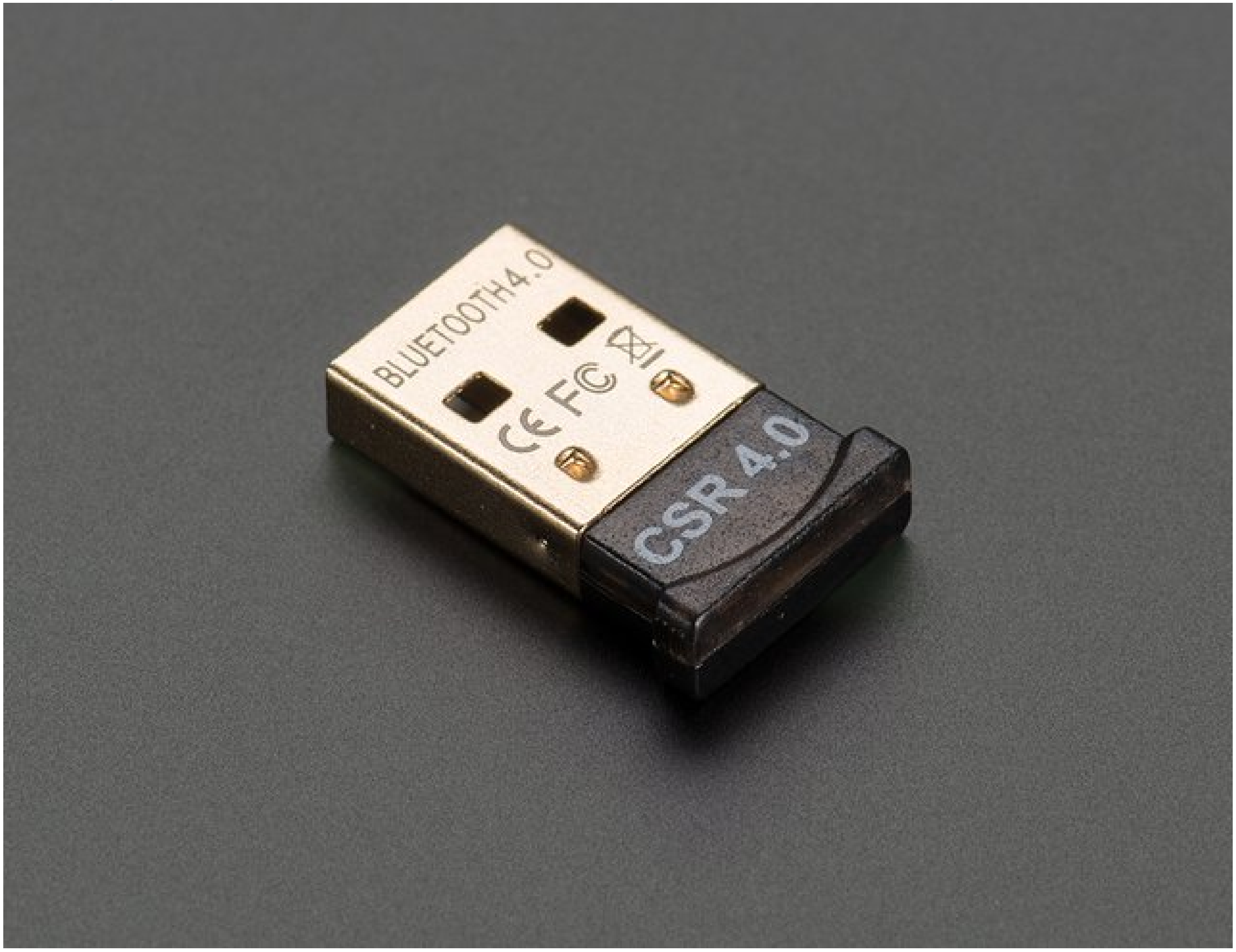


Windows

This page will show you how to pair your Bluefruit EZ-Key to a Windows computer. It's tested on XP and 7 but should work similarly with Windows 8.

You only have to pair once - after the EZ-Key is paired to a computer it will auto-connect from then on

First up, you'll have to make sure you have Bluetooth v2.1 or greater on your computer. Many laptops have BT built in and unless its a really old machine (< 2008), the built in BT should be OK. If you do not have BT built in, you'll need a USB dongle such as this one (http://adafru.it/1327)



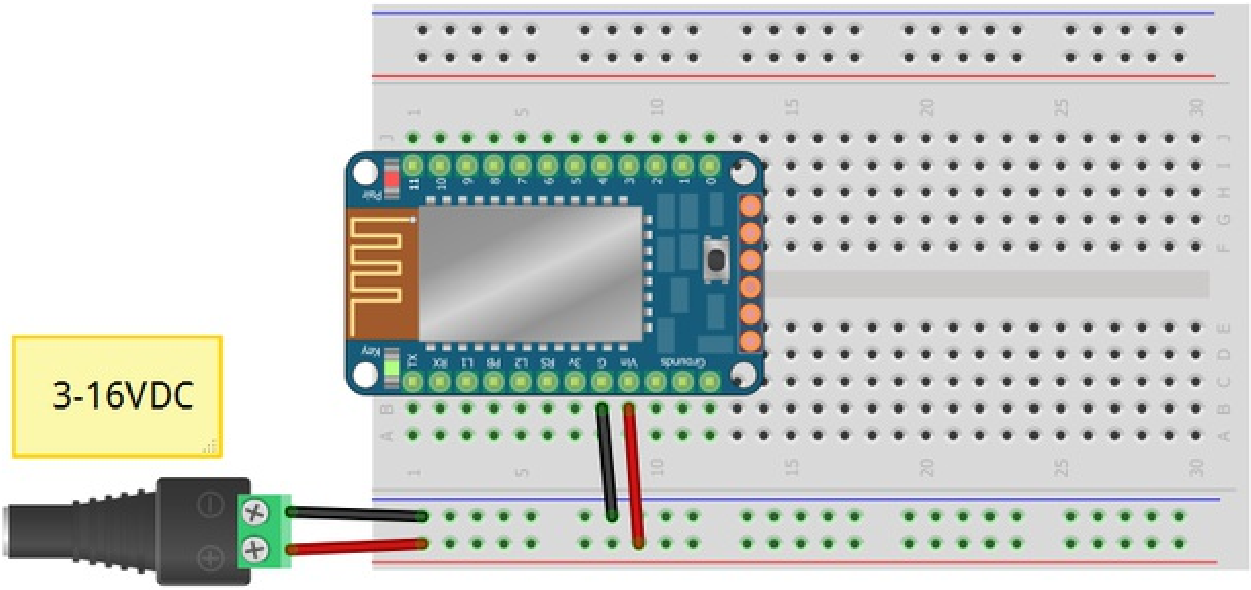
Many ultra-low cost USB adapters you may find are BT v2.0 and NOT v2.1. You MUST have a v2.1 or greater adapter, as v2.0 does not support the way we handle pairing. If you get a BT v4 module you will have no problems, so please do not use "$2" adapters!

Step 0. Install USB adapter

99% of the time, you can just plug it in and Windows will automatically install the drivers, as there are only two main chipsets (CSR & Broadcom) and they have built in support.

Step 1. Power the Bluefruit EZ-Key and Press the Pair Button

The title of this step is pretty much what you have to do. Remember that you have solder the 0.1" headers to the module or at least solder wires to the **Vin** and **Ground** connections. Connect Vin to 3 to 16VDC (5V is ideal) and ground to the ground power wire.

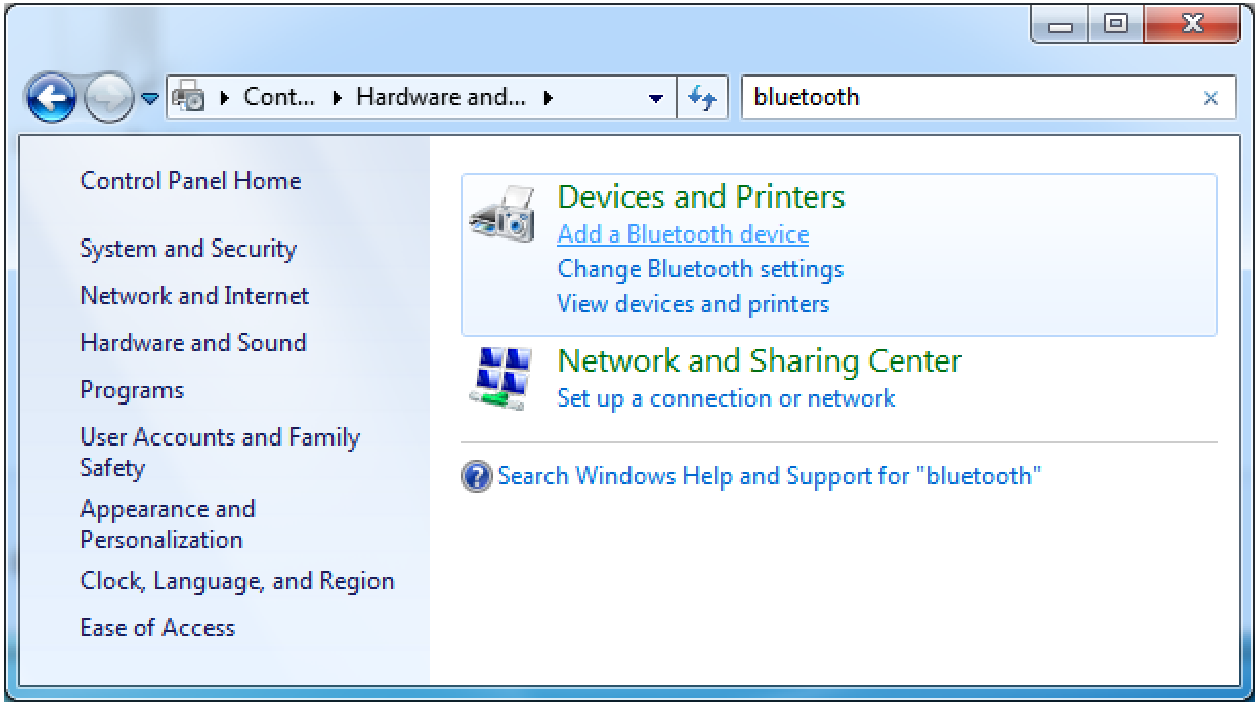


You should see the red LED blink. Now **press the mini button on the EZ Key for 5 seconds** and release, this will erase any old pairing information and let you re-pair to your computer.

The red LED will now blink at a steady once-a-second.

Step 2 Pair using Windows Bluetooth services

In the Control Panel, find the **Add Bluetooth Device** entry



Cont... ▸ Hardware and... ▸
bluetooth
Control Panel Home
System and Security
Network and Internet
Hardware and Sound
Programs
User Accounts and Family Safety
Appearance and Personalization
Clock, Language, and Region
Ease of Access

Devices and Printers

Add a Bluetooth device

Change Bluetooth settings

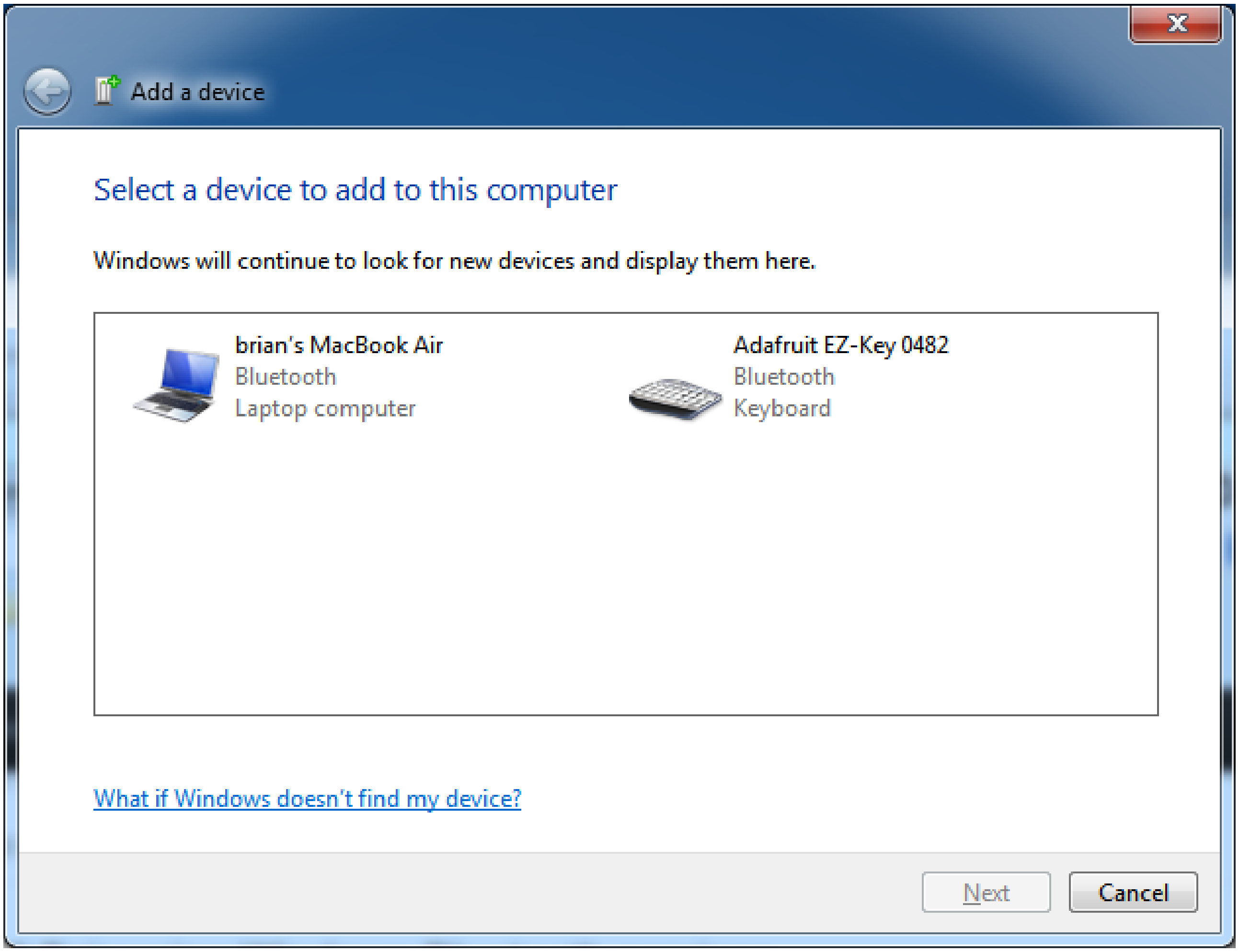
View devices and printers

Network and Sharing Center

Set up a connection or network

Search Windows Help and Support for "bluetooth"

Wait a minute until you see the Adafruit Keyboard device show up with the full ID name, it will look like this:



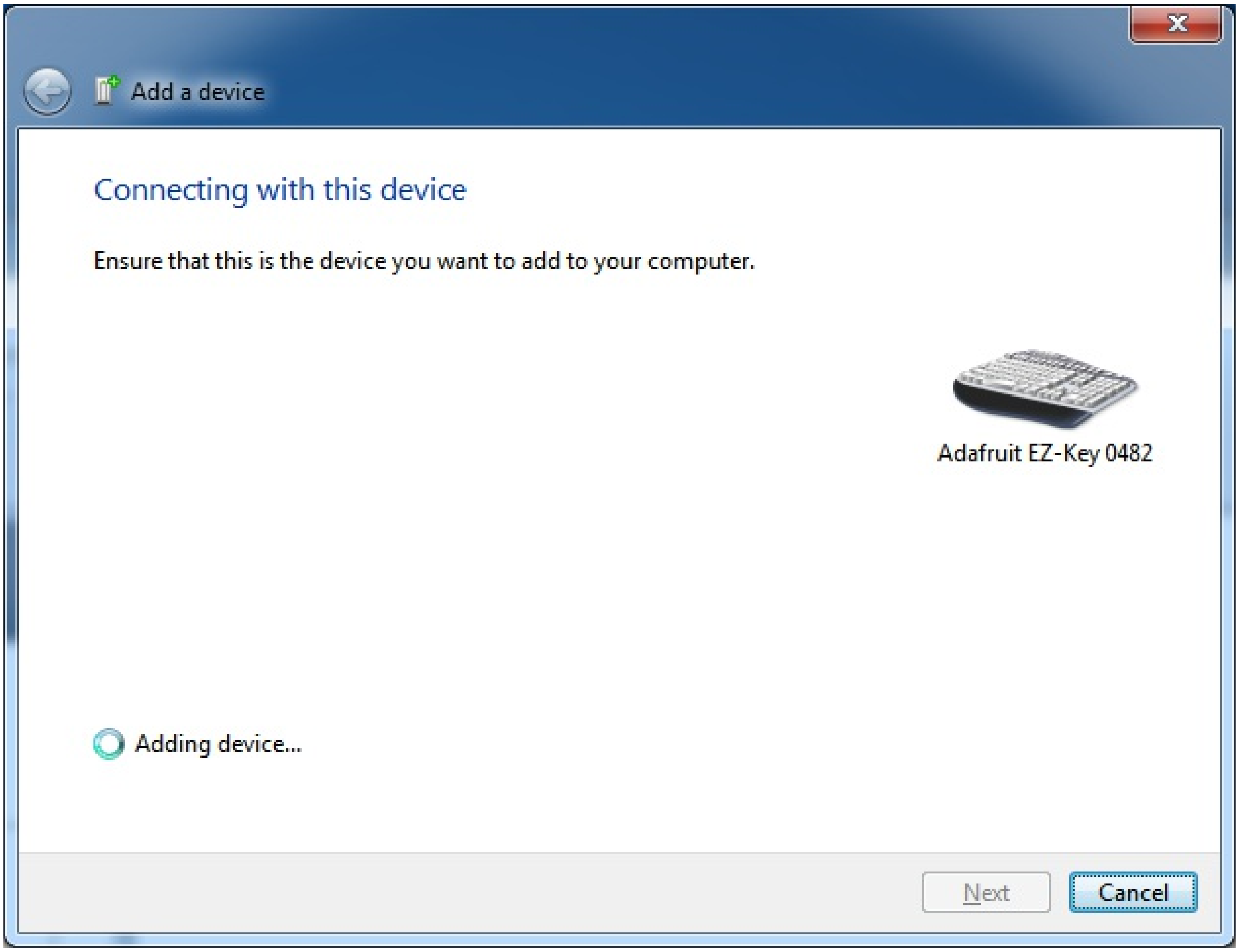
Select a device to add to this computer

Windows will continue to look for new devices and display them here.

brian's MacBook Air
Bluetooth
Laptop computer

Adafruit EZ-Key 0482
Bluetooth
Keyboard

Select the Adafruit EZ-Key and press **Next**



Add a device

Connecting with this device

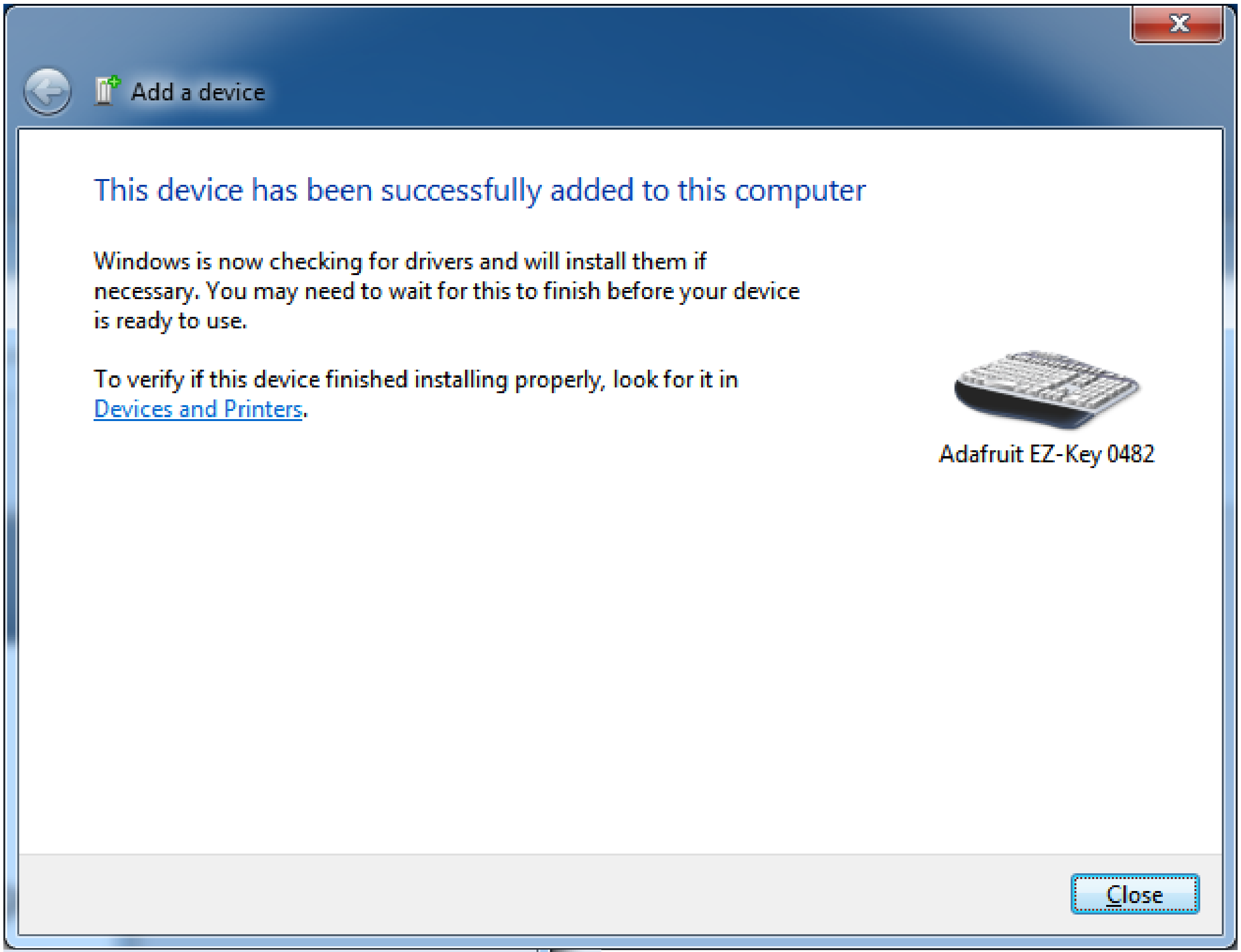
Ensure that this is the device you want to add to your computer.

Adafruit EZ-Key 0482

Adding device...

It may take up to 3 minutes to get the driver and install it, this is normal. If Windows complains about a timeout, just start the process over (it's rare for that to happen)

If it asks you for a passcode, the pairing code is **1234**

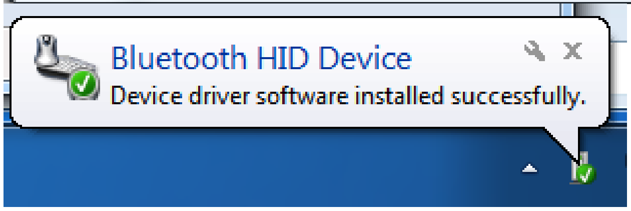


This device has been successfully added to this computer

Windows is now checking for drivers and will install them if necessary. You may need to wait for this to finish before your device is ready to use.

To verify if this device finished installing properly, look for it in Devices and Printers.

Adafruit EZ-Key 0482



Bluetooth HID Device

Device driver software installed successfully.

That's it! You will now see the red LED blinking much slower, to indicate it is paired succesfully

Mac

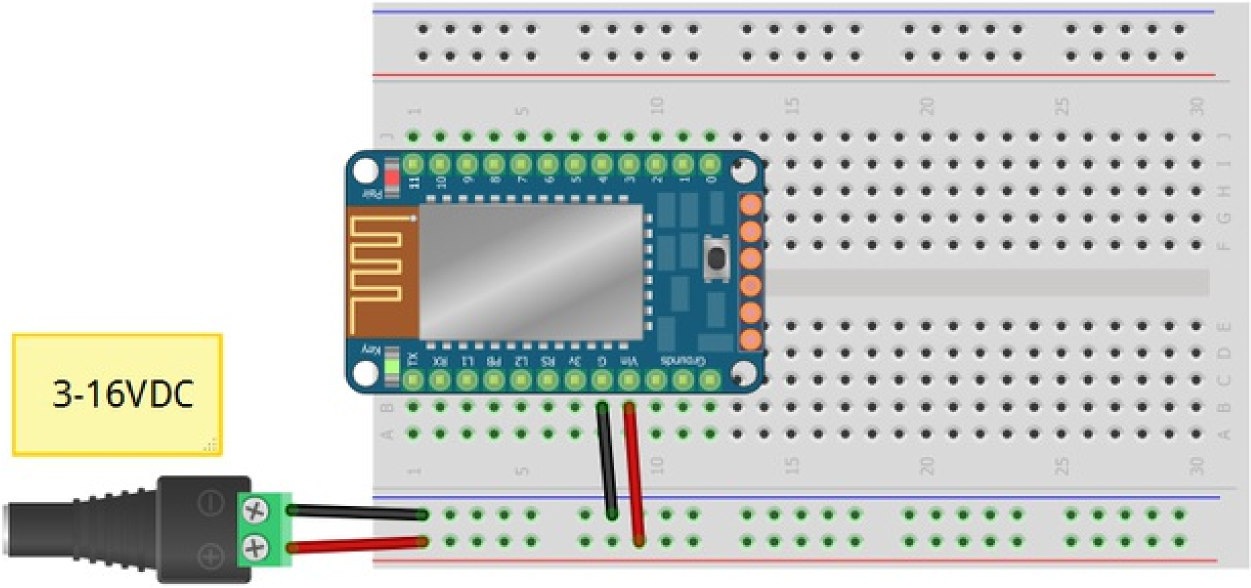
This page will show you how to pair your Bluefruit EZ-Key to a Mac OS X computer. Connecting to iOS is nearly identical.

You only have to pair once - after the EZ-Key is paired to a computer it will auto-connect from then on

We have never found a Mac without BT built in so lucky for you, no extra module is required! (http://adafru.it/1327)

Step 1. Power the Bluefruit EZ-Key and Press the Pair Button

The title of this step is pretty much what you have to do. Remember that you have solder the 0.1" headers to the module or at least solder wires to the **Vin** and **Ground** connections. Connect Vin to 3 to 16VDC (5V is ideal) and ground to the ground power wire.

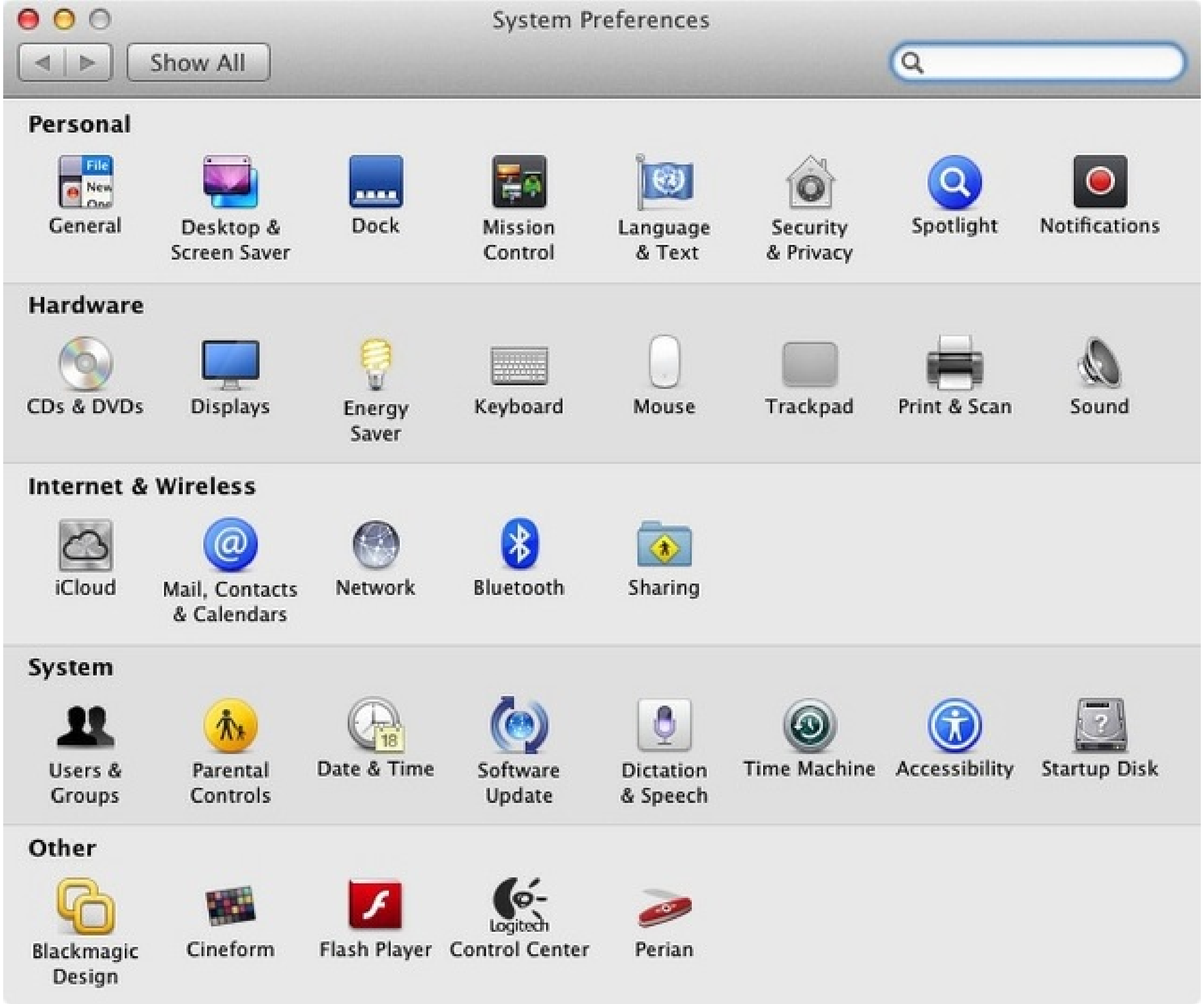


You should see the red LED blink. Now **press the mini button on the EZ Key for 5 seconds** and release, this will erase any old pairing information and let you re-pair to your computer.

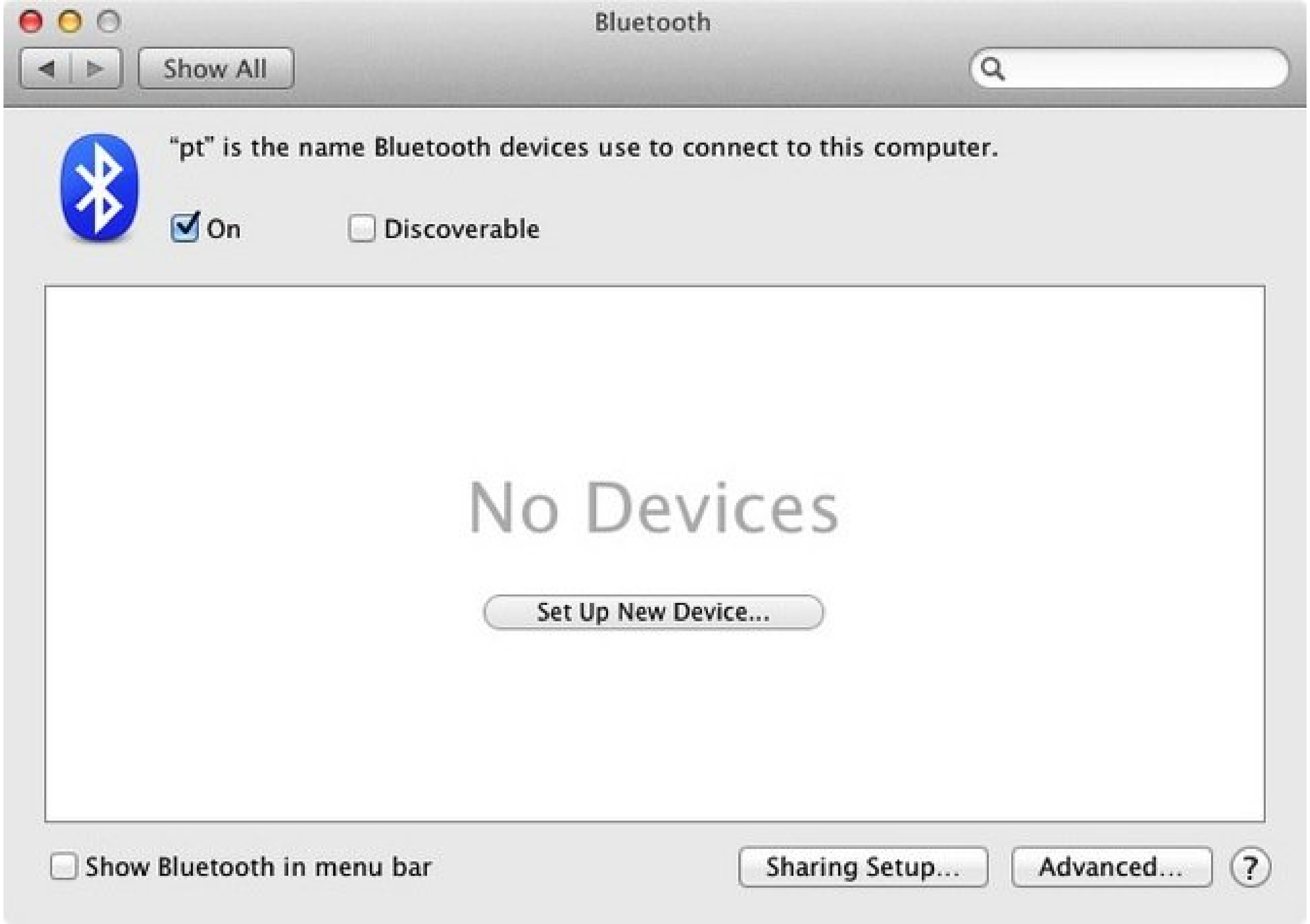
The red LED will now blink at a steady once-a-second.

Step 2. Pair Using MacOS Bluetooth Service

In the System Preferences, find the Bluetooth icon



Double-click to open, make sure Bluetooth is **ON** (some computers have it off, you must have BT on!) and click **Set Up New Device**



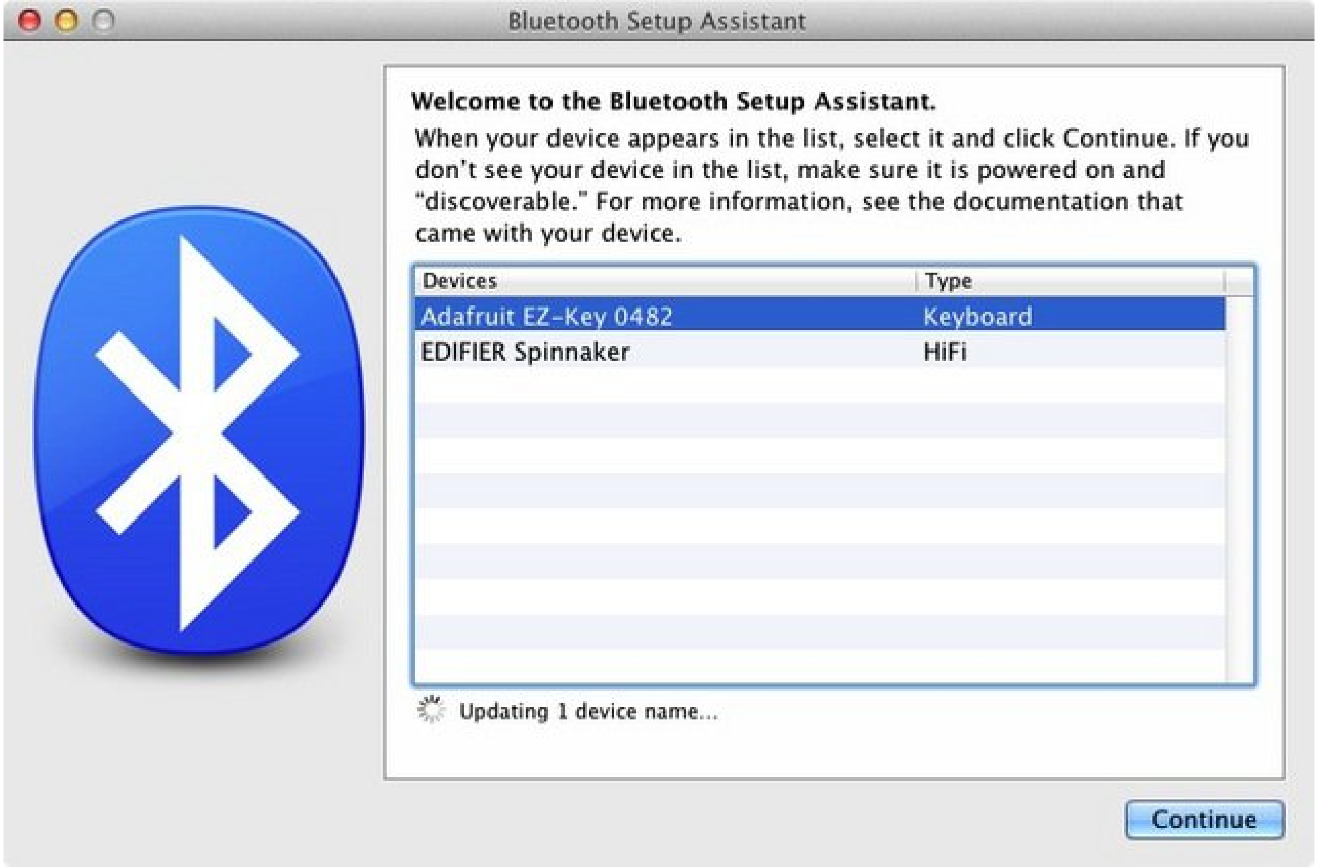
"pt" is the name Bluetooth devices use to connect to this computer.

☑ On ☐ Discoverable

No Devices

☐ Show Bluetooth in menu bar

Let the assistant run for a minute until it locates and displays the EZ-Key module
Select it and click Continue



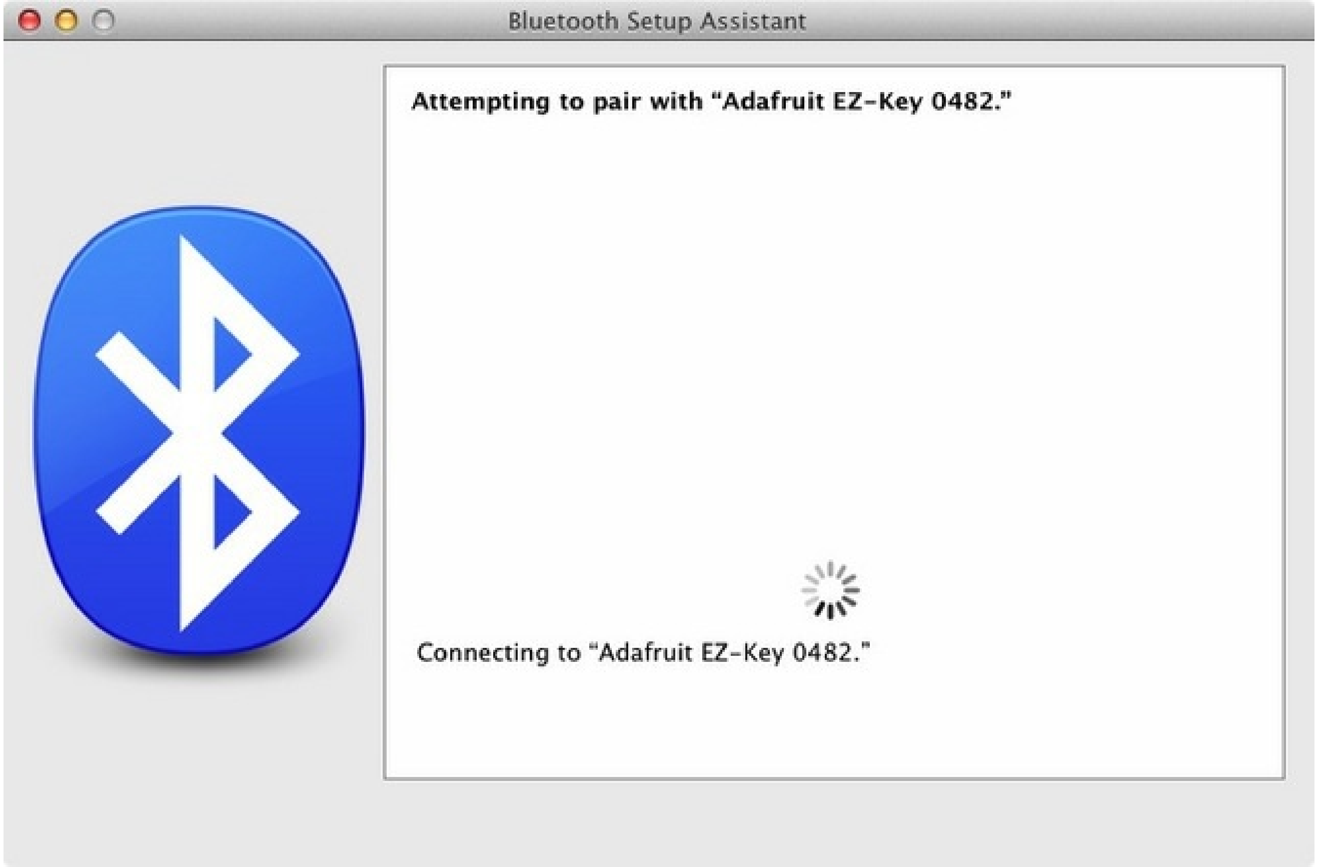
Welcome to the Bluetooth Setup Assistant.

When your device appears in the list, select it and click Continue. If you don't see your device in the list, make sure it is powered on and "discoverable." For more information, see the documentation that came with your device.

Devices	Type
Adafruit EZ-Key 0482	Keyboard
EDIFIER Spinnaker	HiFi

Updating 1 device name...

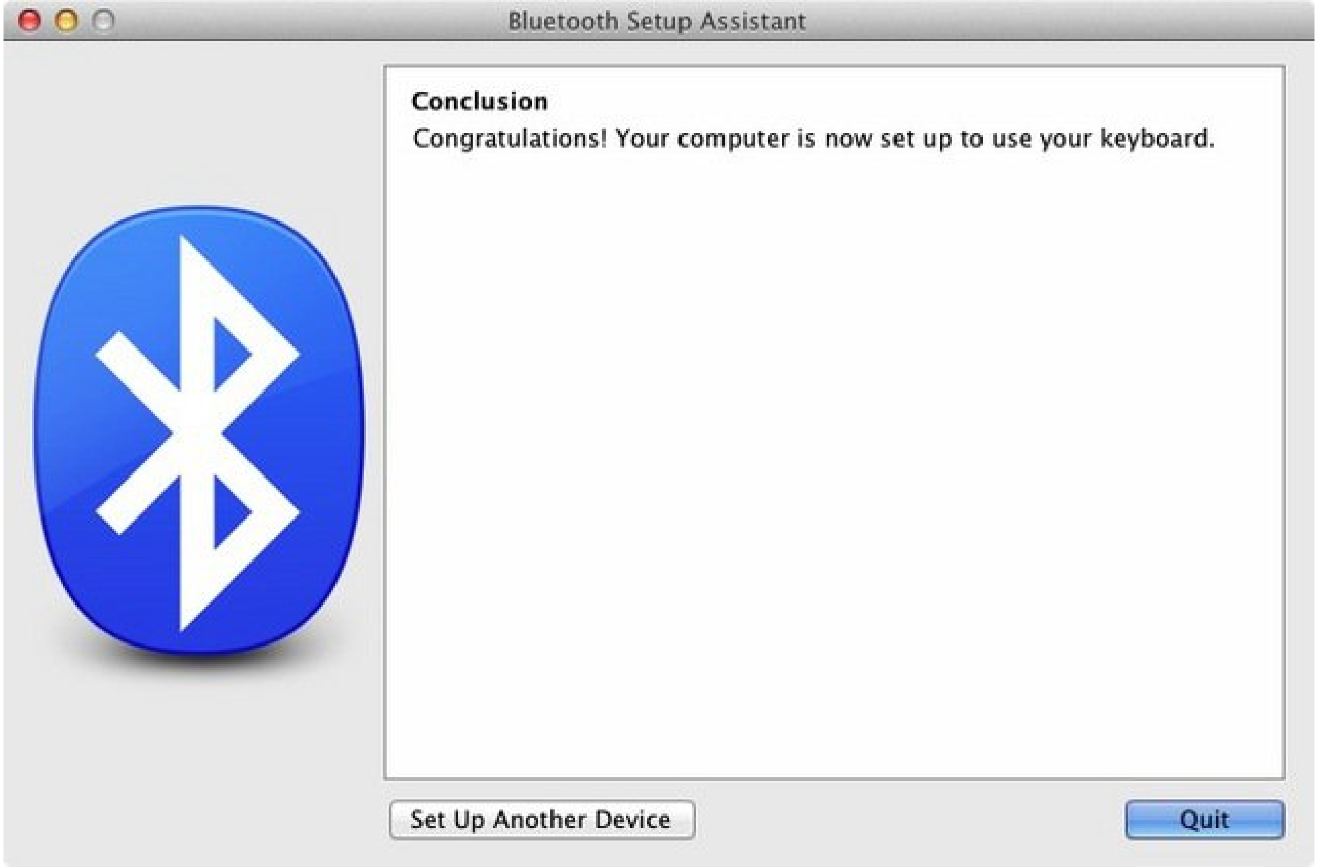
It'll take a minute for it to pair



Attempting to pair with "Adafruit EZ-Key 0482."

Connecting to "Adafruit EZ-Key 0482."

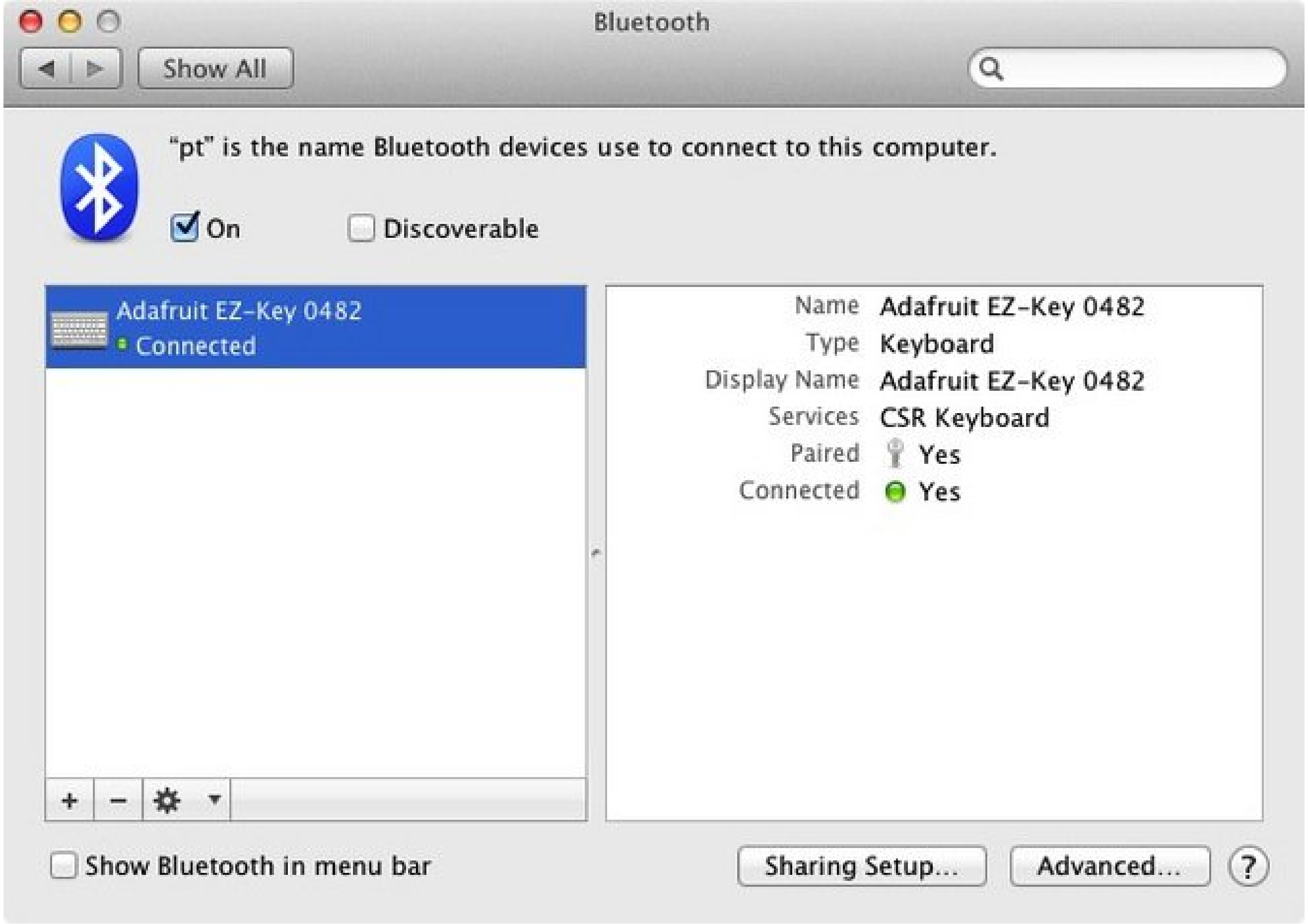
That's it! If you get a pop-up asking you to configure the keyboard layout, just close or quit that window.



Conclusion

Congratulations! Your computer is now set up to use your keyboard.

You can now see that the EZ-Key is paired and connected.



"pt" is the name Bluetooth devices use to connect to this computer.

☑ On
□ Discoverable

Adafruit EZ-Key 0482
● Connected

Name	Adafruit EZ-Key 0482
Type	Keyboard
Display Name	Adafruit EZ-Key 0482
Services	CSR Keyboard
Paired	Yes
Connected	● Yes

□ Show Bluetooth in menu bar

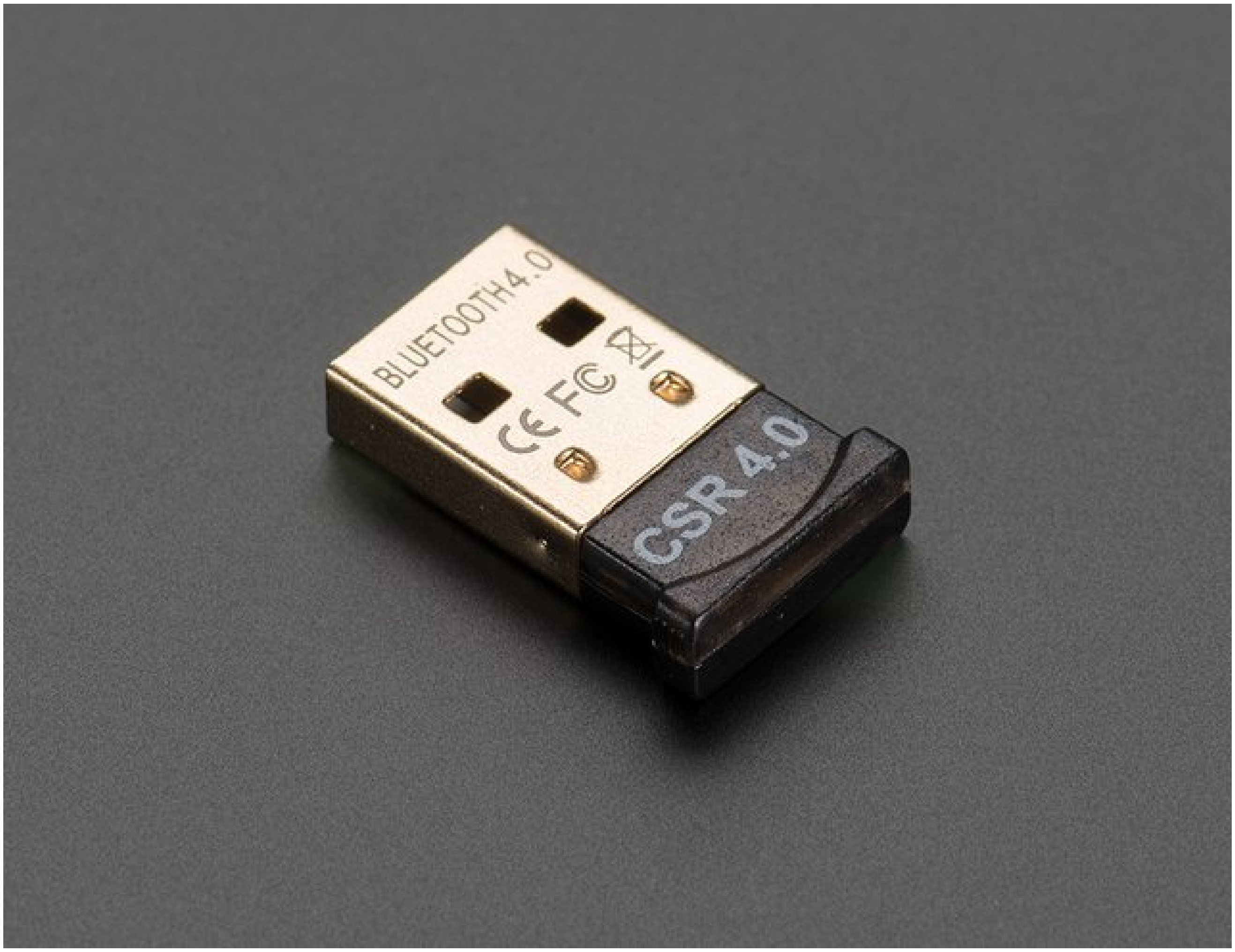
Linux (e.g. Raspberry Pi)

This page will show you how to pair your Bluefruit EZ-Key to a Linux computer. It's tested on Raspberry Pi & Raspbian but the instructions will be similar for other machines and distros (we hope!) Check your distro documentation if this doesn't work.

Thanks to http://www.correlatedcontent.com/blog/bluetooth-keyboard-on-the-raspberry-pi/ (http://adafru.it/cJW) *for the details!*

You only have to pair once - after the EZ-Key is paired to a computer it will auto-connect from then on

First up, you'll have to make sure you have Bluetooth v2.1 or greater on your computer. Many laptops have BT built in and unless its a really old machine (< 2008), the built in BT should be OK. If you do not have BT built in, you'll need a USB dongle such as this one (http://adafru.it/1327)



Many ultra-low cost USB adapters you may find are BT v2.0 and NOT v2.1. You MUST have a v2.1 or greater adapter, as v2.0 does not support the way we handle pairing. If you get a BT v4 module you will have no problems, so please do not use "$2" adapters!

Step 0 Plug in BT Adapter

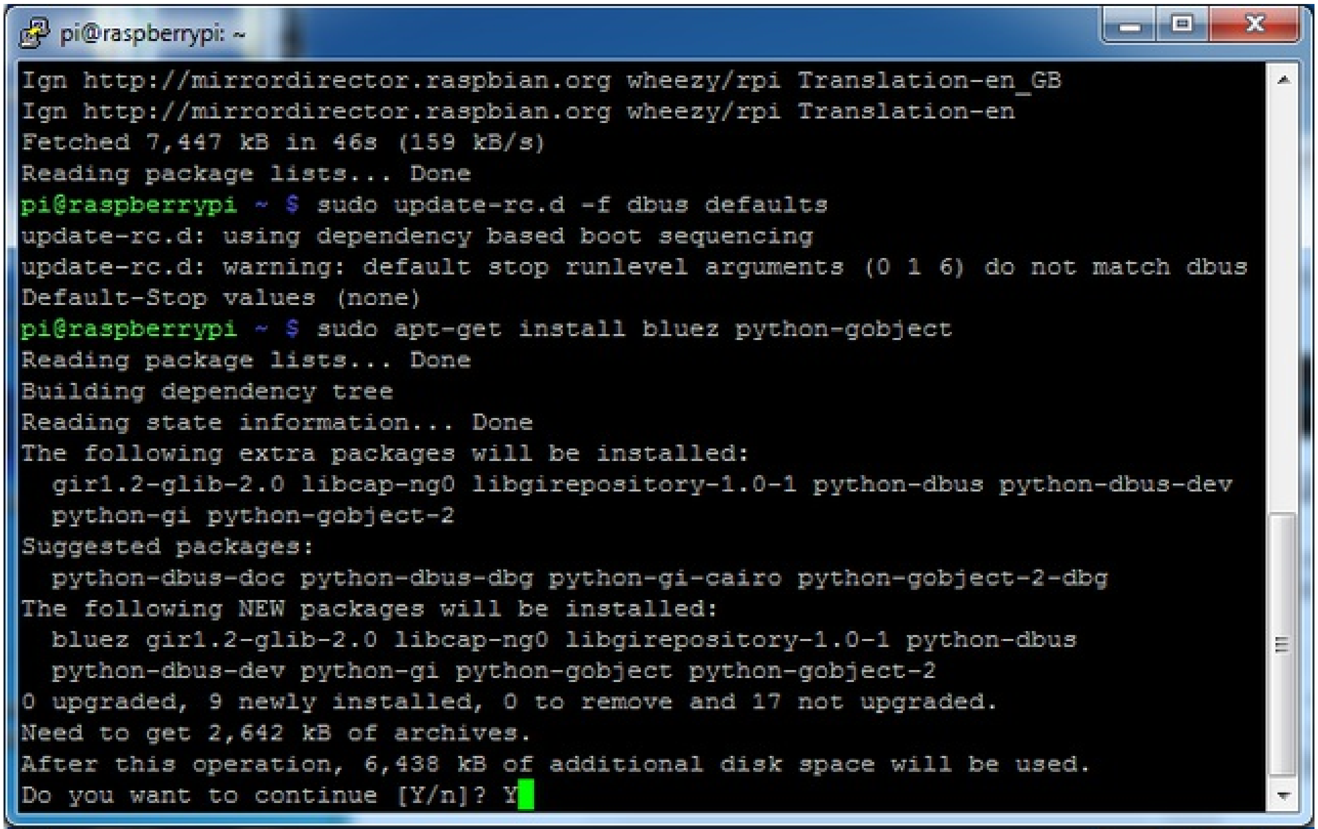
With the Raspberry Pi off, plug in the BT module and reboot.

Step 1 Update & Install Bluez

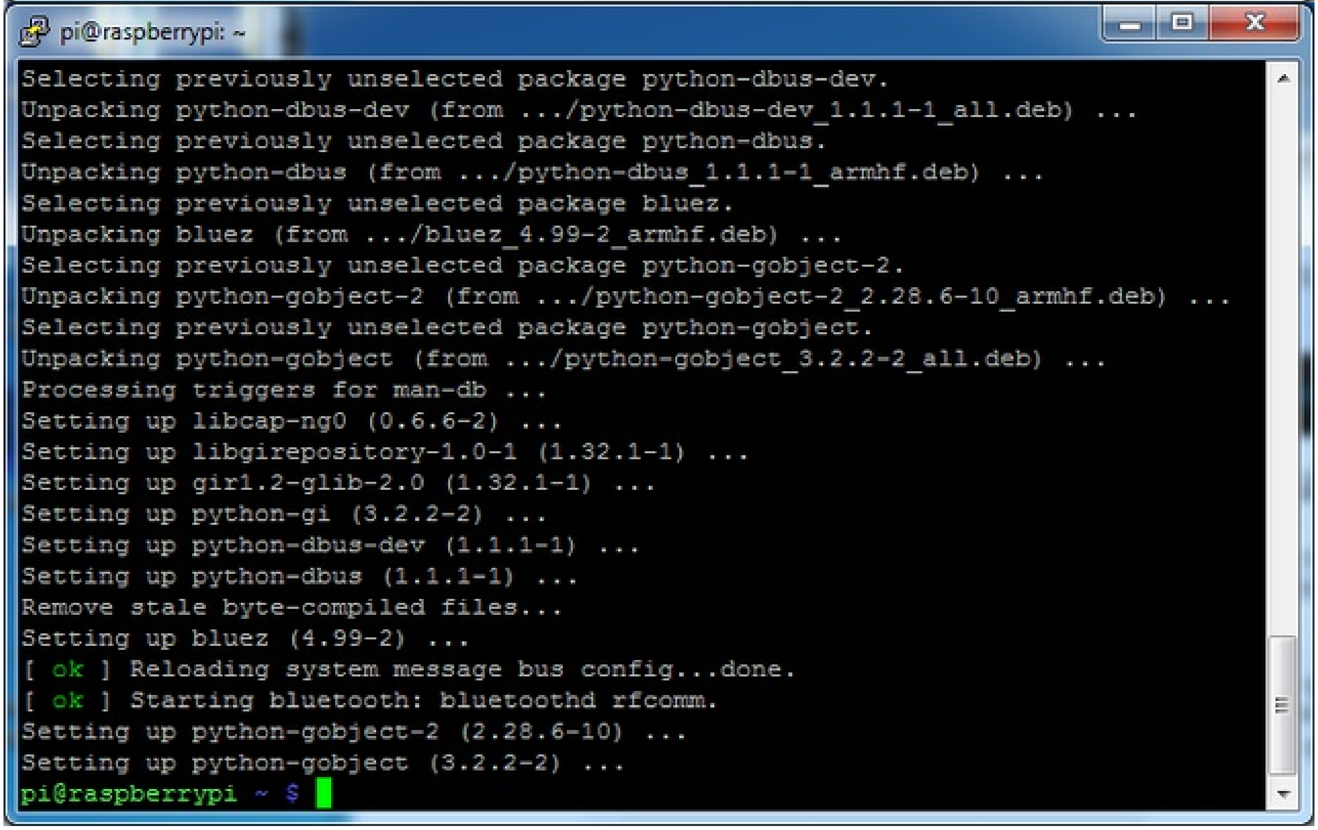
Make sure you have Internet connectivity on your Pi so you can install the following updates & software for Bluetooth control.

All of the following must be typed into a Terminal window or Console or Command line.

sudo apt-get update
sudo update-rc.d -f dbus defaults
sudo apt-get install bluez python-gobject



```
Ign http://mirrordirector.raspbian.org wheezy/rpi Translation-en_GB
Ign http://mirrordirector.raspbian.org wheezy/rpi Translation-en
Fetched 7,447 kB in 46s (159 kB/s)
Reading package lists... Done
pi@raspberrypi ~ $ sudo update-rc.d -f dbus defaults
update-rc.d: using dependency based boot sequencing
update-rc.d: warning: default stop runlevel arguments (0 1 6) do not match dbus
Default-Stop values (none)
pi@raspberrypi ~ $ sudo apt-get install bluez python-gobject
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following extra packages will be installed:
  gir1.2-glib-2.0 libcap-ng0 libgirepository-1.0-1 python-dbus python-dbus-dev
  python-gi python-gobject-2
Suggested packages:
  python-dbus-doc python-dbus-dbg python-gi-cairo python-gobject-2-dbg
The following NEW packages will be installed:
  bluez gir1.2-glib-2.0 libcap-ng0 libgirepository-1.0-1 python-dbus
  python-dbus-dev python-gi python-gobject python-gobject-2
0 upgraded, 9 newly installed, 0 to remove and 17 not upgraded.
Need to get 2,642 kB of archives.
After this operation, 6,438 kB of additional disk space will be used.
Do you want to continue [Y/n]? Y
```

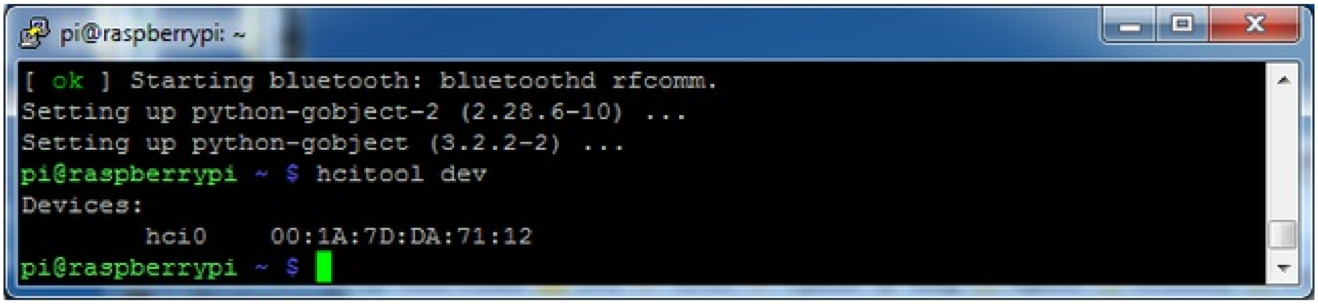


```
Selecting previously unselected package python-dbus-dev.
Unpacking python-dbus-dev (from .../python-dbus-dev_1.1.1-1_all.deb) ...
Selecting previously unselected package python-dbus.
Unpacking python-dbus (from .../python-dbus_1.1.1-1_armhf.deb) ...
Selecting previously unselected package bluez.
Unpacking bluez (from .../bluez_4.99-2_armhf.deb) ...
Selecting previously unselected package python-gobject-2.
Unpacking python-gobject-2 (from .../python-gobject-2_2.28.6-10_armhf.deb) ...
Selecting previously unselected package python-gobject.
Unpacking python-gobject (from .../python-gobject_3.2.2-2_all.deb) ...
Processing triggers for man-db ...
Setting up libcap-ng0 (0.6.6-2) ...
Setting up libgirepository-1.0-1 (1.32.1-1) ...
Setting up gir1.2-glib-2.0 (1.32.1-1) ...
Setting up python-gi (3.2.2-2) ...
Setting up python-dbus-dev (1.1.1-1) ...
Setting up python-dbus (1.1.1-1) ...
Remove stale byte-compiled files...
Setting up bluez (4.99-2) ...
[ ok ] Reloading system message bus config...done.
[ ok ] Starting bluetooth: bluetoothd rfcomm.
Setting up python-gobject-2 (2.28.6-10) ...
Setting up python-gobject (3.2.2-2) ...
pi@raspberrypi ~ $
```

Now run

hcitool dev

to see the bluetooth USB module

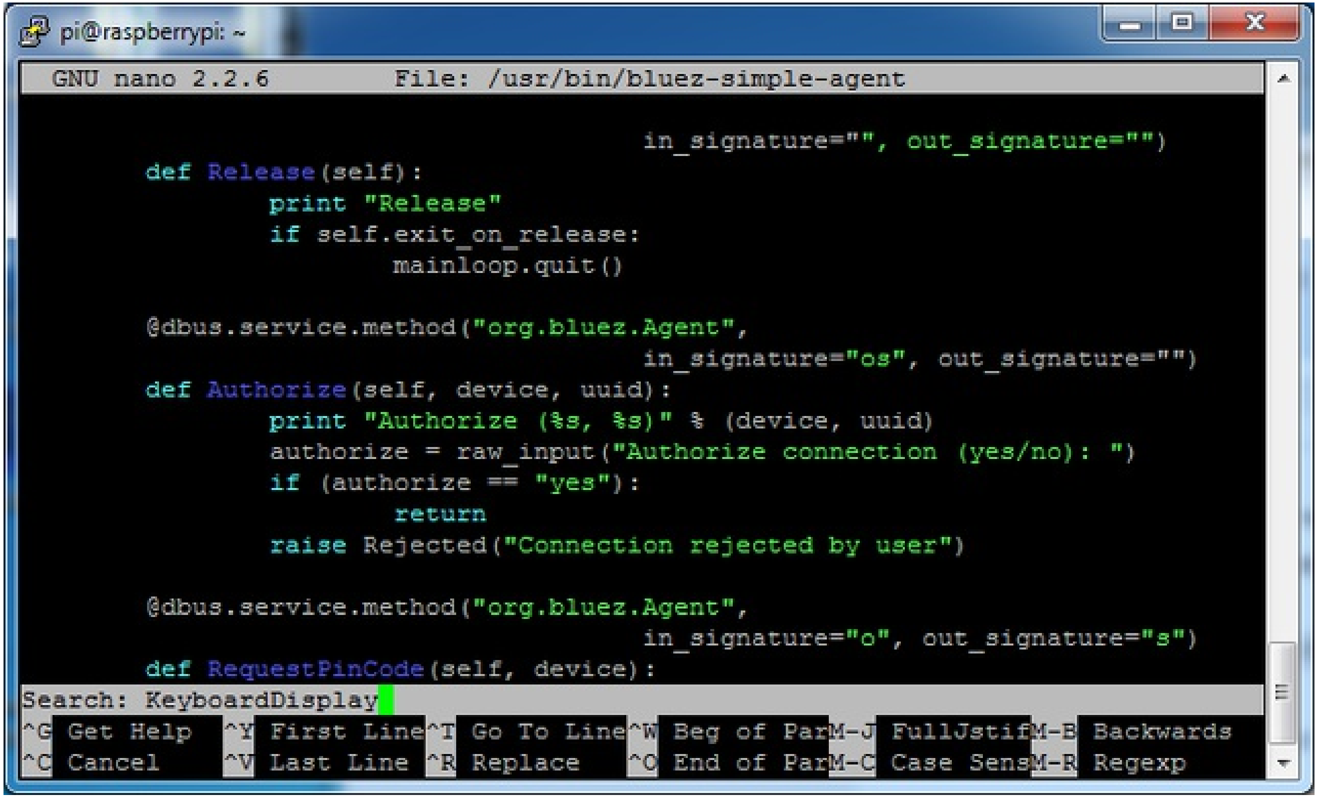


```
[ ok ] Starting bluetooth: bluetoothd rfcomm.
Setting up python-gobject-2 (2.28.6-10) ...
Setting up python-gobject (3.2.2-2) ...
pi@raspberrypi ~ $ hcitool dev
Devices:
        hci0    00:1A:7D:DA:71:12
pi@raspberrypi ~ $
```

Lastly, we'll make a minor edit to allow passkey-less pairing. Run

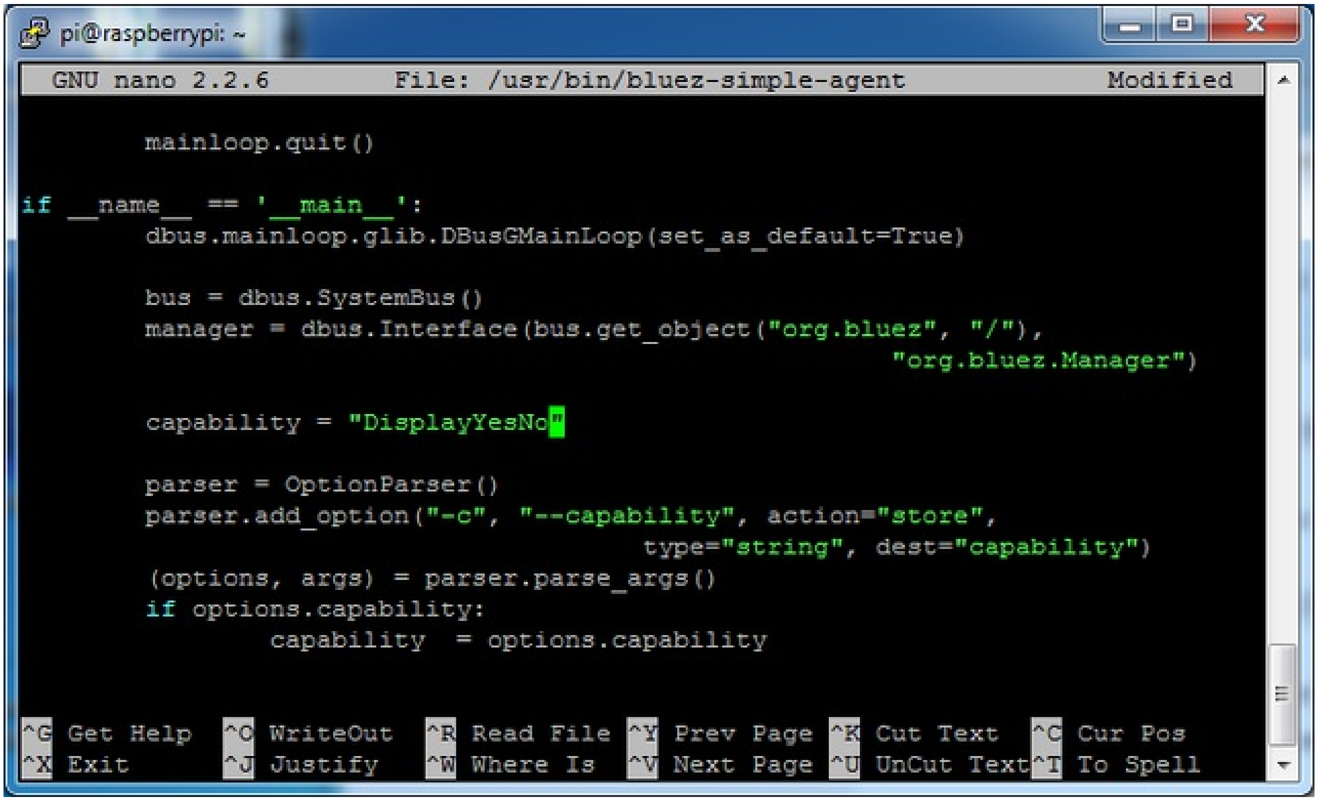
sudo nano /usr/bin/bluez-simple-agent

To edit the agent that manages BT pairing. Type**Control-W** to search for **KeyboardDisplay**



```
  GNU nano 2.2.6        File: /usr/bin/bluez-simple-agent

                                        in_signature="", out_signature="")
        def Release(self):
                print "Release"
                if self.exit_on_release:
                        mainloop.quit()

        @dbus.service.method("org.bluez.Agent",
                                        in_signature="os", out_signature="")
        def Authorize(self, device, uuid):
                print "Authorize (%s, %s)" % (device, uuid)
                authorize = raw_input("Authorize connection (yes/no): ")
                if (authorize == "yes"):
                        return
                raise Rejected("Connection rejected by user")

        @dbus.service.method("org.bluez.Agent",
                                        in_signature="o", out_signature="s")
        def RequestPinCode(self, device):
Search: KeyboardDisplay
^G Get Help  ^Y First Line^T Go To Line^W Beg of ParM-J FullJstifM-B Backwards
^C Cancel    ^V Last Line ^R Replace   ^O End of ParM-C Case SensM-R Regexp
```

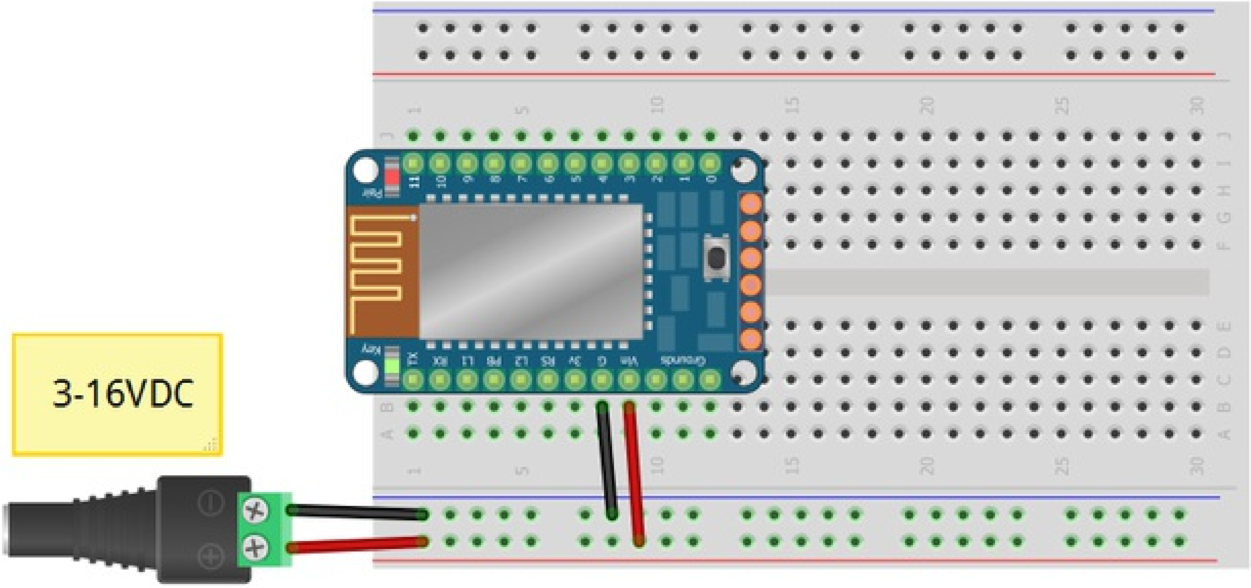
Then edit that line and change **KeyoardDisplay** to **DisplayYesNo**



```
  GNU nano 2.2.6        File: /usr/bin/bluez-simple-agent                Modified

        mainloop.quit()

if __name__ == '__main__':
        dbus.mainloop.glib.DBusGMainLoop(set_as_default=True)

        bus = dbus.SystemBus()
        manager = dbus.Interface(bus.get_object("org.bluez", "/"),
                                                        "org.bluez.Manager")

        capability = "DisplayYesNo"

        parser = OptionParser()
        parser.add_option("-c", "--capability", action="store",
                                type="string", dest="capability")
        (options, args) = parser.parse_args()
        if options.capability:
                capability  = options.capability

^G Get Help   ^O WriteOut   ^R Read File  ^Y Prev Page  ^K Cut Text   ^C Cur Pos
^X Exit       ^J Justify    ^W Where Is   ^V Next Page  ^U UnCut Text^T To Spell
```

Step 2. Power the Bluefruit EZ-Key and Press the Pair Button

The title of this step is pretty much what you have to do. Remember that you have solder the 0.1" headers to the module or at least solder wires to the **Vin** and **Ground** connections. Connect Vin to 3 to 16VDC (5V is ideal) and ground to the ground power wire.



You should see the red LED blink. Now **press the mini button on the EZ Key for 5 seconds** and release, this will erase any old pairing information and let you re-pair to your computer.

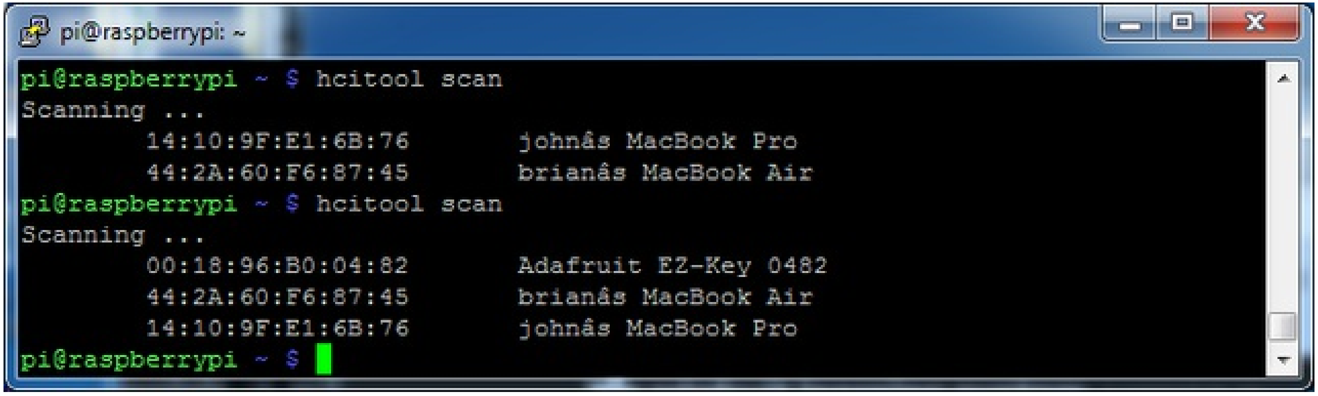
The red LED will now blink at a steady once-a-second.

Step 3. Scan & Connect to Bluefruit Module

Now it's time to find the Bluefruit device. Run

```
hcitool scan
```

to scan for devices. You may have to run it once or twice to see the Adafruit device pop up



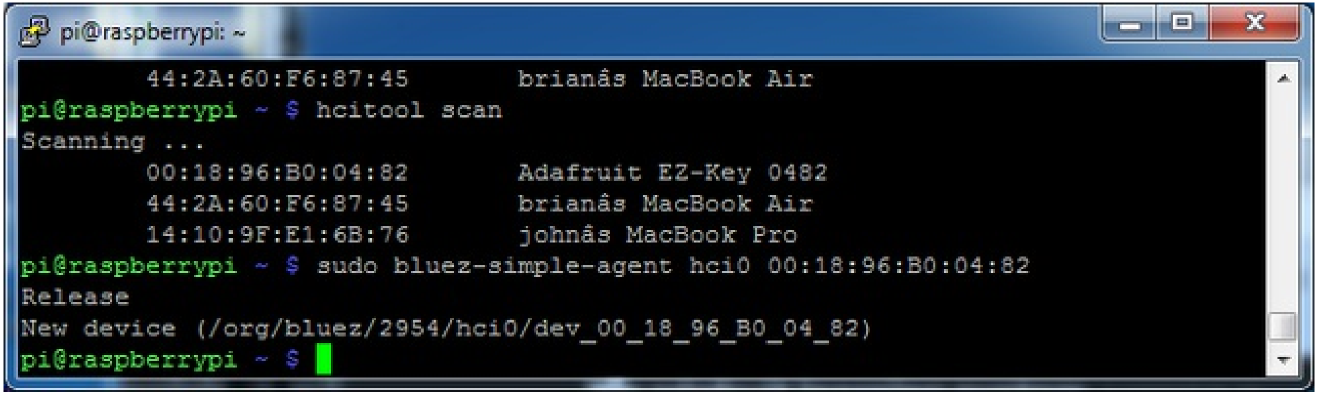
```
pi@raspberrypi ~ $ hcitool scan
Scanning ...
        14:10:9F:E1:6B:76       johnâs MacBook Pro
        44:2A:60:F6:87:45       brianâs MacBook Air
pi@raspberrypi ~ $ hcitool scan
Scanning ...
        00:18:96:B0:04:82       Adafruit EZ-Key 0482
        44:2A:60:F6:87:45       brianâs MacBook Air
        14:10:9F:E1:6B:76       johnâs MacBook Pro
pi@raspberrypi ~ $
```

See that long number before the name? Starts with 00:18:... ? Each module has a **unique** identifier number. Your setup will have a different ID so be sure to type out the exact same ID you have. We will proceed as if you were pairing to the module on my desk :)

We will now create a device for the keyboard. Type in

sudo bluez-simple-agent hci0 00:18:96:B0:04:82

But changing it to the ID number you have



```
        44:2A:60:F6:87:45       brianâs MacBook Air
pi@raspberrypi ~ $ hcitool scan
Scanning ...
        00:18:96:B0:04:82       Adafruit EZ-Key 0482
        44:2A:60:F6:87:45       brianâs MacBook Air
        14:10:9F:E1:6B:76       johnâs MacBook Pro
pi@raspberrypi ~ $ sudo bluez-simple-agent hci0 00:18:96:B0:04:82
Release
New device (/org/bluez/2954/hci0/dev_00_18_96_B0_04_82)
pi@raspberrypi ~ $
```

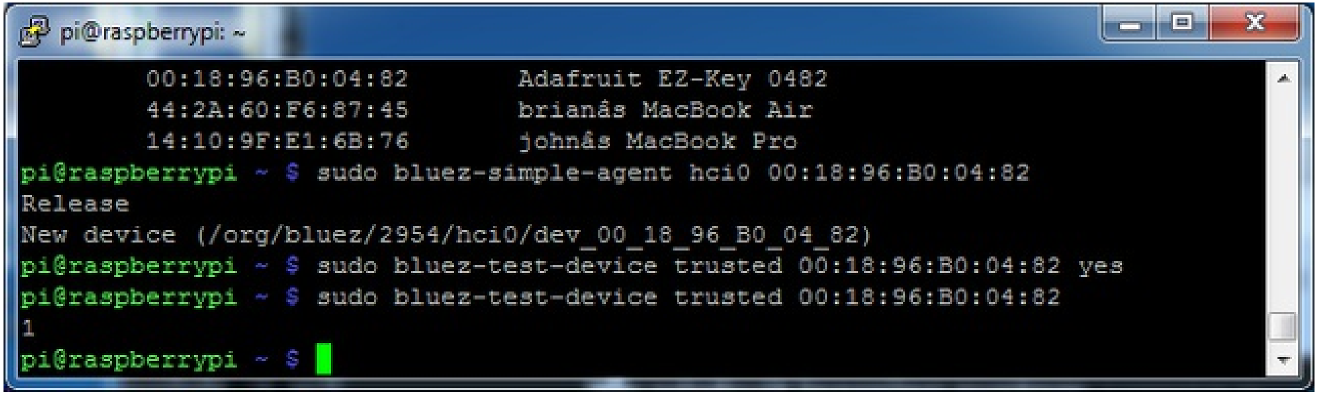
Next, we will *trust* this keyboard. Type in

sudo bluez-test-device trusted 00:18:96:B0:04:82 yes

(don't forget that yes at the end)
followed by

sudo bluez-test-device trusted 00:18:96:B0:04:82

(no yes at the end)
You should see a **1** after the last command. If you get a**0** try again, check that you typed the #'s right.



```
        00:18:96:B0:04:82       Adafruit EZ-Key 0482
        44:2A:60:F6:87:45       brianâs MacBook Air
        14:10:9F:E1:6B:76       johnâs MacBook Pro
pi@raspberrypi ~ $ sudo bluez-simple-agent hci0 00:18:96:B0:04:82
Release
New device (/org/bluez/2954/hci0/dev_00_18_96_B0_04_82)
pi@raspberrypi ~ $ sudo bluez-test-device trusted 00:18:96:B0:04:82 yes
pi@raspberrypi ~ $ sudo bluez-test-device trusted 00:18:96:B0:04:82
1
pi@raspberrypi ~ $
```

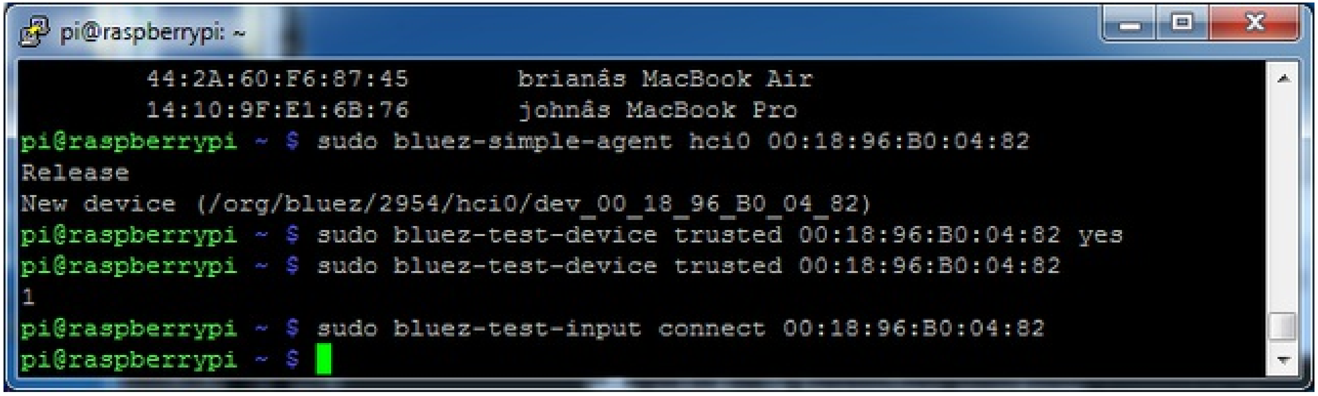
Finally, we can connect! The last command to run is:

sudo bluez-test-input connect 00:18:96:B0:04:82

If you want to ever remove the pairing, type in

sudo bluez-test-device remove 00:18:96:B0:04:82

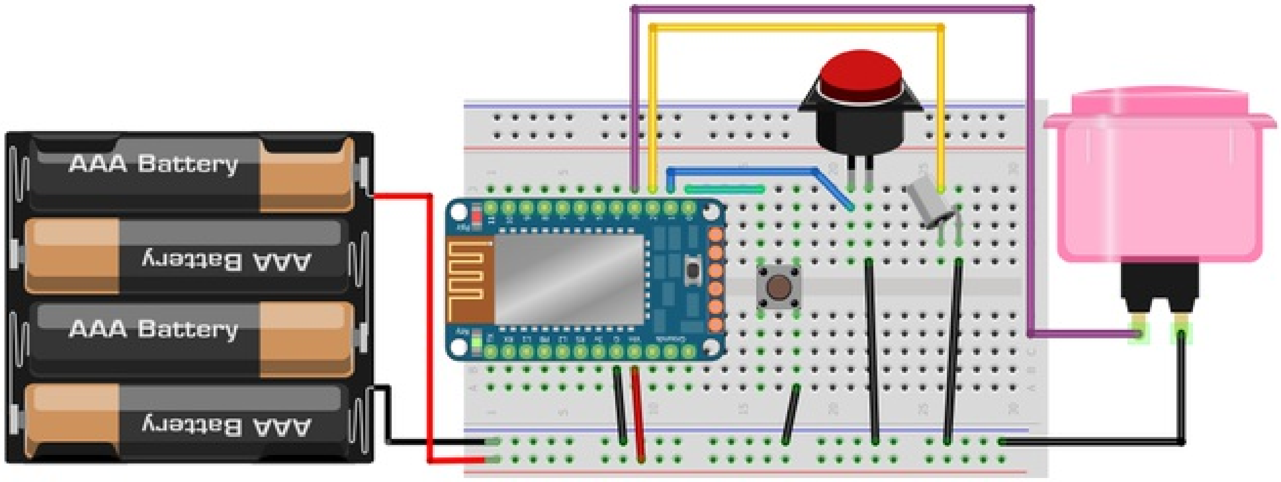
Now you will notice the red LED on the module blink slower.



```
       44:2A:60:F6:87:45       brianâs MacBook Air
       14:10:9F:E1:6B:76       johnâs MacBook Pro
pi@raspberrypi ~ $ sudo bluez-simple-agent hci0 00:18:96:B0:04:82
Release
New device (/org/bluez/2954/hci0/dev_00_18_96_B0_04_82)
pi@raspberrypi ~ $ sudo bluez-test-device trusted 00:18:96:B0:04:82 yes
pi@raspberrypi ~ $ sudo bluez-test-device trusted 00:18:96:B0:04:82
1
pi@raspberrypi ~ $ sudo bluez-test-input connect 00:18:96:B0:04:82
pi@raspberrypi ~ $ 
```

REMEMBER! This is a USB keyboard so if you are SSH'd or connecting via a Console cable, you WON'T see keystrokes appear. On a Raspberry Pi you have to connect a TV to the Composite or HDMI outputs to see the keyboard input.

User Manual



The user manual is the shortest page of this guide, because its *really really easy* to get going.

1. Power the EZ-Key with 3-16VDC power. Batteries work great: 3 or 4 alkaline or rechargeable 1.5V, a 9V, Lithium Ion/Polymer, Lead acid... Whatever you have!
2. Pair the EZ-Key to your computer, laptop, tablet or phone
3. Connect one side of a switch to GPIO #0 through #11. Connect the other side to Ground.
4. Open up a notepad or text editor on the paired computer
5. Press the switch to send a key code
6. Profit?

You will be able to see the green LED blink every time it detects a switch and sends the keycode.

Since GPIO #0 through #3 are arrow keys, they might be more difficult to detect if the notepad software is empty. Try GPIO #4 through #11 which send printable characters

When a switch is pressed, a KEYDOWN command is sent, when it is release, a matching KEYUP goes out. You can have up to 6 switches pressed at once and it will be like they were pressed all at the same time. 6 is a strict limit of Bluetooth.

Don't forget: You don't have to use a plain clicky switch! Try tilt sensors, reed switches, conductive velcro, big stompy buttons, arcade joysticks, ANYTHING that

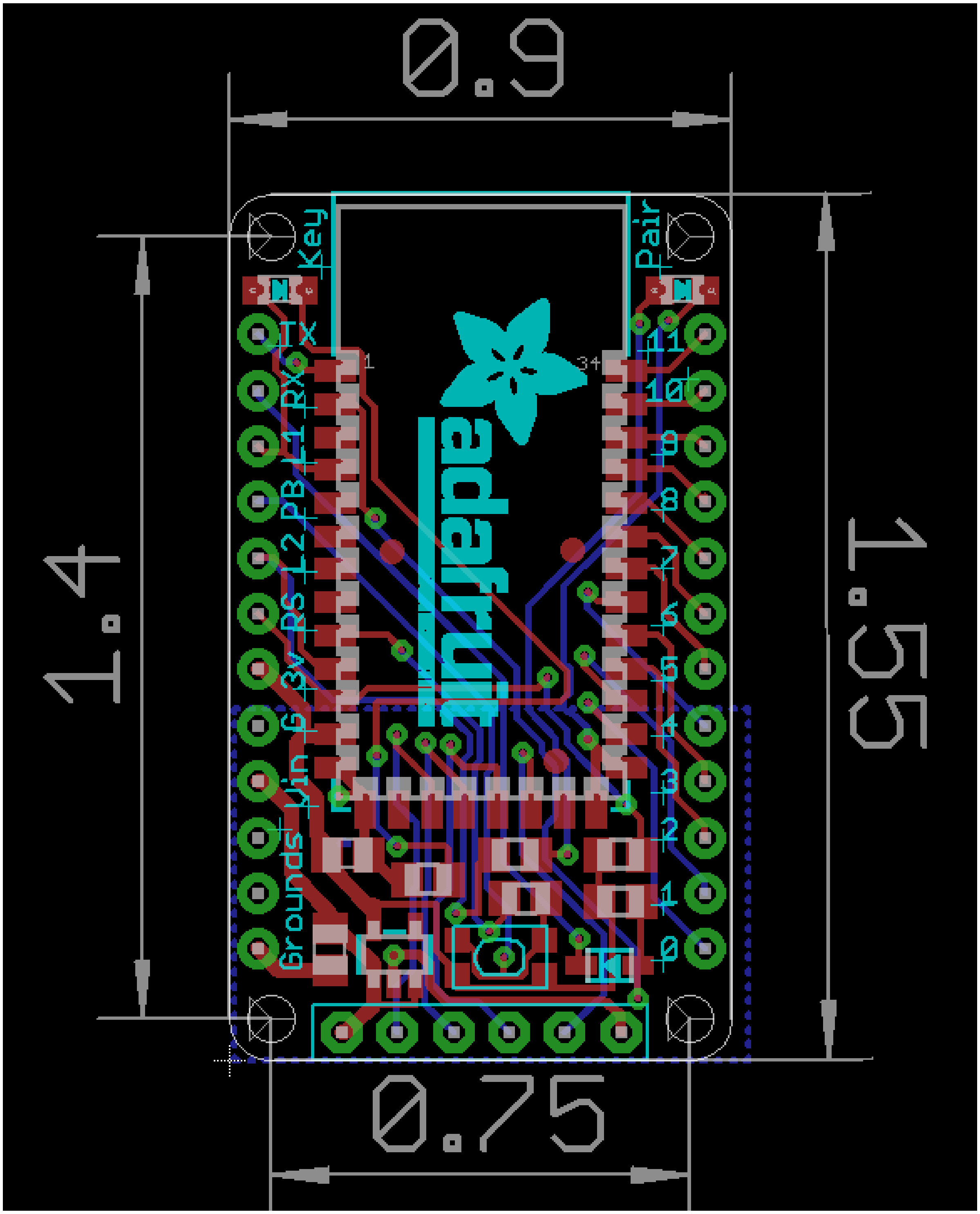
makes/breaks two contacts

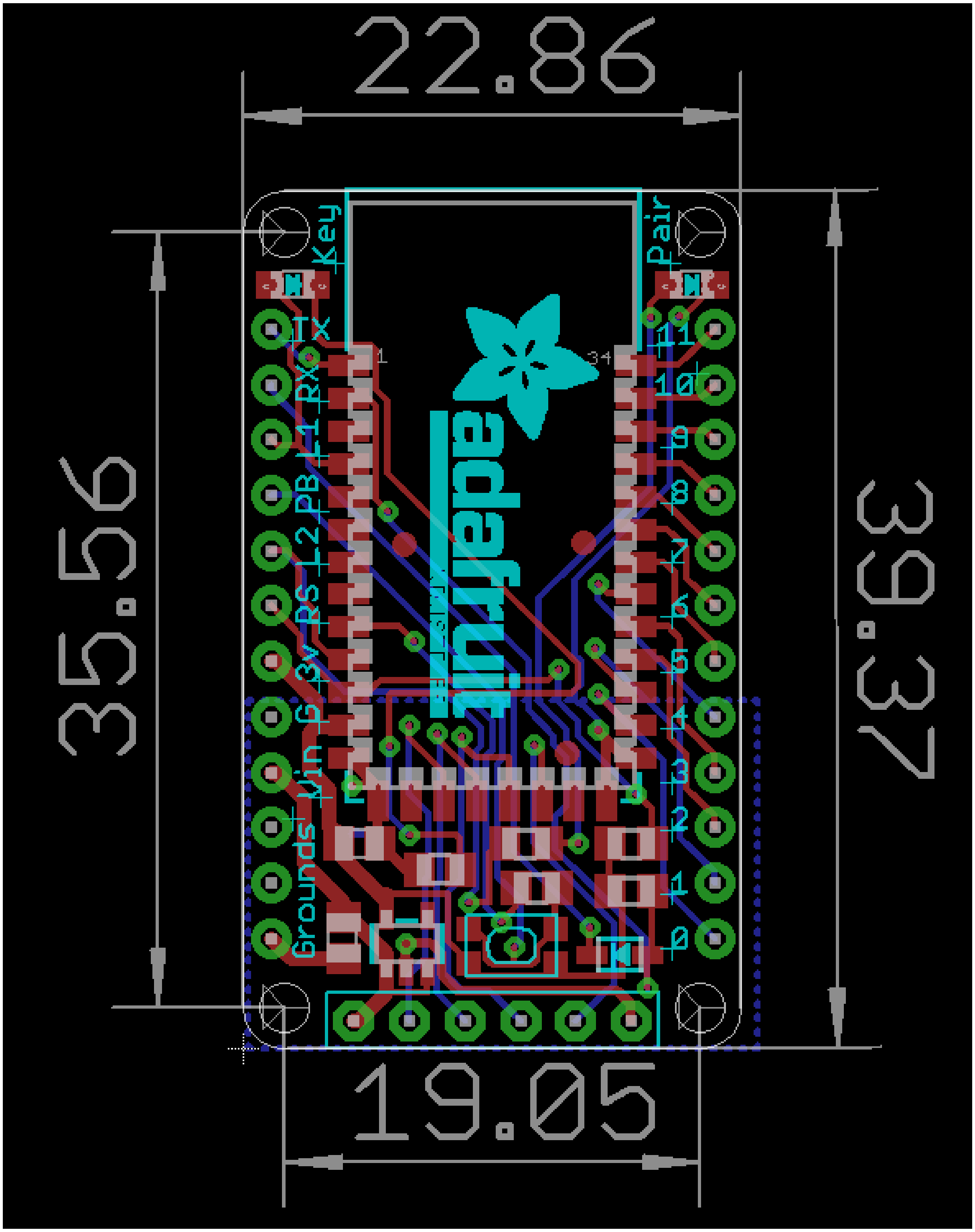
Here is the default switch-to-key mapping:

- #0 - Up Arrow
- #1 - Down Arrow
- #2 - Left Arrow
- #3 - Right Arrow
- #4 - Return
- #5 - Space
- #6 - the number '1'
- #7 - the number '2'
- #8 - lowercase 'w'
- #9 - lowercase 'a'
- #10 - lowercase 's'
- #11 - lowercase 'd'

You can customize these with a little bit of effort, see the Remapping Buttons page.

Dimensions in Inches & mm


0.9
1.4
1.55
0.75
Key
Pair
TX
RX
L1
PB
L2
RS
3v
G
Vin
Grounds
1
34
11
10
9
8
7
6
5
4
3
2
1
0
adafruit


22.86
39.37
35.56
19.05
Key
Pair
Tx
RX
L1
PB
L2
RS
3v
G
Vin
Grounds
11
10
9
8
7
6
5
4
3
2
1
0
1
34
adafruit

Sending Keys/Mouse Via Serial

For advanced users, you may want to hook up your Bluefruit to a microcontroller and send characters or strings via Bluetooth.

You can do this with the UART port on the EZ-Key. The UART pins are labeled RX (data going *into* the module) and TX (debug data coming *out* of the module). You can get away with just connecting to RX. The RX pin is 5V compliant, you can send it 3V or 5V TTL logic levels. Use 9600 baud serial, all microcontrollers will support this.

You can also send raw HID Keyboard reports for complex key-stroke combinations and controls.

In version v1.1 (Oct 22, 2013 or later) HID Mouse reports are also supported, you can send mouse movement and clicks via the UART and microcontroller
In version v1.2 (Nov 2013 or later) HID consumer report keys are supported, there are a few supported 'multimedia keys' - see below for a list and how to send via the UART

Don't forget to also tie a ground pin from your microcontroller to the EZ-Key for the logic ground reference!

Printable character keymap

For printing ASCII characters, you can simply send those to the UART and they will be 'typed out'. See below for the list of printable ASCII characters, starting with 0x20 and ending with 0x7E

Thanks to Wikipedia for this nice chart!(http://adafru.it/cJX)

Binary	Oct	Dec	Hex	Glyph
010 0000	040	32	20	
010 0001	041	33	21	!
010 0010	042	34	22	"
010 0011	043	35	23	#
010 0100	044	36	24	$
010 0101	045	37	25	%

010 0110	046	38	26	&
010 0111	047	39	27	'
010 1000	050	40	28	(
010 1001	051	41	29	)
010 1010	052	42	2A	*
010 1011	053	43	2B	+
010 1100	054	44	2C	,
010 1101	055	45	2D	-
010 1110	056	46	2E	.
010 1111	057	47	2F	/
011 0000	060	48	30	0
011 0001	061	49	31	1
011 0010	062	50	32	2
011 0011	063	51	33	3
011 0100	064	52	34	4
011 0101	065	53	35	5
011 0110	066	54	36	6
011 0111	067	55	37	7
011 1000	070	56	38	8
011 1001	071	57	39	9
011 1010	072	58	3A	:
011 1011	073	59	3B	;
011 1100	074	60	3C	<
011 1101	075	61	3D	=
011 1110	076	62	3E	>
011 1111	077	63	3F	?
Binary	**Oct**	**Dec**	**Hex**	**Glyph**
100 0000	100	64	40	@
100 0001	101	65	41	A
100 0010	102	66	42	B
100 0011	103	67	43	C
100 0100	104	68	44	D
100 0101	105	69	45	E
100 0110	106	70	46	F
100 0111	107	71	47	G

100 1000	110	72	48	H
100 1001	111	73	49	I
100 1010	112	74	4A	J
100 1011	113	75	4B	K
100 1100	114	76	4C	L
100 1101	115	77	4D	M
100 1110	116	78	4E	N
100 1111	117	79	4F	O
101 0000	120	80	50	P
101 0001	121	81	51	Q
101 0010	122	82	52	R
101 0011	123	83	53	S
101 0100	124	84	54	T
101 0101	125	85	55	U
101 0110	126	86	56	V
101 0111	127	87	57	W
101 1000	130	88	58	X
101 1001	131	89	59	Y
101 1010	132	90	5A	Z
101 1011	133	91	5B	[
101 1100	134	92	5C	\
101 1101	135	93	5D	]
101 1110	136	94	5E	^
101 1111	137	95	5F	_

Binary	Oct	Dec	Hex	Glyph
110 0000	140	96	60	`
110 0001	141	97	61	a
110 0010	142	98	62	b
110 0011	143	99	63	c
110 0100	144	100	64	d
110 0101	145	101	65	e
110 0110	146	102	66	f
110 0111	147	103	67	g
110 1000	150	104	68	h
110 1001	151	105	69	i

110 1010	152	106	6A	j
110 1011	153	107	6B	k
110 1100	154	108	6C	l
110 1101	155	109	6D	m
110 1110	156	110	6E	n
110 1111	157	111	6F	o
111 0000	160	112	70	p
111 0001	161	113	71	q
111 0010	162	114	72	r
111 0011	163	115	73	s
111 0100	164	116	74	t
111 0101	165	117	75	u
111 0110	166	118	76	v
111 0111	167	119	77	w
111 1000	170	120	78	x
111 1001	171	121	79	y
111 1010	172	122	7A	z
111 1011	173	123	7B	{
111 1100	174	124	7C	|
111 1101	175	125	7D	}
111 1110	176	126	7E	~

Non-printable Characters

There's a lot of other keycodes you may want to send, here is the mapping from the 8-bit byte to the keycode sent.

HEX	Keyname
0x01	Insert
0x02	Home
0x03	Page Up
0x04	Delete
0x05	End

0x06	Page Down
0x07	Right Arrow
0x08	Backspace
0x09	Tab
0x0A	Enter
0x0B	Left Arrow
0x0C	Down Arrow
0x0D	Enter
0x0E	Up Arrow
0x0F - 0x1A	F1 - F12
0x1B	Esc
0x1C	Caps Lock
0x1D	Scroll Lock
0x1E	Break
0x1F	Num Lock
0x20-0x7E	Printable Ascii
0x7F	Toggle iOS Keyboard
0xE0	Left Control
0xE1	Left Shift
0xE2	Left Alt
0xE3	Left GUI
0xE4	Right Control
0xE5	Right Shift
0xE6	Right Alt
0xE7	Right GUI

Raw HID Keyboard Reports

Bluefruit can send raw HID Keyboard reports. This allows sending any modifier keys + up to 6 keycodes at once. Its advanced but super useful for when you want to have fine-control of keypresses!

Raw HID reports start with **0xFD** and have 8 bytes following. For keyboard, its

0xFD [modifiers] 0x00 [keycode1] [keycode2] [keycode3] [keycode4] [keycode5] [keycode6]

Raw USB HID keycodes are not the same as ASCII!

Here's a list of USB HID keycodes (its in java format but you get the idea), you can also get another list here http://www.freebsddiary.org/APC/usb_hid_usages.php (http://adafru.it/cQV) under "7 Keyboard"

```
// Bits in usbHidKeyboardInput.modifiers
final byte MODIFIER_NONE          =byte((0));
final byte MODIFIER_CONTROL_LEFT  =byte((1<<0));
final byte MODIFIER_SHIFT_LEFT    =byte((1<<1));
final byte MODIFIER_ALT_LEFT      =byte((1<<2));
final byte MODIFIER_GUI_LEFT      =byte((1<<3));
final byte MODIFIER_CONTROL_RIGHT =byte((1<<4));
final byte MODIFIER_SHIFT_RIGHT   =byte((1<<5));
final byte MODIFIER_ALT_RIGHT     =byte((1<<6));
final byte MODIFIER_GUI_RIGHT     =byte((1<<7));

// Values for usbHidKeyboardInput.keyCodes
// Only the key codes for common keys are defined here. See Hut1_12.pdf for a full list.
final byte KEY_NONE               =byte(0x00);
final byte KEY_A                  =byte(0x04);
final byte KEY_B                  =byte(0x05);
final byte KEY_C                  =byte(0x06);
final byte KEY_D                  =byte(0x07);
final byte KEY_E                  =byte(0x08);
final byte KEY_F                  =byte(0x09);
final byte KEY_G                  =byte(0x0A);
final byte KEY_H                  =byte(0x0B);
final byte KEY_I                  =byte(0x0C);
final byte KEY_J                  =byte(0x0D);
final byte KEY_K                  =byte(0x0E);
final byte KEY_L                  =byte(0x0F);
final byte KEY_M                  =byte(0x10);
final byte KEY_N                  =byte(0x11);
final byte KEY_O                  =byte(0x12);
final byte KEY_P                  =byte(0x13);
final byte KEY_Q                  =byte(0x14);
final byte KEY_R                  =byte(0x15);
final byte KEY_S                  =byte(0x16);
final byte KEY_T                  =byte(0x17);
final byte KEY_U                  =byte(0x18);
final byte KEY_V                  =byte(0x19);
final byte KEY_W                  =byte(0x1A);
final byte KEY_X                  =byte(0x1B);
final byte KEY_Y                  =byte(0x1C);
final byte KEY_Z                  =byte(0x1D);
final byte KEY_1                  =byte(0x1E);
final byte KEY_2                  =byte(0x1F);
final byte KEY_3              =byte(0x20);
final byte KEY_4              =byte(0x21);
final byte KEY_5              =byte(0x22);
final byte KEY_6              =byte(0x23);
final byte KEY_7              =byte(0x24);
final byte KEY_8              =byte(0x25);
final byte KEY_9              =byte(0x26);
final byte KEY_0              =byte(0x27);
final byte KEY_RETURN           =byte(0x28);
final byte KEY_ESCAPE           =byte(0x29);
final byte KEY_BACKSPACE         =byte(0x2A);
final byte KEY_TAB              =byte(0x2B);
final byte KEY_SPACE            =byte(0x2C);
final byte KEY_MINUS            =byte(0x2D);
final byte KEY_EQUAL            =byte(0x2E);
final byte KEY_BRACKET_LEFT      =byte(0x2F);
final byte KEY_BRACKET_RIGHT     =byte(0x30);
final byte KEY_BACKSLASH         =byte(0x31);
final byte KEY_EUROPE_1          =byte(0x32);
final byte KEY_SEMICOLON         =byte(0x33);
final byte KEY_APOSTROPHE        =byte(0x34);
final byte KEY_GRAVE            =byte(0x35);
final byte KEY_COMMA            =byte(0x36);
final byte KEY_PERIOD           =byte(0x37);
final byte KEY_SLASH            =byte(0x38);
final byte KEY_CAPS_LOCK         =byte(0x39);
final byte KEY_F1              =byte(0x3A);
final byte KEY_F2              =byte(0x3B);
final byte KEY_F3              =byte(0x3C);
final byte KEY_F4              =byte(0x3D);
final byte KEY_F5              =byte(0x3E);
final byte KEY_F6              =byte(0x3F);
final byte KEY_F7              =byte(0x40);
final byte KEY_F8              =byte(0x41);
final byte KEY_F9              =byte(0x42);
final byte KEY_F10              =byte(0x43);
final byte KEY_F11              =byte(0x44);
final byte KEY_F12              =byte(0x45);
final byte KEY_PRINT_SCREEN      =byte(0x46);
final byte KEY_SCROLL_LOCK       =byte(0x47);
final byte KEY_PAUSE            =byte(0x48);
final byte KEY_INSERT           =byte(0x49);
final byte KEY_HOME             =byte(0x4A);
final byte KEY_PAGE_UP          =byte(0x4B);
final byte KEY_DELETE           =byte(0x4C);
final byte KEY_END              =byte(0x4D);
final byte KEY_PAGE_DOWN         =byte(0x4E);
final byte KEY_ARROW_RIGHT       =byte(0x4F);
final byte KEY_ARROW_LEFT        =byte(0x50);
final byte KEY_ARROW_DOWN        =byte(0x51);
final byte KEY_ARROW_UP          =byte(0x52);
final byte KEY_NUM_LOCK          =byte(0x53);
final byte KEY_KEYPAD_DIVIDE      =byte(0x54);
final byte KEY_KEYPAD_MULTIPLY    =byte(0x55);
final byte KEY_KEYPAD_SUBTRACT    =byte(0x56);
final byte KEY_KEYPAD_ADD        =byte(0x57);
final byte KEY_KEYPAD_ENTER       =byte(0x58);
final byte KEY_KEYPAD_1          =byte(0x59);
final byte KEY_KEYPAD_2          =byte(0x5A);
final byte KEY_KEYPAD_3          =byte(0x5B);
final byte KEY_KEYPAD_4          =byte(0x5C);
final byte KEY_KEYPAD_5          =byte(0x5D);
final byte KEY_KEYPAD_6          =byte(0x5E);
final byte KEY_KEYPAD_7          =byte(0x5F);
final byte KEY_KEYPAD_8          =byte(0x60);
final byte KEY_KEYPAD_9          =byte(0x61);
final byte KEY_KEYPAD_0          =byte(0x62);
final byte KEY_KEYPAD_DECIMAL     =byte(0x63);
final byte KEY_EUROPE_2          =byte(0x64);
final byte KEY_APPLICATION       =byte(0x65);
final byte KEY_POWER             =byte(0x66);
final byte KEY_KEYPAD_EQUAL       =byte(0x67);
final byte KEY_F13              =byte(0x68);
final byte KEY_F14              =byte(0x69);
final byte KEY_F15              =byte(0x6A);
final byte KEY_CONTROL_LEFT       =byte(0xE0);
final byte KEY_SHIFT_LEFT        =byte(0xE1);
final byte KEY_ALT_LEFT          =byte(0xE2);
final byte KEY_GUI_LEFT          =byte(0xE3);
final byte KEY_CONTROL_RIGHT      =byte(0xE4);
final byte KEY_SHIFT_RIGHT       =byte(0xE5);
final byte KEY_ALT_RIGHT         =byte(0xE6);
final byte KEY_GUI_RIGHT         =byte(0xE7);
```

Here is the Arduino function we use to send a raw **keyCommand**. For example, if you want to send the keystroke for the letter 'a' (no shift) you'll want to call

```
keyCommand(0, 4);
```

to press the keycode 4 ('a') followed by a release

```
keyCommand(0, 0);
```

if you want to send the keystroke for SHIFT 'a' you'll want to call

```
keyCommand(MODIFIER_SHIFT_LEFT, 4);
```

if you want to send the keystroke for CTRL-SHIFT 'a' you'll want to call

```
keyCommand(MODIFIER_SHIFT_LEFT | MODIFIER_CONTROL_LEFT, 4);
```

You can also send multiple keystrokes ('chords'). If you want to press 'a' and 'b' at the same time, send

```
    keyCommand(0, 4, 5);
```

for keycodes 4 and 5 at the same time. You can send up to 6 consecutive keys at once, don't forget to send the release 'key up' command or the key will be 'stuck'!

```
void keyCommand(uint8_t modifiers, uint8_t keycode1, uint8_t keycode2 = 0, uint8_t keycode3 = 0,
                uint8_t keycode4 = 0, uint8_t keycode5 = 0, uint8_t keycode6 = 0) {
  BT.write(0xFD);       // our command
  BT.write(modifiers);  // modifier!
  BT.write((byte)0x00); // 0x00
  BT.write(keycode1);   // key code #1
  BT.write(keycode2); // key code #2
  BT.write(keycode3); // key code #3
  BT.write(keycode4); // key code #4
  BT.write(keycode5); // key code #5
  BT.write(keycode6); // key code #6
}
```

Raw HID Mouse Reports

As of v1.1 (shipping Oct 22, 2013) Bluefruit can also send raw HID Mouse reports. This allows moving and clicking a virtual mouse! Mouse reports are *relative* movement. So you can send 'go left 4 units' but you cant send 'go to absolute location x, y'

Raw HID reports start with **0xFD** and have 8 bytes following. For mouse, its
0xFD 0x00 0x03 [buttons] [left/right] [up/down] 0x0 0x0 0x0

For buttons, its a bitmask, left button (button 0) is**0x01** right button (button 1) is**0x02**, etc so that **button n** is **(1 << n)** you can | these together

up/down/left/right are again, relative movements. You can move up to +127 up/left to -127 down/right units at a time.

```
void mouseCommand(uint8_t buttons, uint8_t x, uint8_t y) {
  BT.write(0xFD);
  BT.write((byte)0x00);
  BT.write((byte)0x03);
  BT.write(buttons);
  BT.write(x);
  BT.write(y);
  BT.write((byte)0x00);
  BT.write((byte)0x00);
  BT.write((byte)0x00);
}
```

For example if we wanted to click the left button and drag the mouse down 50 units send

mouseCommand(0x1, 0, -50);

Raw HID Consumer Reports

As of v1.2, Bluefruit can send raw HID consumer reports. There are "Home", "KeyboardLayout", "Search", "Snapshot", "VolumeUp", "VolumeDown", "Play/Pause", "Fast Forward", "Rewind","Scan Next Track", "Scan Previous Track", "Random Play","Stop" keys you can use with a 2 bytes bitmask.

Raw HID consumer report start with **0xFD** and have 8 bytes following. For consumer keys, its
0xFD 0x00 0x02 [bitmask] [bitmask] 0x0 0x0 0x0 0x0

"Home" is bit 0, the bitmask is 0x01 0x00
"Stop" is bit 12, the bitmask is 0x00 0x10

You can | these together like mouse report

```
void consumerCommand(uint8_t mask0,uint8_t mask1) {
  BT.write(0xFD);
  BT.write((byte)0x00);
  BT.write((byte)0x02);
  BT.write(mask0);
  BT.write(mask1);
  BT.write((byte)0x00);
  BT.write((byte)0x00);
  BT.write((byte)0x00);
  BT.write((byte)0x00);
}
```

For example if we wanted to click the Play/Pause

consumerCommand**(0x40,0x00);**

Then release it

consumerCommand**(0x00,0x00);**

Testing Sketch (Arduino)

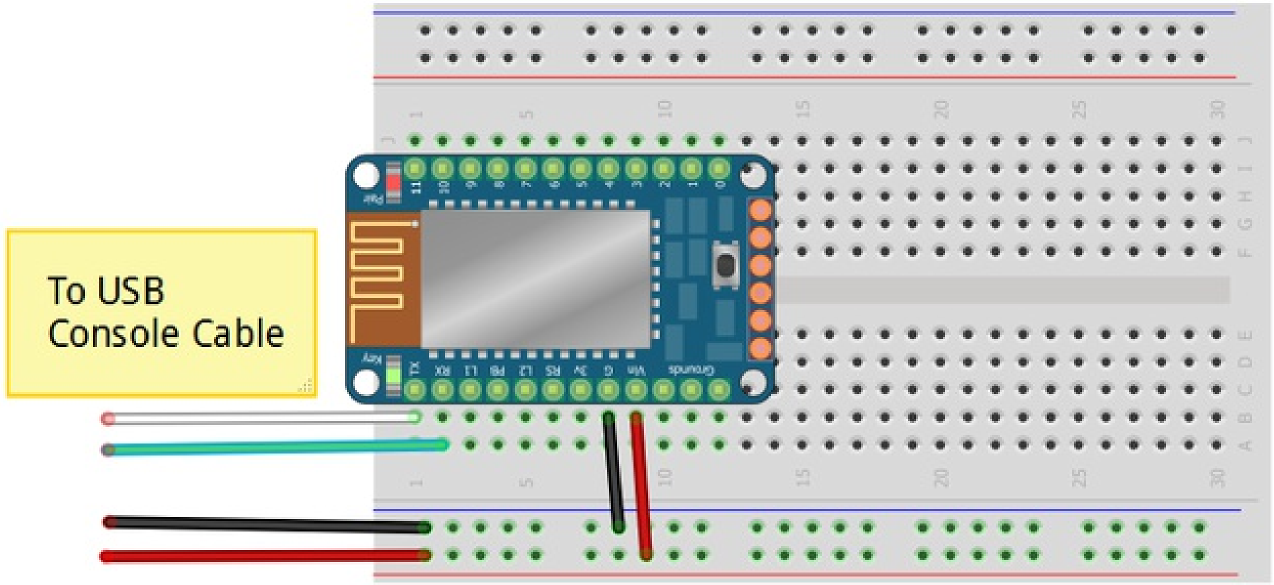
We use this code to generate/test the various UART-sendable characters, you can use it for reference to control via an Arduino or other microcontroller. Remember that some of these non-printing characters can really confuse your computer so use with care!

```
// Adafruit test code for Bluefruit EZ-Key serial reports
// Uncomment tests as you wish, remember that this will
// send various keypresses to your computer which may really
// annoy it! We used
// http://www.cambiaresearch.com/articles/15/javascript-char-codes-key-codes
// to test the non-printing characters!
// Connect the RX pin on the EZ-Key to digital #2 on the UNO

#include <SoftwareSerial.h>

SoftwareSerial BT = SoftwareSerial(3, 2);

void printabletest() {
  Serial.println("Testing printable 0x20-0x7E...");
  for (char c = 0x20; c <= 0x7E; c++) {
    Serial.write(c);
    BT.write(c);
    delay(10);
  }

  BT.write('\n');
  delay(3000);
  Serial.read(); // eat one char
  Serial.println();

  for (uint16_t i=0; i<200; i++) {
    while (Serial.available())
      Serial.write(Serial.read());
    delay(10);
  }
}

void nonprinting() {
  Serial.println("Insert");
  BT.write(1); delay(1000);
  Serial.println("Home");
  BT.write(2); delay(1000);
  Serial.println("Page Up");
  BT.write(3); delay(1000);
  Serial.println("Delete");
  BT.write(4); delay(1000);
  Serial.println("End");
  BT.write(5); delay(1000);
  Serial.println("Page Down");
  BT.write(6); delay(1000);
  Serial.println("Right Arrow");
  BT.write(7); delay(1000);
  Serial.println("Backspace");
  BT.write(8); delay(1000);
  Serial.println("Tab");
  BT.write(9); delay(1000);
  Serial.println("Enter");
  BT.write(10); delay(1000);
  Serial.println("Left Arrow");
  BT.write(11); delay(1000);
  Serial.println("Down Arrow");
  BT.write(12); delay(1000);
  Serial.println("Enter");
  BT.write(13); delay(1000);
  Serial.println("Up Arrow");
  BT.write(14); delay(1000);

  for (uint8_t i=15; i<27; i++) {
    Serial.print("F"); Serial.println(i-14, DEC);
     BT.write(i); delay(500);
  }
  Serial.println("ESC");
  BT.write(27); delay(1000);
  Serial.println("Capslock");
  BT.write(28); delay(1000);
  Serial.println("Scroll lock");
  BT.write(29); delay(1000);
  Serial.println("Break");
  BT.write(30); delay(1000);
  Serial.println("Numlock");
  BT.write(31); delay(500);

}
void altkeystest() {
  Serial.println("Left Control\n");
  BT.write(0xE0);
  delay(500);
  Serial.println("Left Shift\n");
  BT.write(0xE1);
  delay(500);
  Serial.println("Left Alt\n");
  BT.write(0xE2);
  delay(500);
  Serial.println("Left GUI\n");
  BT.write(0xE3);
  delay(500);
  Serial.println("Right Control\n");
  BT.write(0xE4);
  delay(500);
  Serial.println("Right Shift\n");
  BT.write(0xE5);
  delay(500);
  Serial.println("Right Alt\n");
  BT.write(0xE6);
  delay(500);
  Serial.println("Right GUI\n");
  BT.write(0xE7);
  delay(500);
}

void iphonekeyboard() {
  while (1) {
    Serial.println("toggle keyboard");
    BT.write(0x7F);
    delay(1000);
  }
}

void keyCommand(uint8_t modifiers, uint8_t keycode1, uint8_t keycode2 = 0, uint8_t keycode3 = 0,
                uint8_t keycode4 = 0, uint8_t keycode5 = 0, uint8_t keycode6 = 0) {
  BT.write(0xFD);       // our command
  BT.write(modifiers);  // modifier!
  BT.write((byte)0x00); // 0x00
  BT.write(keycode1);   // key code #1
  BT.write(keycode2); // key code #2
  BT.write(keycode3); // key code #3
  BT.write(keycode4); // key code #4
  BT.write(keycode5); // key code #5
  BT.write(keycode6); // key code #6
}

void rawkeytest() {
  // test sending a single 'a' (keycode 4)
  keyCommand(0, 4);
  delay(100);
  keyCommand(0, 0);
}

/************** Support added in v1.1 */

void rawmousetest() {
  Serial.println("Move mouse!");
  Serial.println("Right");
  mouseCommand(0, -100, 0);
  delay(200);
  Serial.println("Down");
  mouseCommand(0, 0, -100);
  delay(200);
  Serial.println("Left");
  mouseCommand(0, 100, 0);
  delay(200);
  Serial.println("Up");
  mouseCommand(0, 0, 100);
  delay(200);
}

void mouseCommand(uint8_t buttons, uint8_t x, uint8_t y) {
  BT.write(0xFD);
  BT.write((byte)0x00);
  BT.write((byte)0x03);
  BT.write(buttons);
  BT.write(x);
  BT.write(y);
  BT.write((byte)0x00);
  BT.write((byte)0x00);
  BT.write((byte)0x00);
}

/************** Support added in v1.1 */

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

  Serial.println("Softserial/BT test!\n\rPress any key+return to start");
  while (! Serial.available());

  printabletest(); delay(200);
  //altkeystest();
  //nonprinting();
  //iphonekeyboard();
  //rawkeytest();

  //Supported in v1.1
  //rawmousetest();
}

void loop() {

}
```

Remapping the Buttons (Serial)

If you have a Bluefruit v1.1 or later, you can wirelessly re-map the keys, no need to use a console cable (altho the console cable technique will still work). Original v1.0 Bluefruits can only be remapped over serial console

The default buttons->keypresses will be satisfactory for most projects. However, you may want to customize those keys so when a GPIO pin is pulled to ground, a different keystroke is sent. It isn't that difficult to do! However, you will need an FTDI Friend (http://adafru.it/284) or USB console cable (http://adafru.it/954) to connect the EZ-Key to your computer.



If using an FTDI friend, connect the TX pin of the EZ-Key to the RX pin on your FTDI friend and the RX pin to the TX pin (receiver goes to transmit). Also be sure to power the EZ-Key. You do not have to pair it for this remapping

You will have to put the EZ-Key into re-mapping mode. To do this, the pair-button must be pressed while the module is powered up. The easiest way to do this is to disconnect the red Vin wire from the console cable, press down on the button, then plug in the red wire, that's it!
As a 'safety' procedure, every time you power the module with the re-pair button pressed (to enter remapping mode) it will reload the default keymap. This is so if you somehow really mess things up, you can always get back to the default keymap without a console cable

Load Processing Sketch

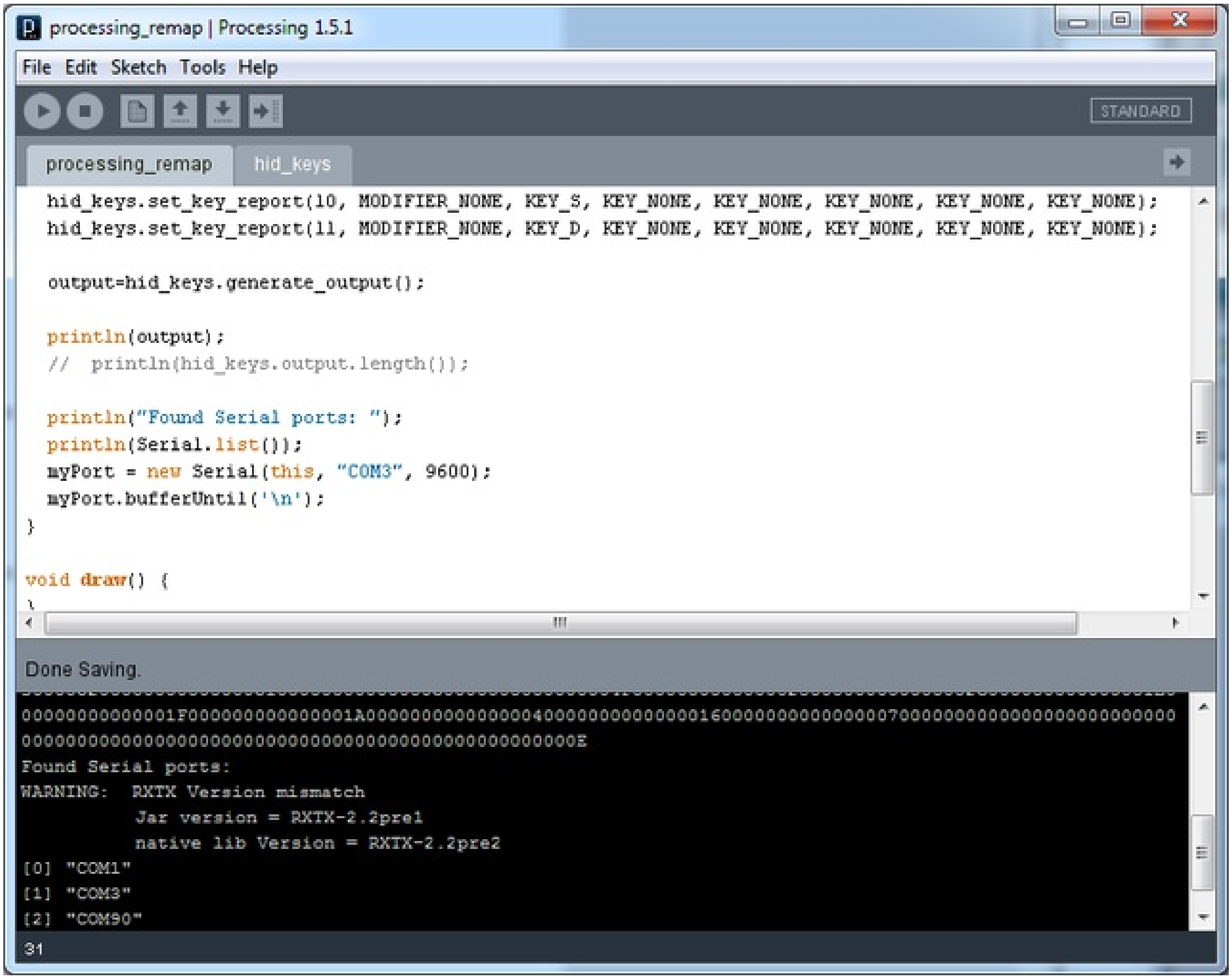
We wrote the re-mapping software in Processing.Processing is cross-platform and easy to install (http://adafru.it/kA6). Please download Processing v1.5.1 (http://adafru.it/cK1) since that's known to work

You'll also need to download and install the ControlP5 library

Click to download the ControlP5 library
http://adafru.it/djs

Then click below to download the remap software

EZ-Key_Remapper.zip
http://adafru.it/cK2
Uncompress and open the remapper.pde in Processing. Plug in the USB console cable or FTDI adapter. Select **Sketch->Run** menu item and look in the debug window below, you should see **Found Serial Ports:** and then a list of ports. On Windows, it will be something like **COM1, COM2, COM3** etc. On Mac/Linux it will be something like/**dev/USBtty** or **/dev/cu.usbserial**


processing_remap | Processing 1.5.1
File Edit Sketch Tools Help
STANDARD
processing_remap
hid_keys
Done Saving.
31

```
  hid_keys.set_key_report(10, MODIFIER_NONE, KEY_S, KEY_NONE, KEY_NONE, KEY_NONE, KEY_NONE, KEY_NONE);
  hid_keys.set_key_report(11, MODIFIER_NONE, KEY_D, KEY_NONE, KEY_NONE, KEY_NONE, KEY_NONE, KEY_NONE);

  output=hid_keys.generate_output();

  println(output);
  //  println(hid_keys.output.length());

  println("Found Serial ports: ");
  println(Serial.list());
  myPort = new Serial(this, "COM3", 9600);
  myPort.bufferUntil('\n');
}

void draw() {
```

```
00000000000001F000000000000001A0000000000000004000000000000001600000000000000070000000000000000000000000
00000000000000000000000000000000000000000000000000E
Found Serial ports:
WARNING:  RXTX Version mismatch
          Jar version = RXTX-2.2pre1
          native lib Version = RXTX-2.2pre2
[0] "COM1"
[1] "COM3"
[2] "COM90"
```

Copy and paste the list into a notepad, now **File->Run** it again, this time with the USB console cable unplugged. The list should be one line shorter. In this case **COM3** is missing. That means that the cable name is **COM3**
in the line

myPort = new Serial(this, "COM3", 9600);

Change "COM3" to whatever the cable name is. Again, for Mac or Linux it will probably be **/dev/cu.something**

Replug in the FTDI/Console cable.

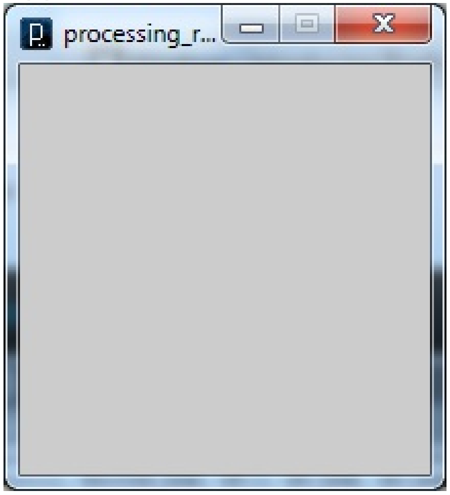
Select **File->Run** again to start the script with the correct /dev or COM port

Now disconnect the red wire that is powering the EZ-Key, press and hold the Pair button on the module and reconnect the red wire. You should see:

Adafruit Bluefruit HID 9/18/2013

Remap ready!

Appear in the bottom half of the screen. Now click that large gray square window to remap

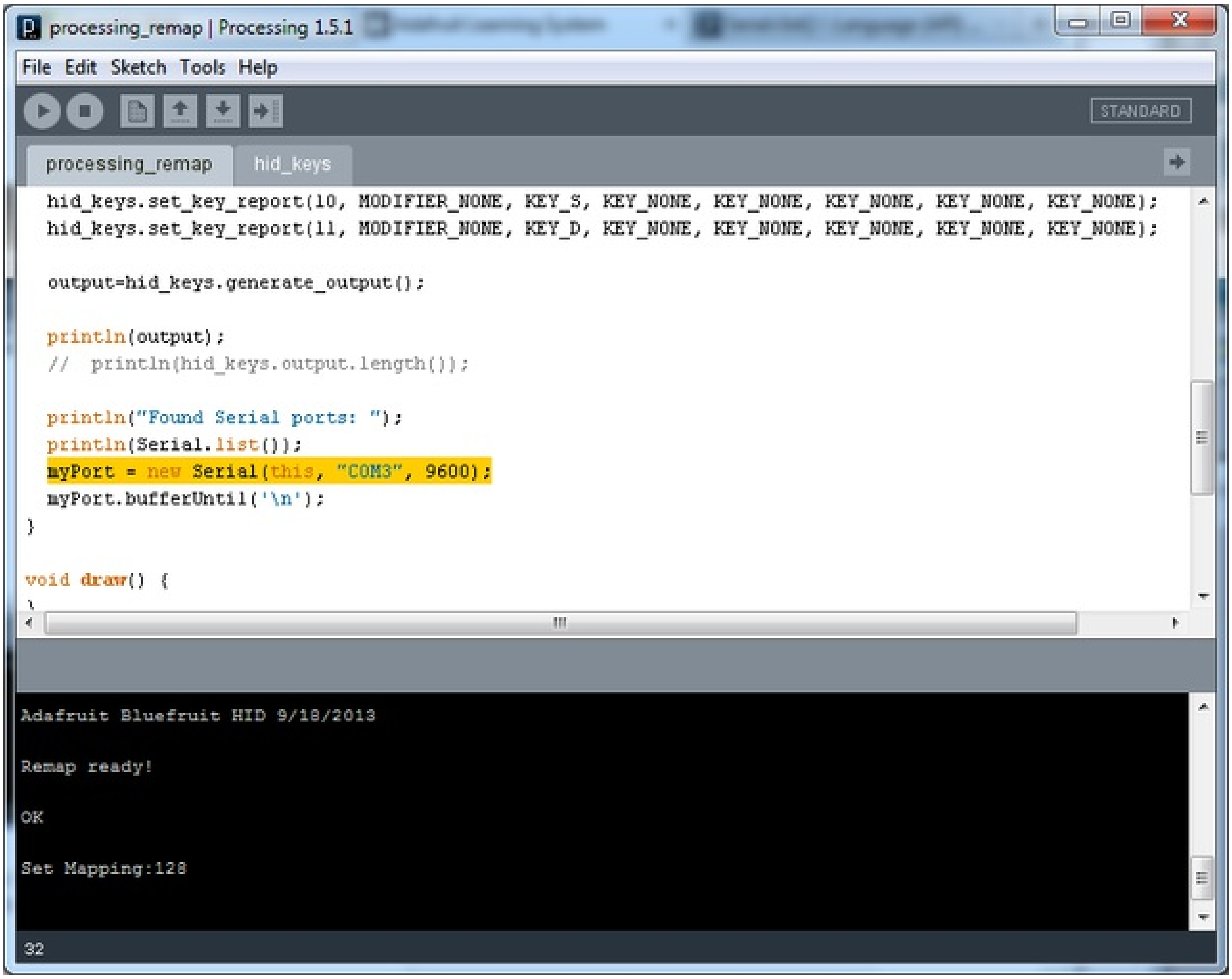


You should see

OK

Set Mapping:128

Indicating that the mapping was sent.


processing_remap | Processing 1.5.1
File Edit Sketch Tools Help
STANDARD
processing_remap
hid_keys
32

```
  hid_keys.set_key_report(10, MODIFIER_NONE, KEY_S, KEY_NONE, KEY_NONE, KEY_NONE, KEY_NONE, KEY_NONE);
  hid_keys.set_key_report(11, MODIFIER_NONE, KEY_D, KEY_NONE, KEY_NONE, KEY_NONE, KEY_NONE, KEY_NONE);

  output=hid_keys.generate_output();

  println(output);
  //  println(hid_keys.output.length());

  println("Found Serial ports: ");
  println(Serial.list());
  myPort = new Serial(this, "COM3", 9600);
  myPort.bufferUntil('\n');
}

void draw() {
}
```

```
Adafruit Bluefruit HID 9/18/2013

Remap ready!

OK

Set Mapping:128
```

Customizing Keys

OK now you want to actually change the key report. Each of the 12 keys has a report. This is an example for #0:

hid_keys.set_key_report(0, MODIFIER_NONE, KEY_A, KEY_NONE, KEY_NONE, KEY_NONE, KEY_NONE, KEY_NONE);

There are 8 'arguments' to each report:

The first one is the GPIO#, in this case its**#0**.

Second is the modifier keys, such as Shift, Control, Alt, etc. See**hid_keys.pde** for a list of the modifier available. You can 'or' the modifiers. For example, if you want to press shift and control modifiers at the same time, use **MODIFIER_SHIFT_LEFT | MODIFIER_CONTROL_LEFT**

The last 6 are the 6 slots available for concurrent keys. You can have up to 6 key-codes sent at once (handy for when you want to send complex key reports. Check **hid_keys.pde** for the list of all the keycodes!

Remapping the Buttons (Wireless)

The default buttons->keypresses will be satisfactory for most projects. However, you may want to customize those keys so when a GPIO pin is pulled to ground, a different keystroke is sent. It isn't that difficult to do!

Bluefruit v1.1 (Oct 22, 2013 or later) supports over-the-air key remapping**and** also you can map the pins to mouse movement or mouse clicks

Bluefruit v1.2 (Nov 7 2013 or later) - We have added support for mapping buttons to some "Consumer Report" keys, also known as Multimedia buttons. You can use the graphical remapper for the consumer keys.

Over-the-air remapping is suggested since there's no wiring required and you can do it at any time. You just need to make sure the Bluefruit is paired to your computer. It's been tested on Mac and Windows but should also work on Linux.

You'll need to install Processing v1.5.1 to run the remapper. (http://adafru.it/cK1)We suggest 1.5.1, 2.0+ may not work. If v1.5 is giving you problems, though, try v2!

You will also need to download and install the ControlP5 library:

In Windows, you should use the 32-bit version of Processing!
Download the ControlP5 LIbrary
http://adafru.it/djs

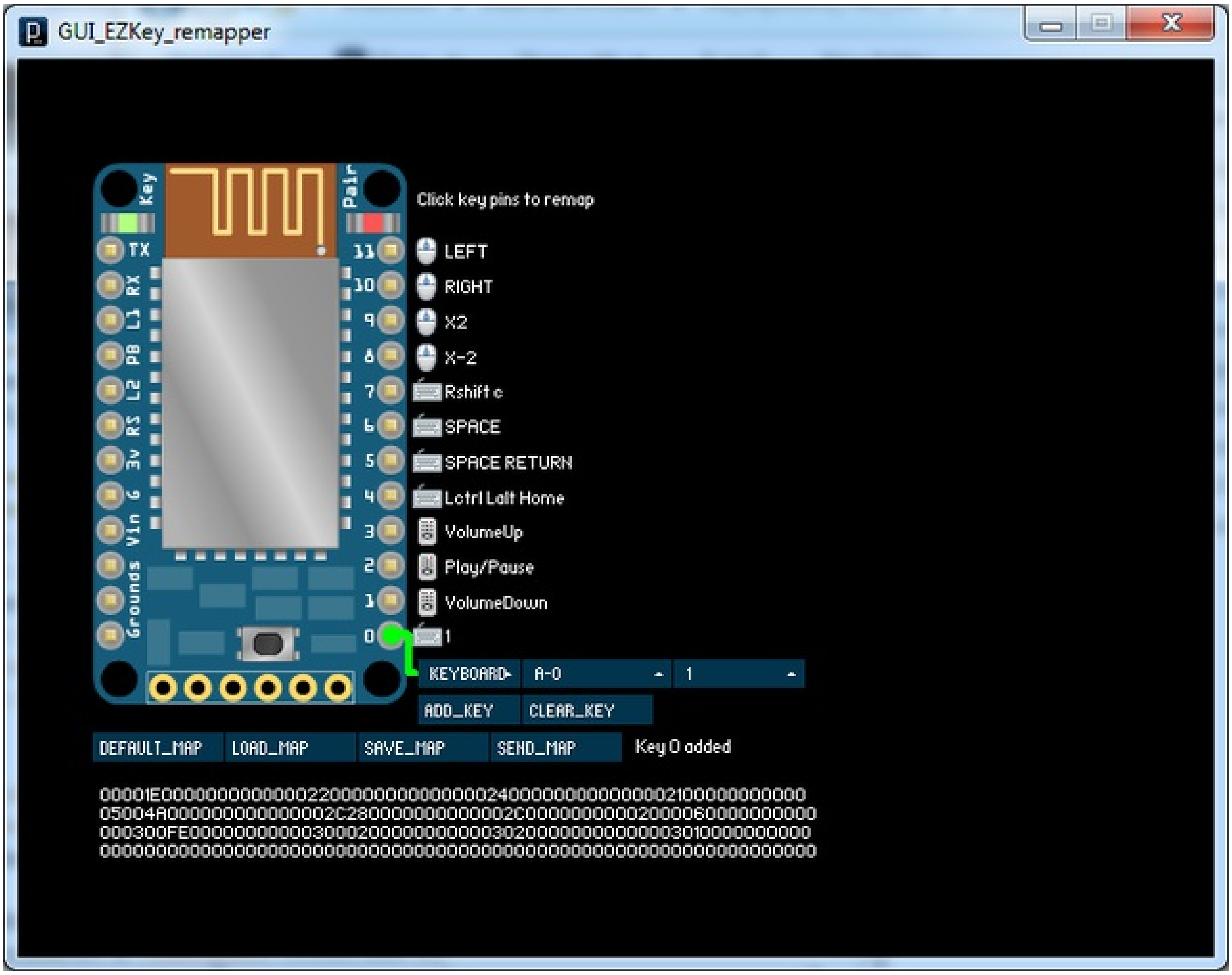
Then download the zip with the wireless remapper code

GUI EZ-Key Remapper 12/20/2013
http://adafru.it/d2B
Uncompress and open the **GUI_EZKey_remapper.pde** in Processing.
Make sure the Bluefruit is paired to your computer, since we need to be paired in order to send it the new keycodes

Select **Sketch -> Run** or click the play button to start the graphical remapper

You can now select the pins and then using the menus below select what mouse/keyboard/consumer reports you want. Mouse reports can be X movement, Y movement, wheel, and three different buttons in any combination. Keyboard reports can be up to 6 keystroke commands, with modifiers, ascii-type, non-printing, etc. Consumer reports are special commands, such as volume up, volume down, iPhone keyboard display/hide and cannot be mixed or combined unlike mouse & Keyboard.

You can also save and load your keymap which is handy for trying out different mappings! If you get the error "the function setindex(int) does not exist", try using Processing version 2.2.1 and ControlP5 version 2.0.4



Key 0 added

```
00001E000000000000002200000000000000240000000000000021000000000000
05004A000000000000002C28000000000000002C00000000000002000600000000
000300FE000000000003000200000000000302000000000000030100000000000
0000000000000000000000000000000000000000000000000000000000000000
```

When you're done, click **SEND_MAP** to send it to your Bluefruit. In the main Processing window you should see the text report indicate that it found a Bluefruit and also that it sent data with a **Checksum Match**

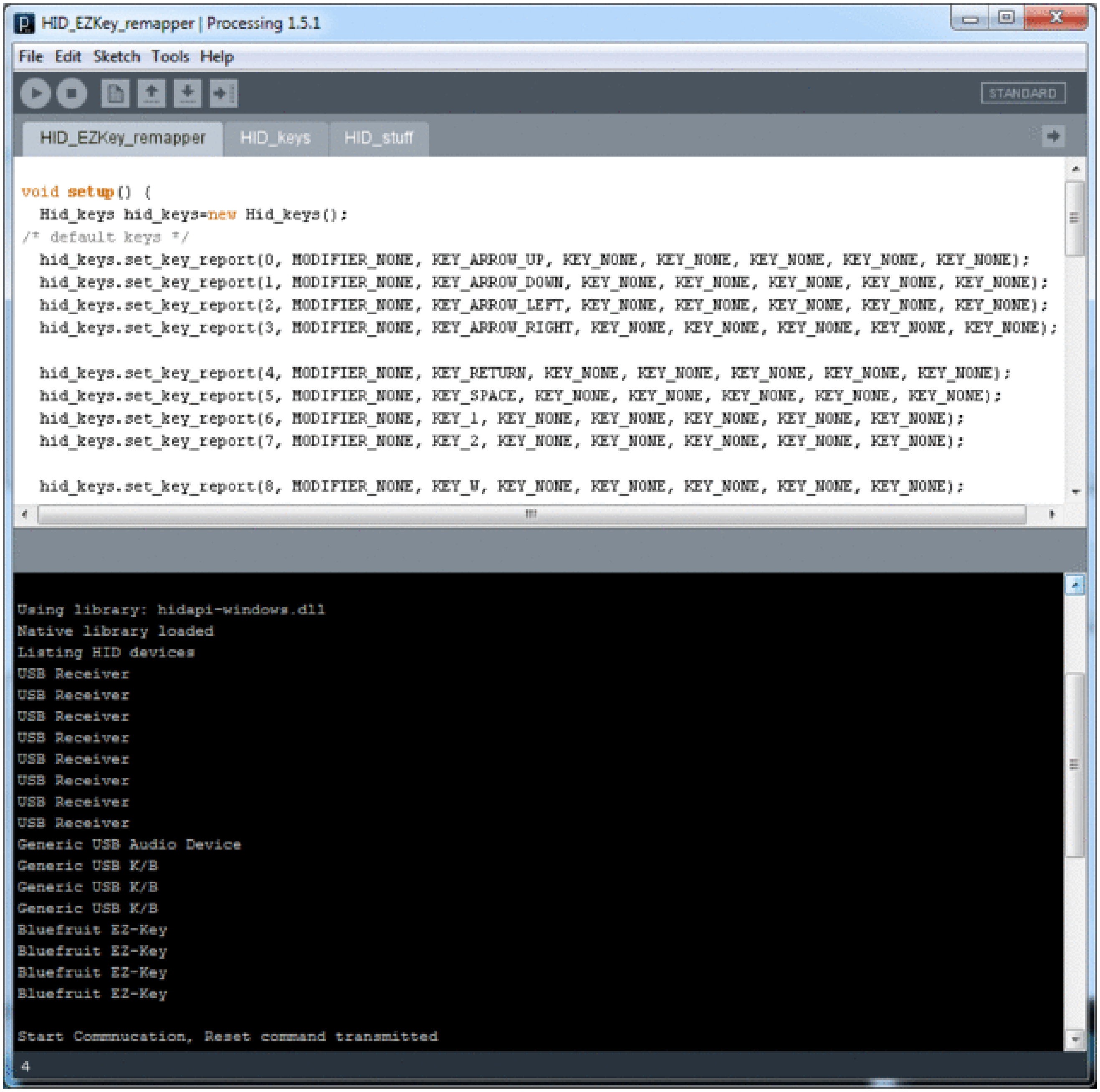
"Text style" over-the-air remapper

The original remapper is a Processing sketch that requires editing the reports by hand. While this is the most powerful, most people may be happier with the graphical remapper above. Still, for more advanced users, you can edit the reports per button with more

flexibility here.
HID_EZKey_remapper.zip
http://adafru.it/cQX
Uncompress and open the **HID_EZKey_remapper.pde** in Processing.
Make sure the Bluefruit is paired to your computer, since we need to be paired in order to send it the new keycodes

Select **Sketch -> Run** or click the play button

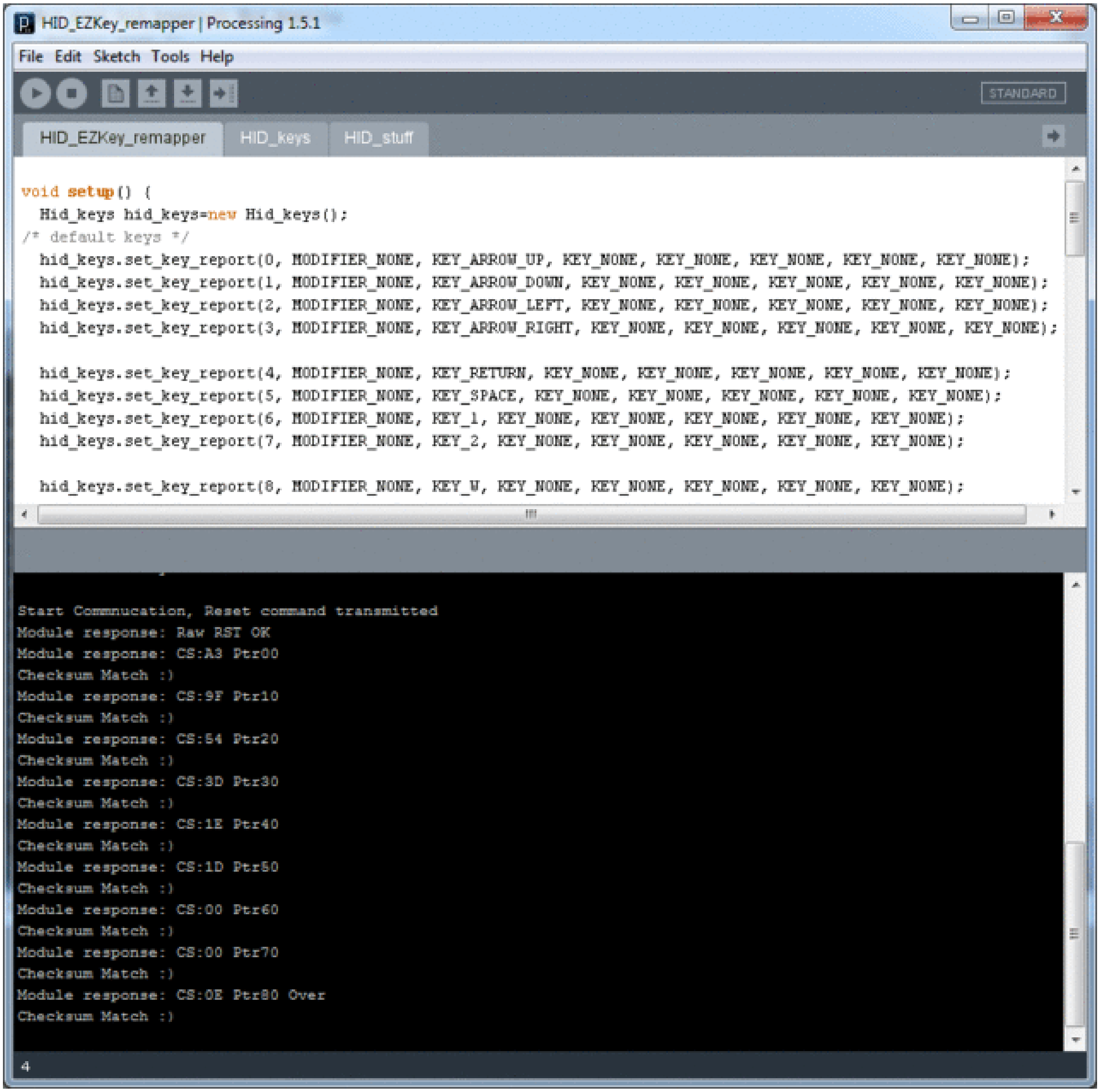
You should see the text report indicate that it found a Bluefruit and also that it sent data with a **Checksum Match**


HID_EZKey_remapper | Processing 1.5.1
File Edit Sketch Tools Help
STANDARD
HID_EZKey_remapper
HID_keys
HID_stuff

```
void setup() {
  Hid_keys hid_keys=new Hid_keys();
/* default keys */
  hid_keys.set_key_report(0, MODIFIER_NONE, KEY_ARROW_UP, KEY_NONE, KEY_NONE, KEY_NONE, KEY_NONE, KEY_NONE);
  hid_keys.set_key_report(1, MODIFIER_NONE, KEY_ARROW_DOWN, KEY_NONE, KEY_NONE, KEY_NONE, KEY_NONE, KEY_NONE);
  hid_keys.set_key_report(2, MODIFIER_NONE, KEY_ARROW_LEFT, KEY_NONE, KEY_NONE, KEY_NONE, KEY_NONE, KEY_NONE);
  hid_keys.set_key_report(3, MODIFIER_NONE, KEY_ARROW_RIGHT, KEY_NONE, KEY_NONE, KEY_NONE, KEY_NONE, KEY_NONE);

  hid_keys.set_key_report(4, MODIFIER_NONE, KEY_RETURN, KEY_NONE, KEY_NONE, KEY_NONE, KEY_NONE, KEY_NONE);
  hid_keys.set_key_report(5, MODIFIER_NONE, KEY_SPACE, KEY_NONE, KEY_NONE, KEY_NONE, KEY_NONE, KEY_NONE);
  hid_keys.set_key_report(6, MODIFIER_NONE, KEY_1, KEY_NONE, KEY_NONE, KEY_NONE, KEY_NONE, KEY_NONE);
  hid_keys.set_key_report(7, MODIFIER_NONE, KEY_2, KEY_NONE, KEY_NONE, KEY_NONE, KEY_NONE, KEY_NONE);

  hid_keys.set_key_report(8, MODIFIER_NONE, KEY_W, KEY_NONE, KEY_NONE, KEY_NONE, KEY_NONE, KEY_NONE);
```

```
Using library: hidapi-windows.dll
Native library loaded
Listing HID devices
USB Receiver
USB Receiver
USB Receiver
USB Receiver
USB Receiver
USB Receiver
USB Receiver
USB Receiver
Generic USB Audio Device
Generic USB K/B
Generic USB K/B
Generic USB K/B
Bluefruit EZ-Key
Bluefruit EZ-Key
Bluefruit EZ-Key
Bluefruit EZ-Key

Start Commnucation, Reset command transmitted
```

4


HID_EZKey_remapper | Processing 1.5.1
File Edit Sketch Tools Help
STANDARD
HID_EZKey_remapper
HID_keys
HID_stuff
4

```
void setup() {
  Hid_keys hid_keys=new Hid_keys();
/* default keys */
  hid_keys.set_key_report(0, MODIFIER_NONE, KEY_ARROW_UP, KEY_NONE, KEY_NONE, KEY_NONE, KEY_NONE, KEY_NONE);
  hid_keys.set_key_report(1, MODIFIER_NONE, KEY_ARROW_DOWN, KEY_NONE, KEY_NONE, KEY_NONE, KEY_NONE, KEY_NONE);
  hid_keys.set_key_report(2, MODIFIER_NONE, KEY_ARROW_LEFT, KEY_NONE, KEY_NONE, KEY_NONE, KEY_NONE, KEY_NONE);
  hid_keys.set_key_report(3, MODIFIER_NONE, KEY_ARROW_RIGHT, KEY_NONE, KEY_NONE, KEY_NONE, KEY_NONE, KEY_NONE);

  hid_keys.set_key_report(4, MODIFIER_NONE, KEY_RETURN, KEY_NONE, KEY_NONE, KEY_NONE, KEY_NONE, KEY_NONE);
  hid_keys.set_key_report(5, MODIFIER_NONE, KEY_SPACE, KEY_NONE, KEY_NONE, KEY_NONE, KEY_NONE, KEY_NONE);
  hid_keys.set_key_report(6, MODIFIER_NONE, KEY_1, KEY_NONE, KEY_NONE, KEY_NONE, KEY_NONE, KEY_NONE);
  hid_keys.set_key_report(7, MODIFIER_NONE, KEY_2, KEY_NONE, KEY_NONE, KEY_NONE, KEY_NONE, KEY_NONE);

  hid_keys.set_key_report(8, MODIFIER_NONE, KEY_W, KEY_NONE, KEY_NONE, KEY_NONE, KEY_NONE, KEY_NONE);
```

```
Start Commnucation, Reset command transmitted
Module response: Raw RST OK
Module response: CS:A3 Ptr00
Checksum Match :)
Module response: CS:9F Ptr10
Checksum Match :)
Module response: CS:54 Ptr20
Checksum Match :)
Module response: CS:3D Ptr30
Checksum Match :)
Module response: CS:1E Ptr40
Checksum Match :)
Module response: CS:1D Ptr50
Checksum Match :)
Module response: CS:00 Ptr60
Checksum Match :)
Module response: CS:00 Ptr70
Checksum Match :)
Module response: CS:0E Ptr80 Over
Checksum Match :)
```

That's it, you can use it immediately with the new keymap!

FAQ

What is the current draw for Bluefruit?

Bluefruit requires ~3VDC, there's an onboard regulator for providing nice clean power. It draws 25mA at all times while paired and an extra 2mA on average during transmission

When the module is in Reset mode (Reset tied low) it draws 2-3mA quiescent.

What is the latency ("fly time") for the 12 GPIO buttons?

For Bluefruit v1.0, latency is about 100-120ms. For v1.1 and later (October 22, 2013) latency is 25-35ms. There is some 'jitter' in the latency but we find its not that noticeable.

Is it possible to firmware update Bluefruits?

There is no way to re-program or update Bluefruit modules internal firmware - the key remapping is changing the EEPROM data, not the program storage.

How come the EZ-Key can be sluggish on Android?

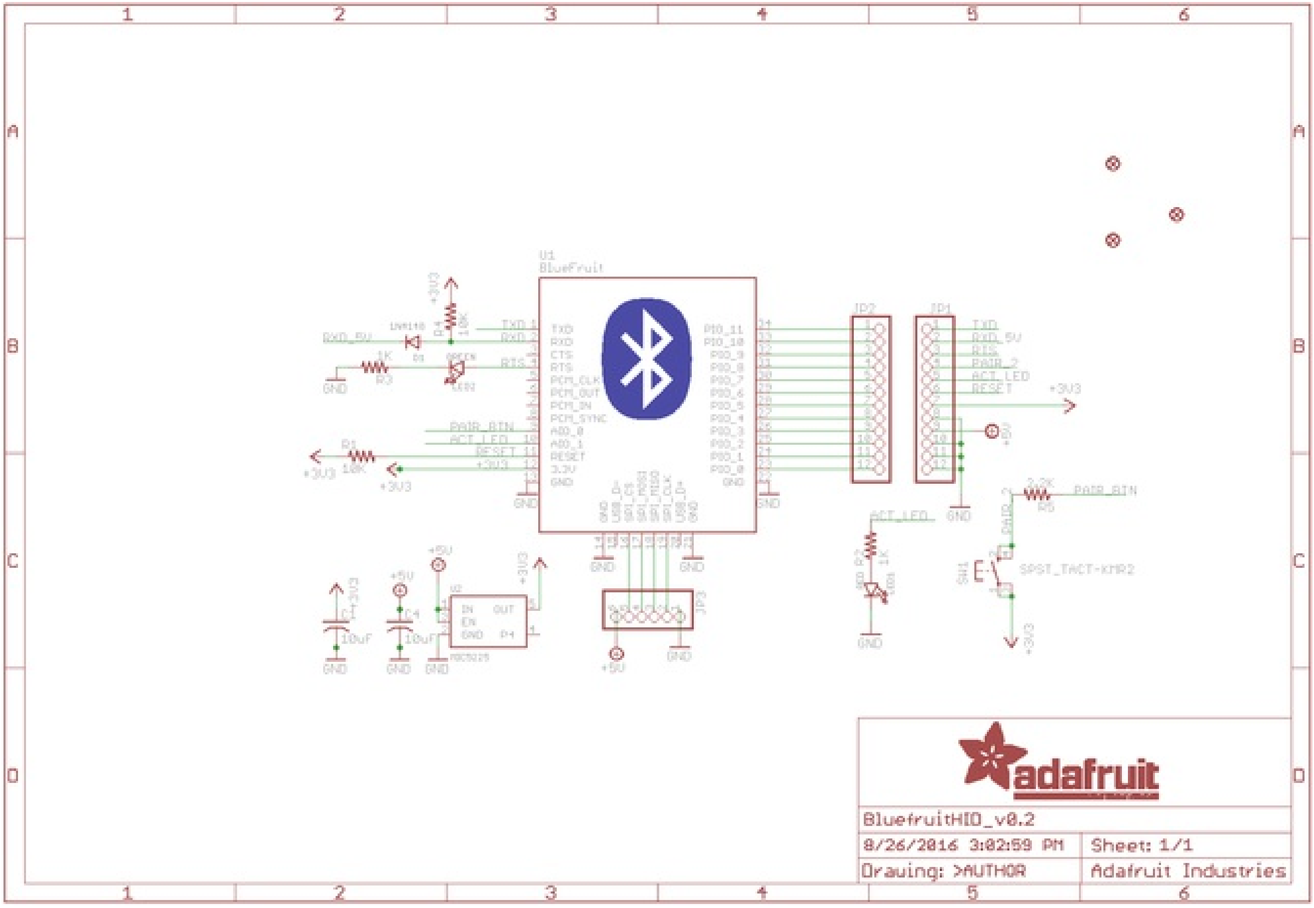
There's some issues with Android 4.1 thru 4.4 that cause Bluetooth to act odd due to WiFi interference. Try turning off WiFi to see if that helps. [check this post for details! (http://adafru.it/djF)]

Downloads

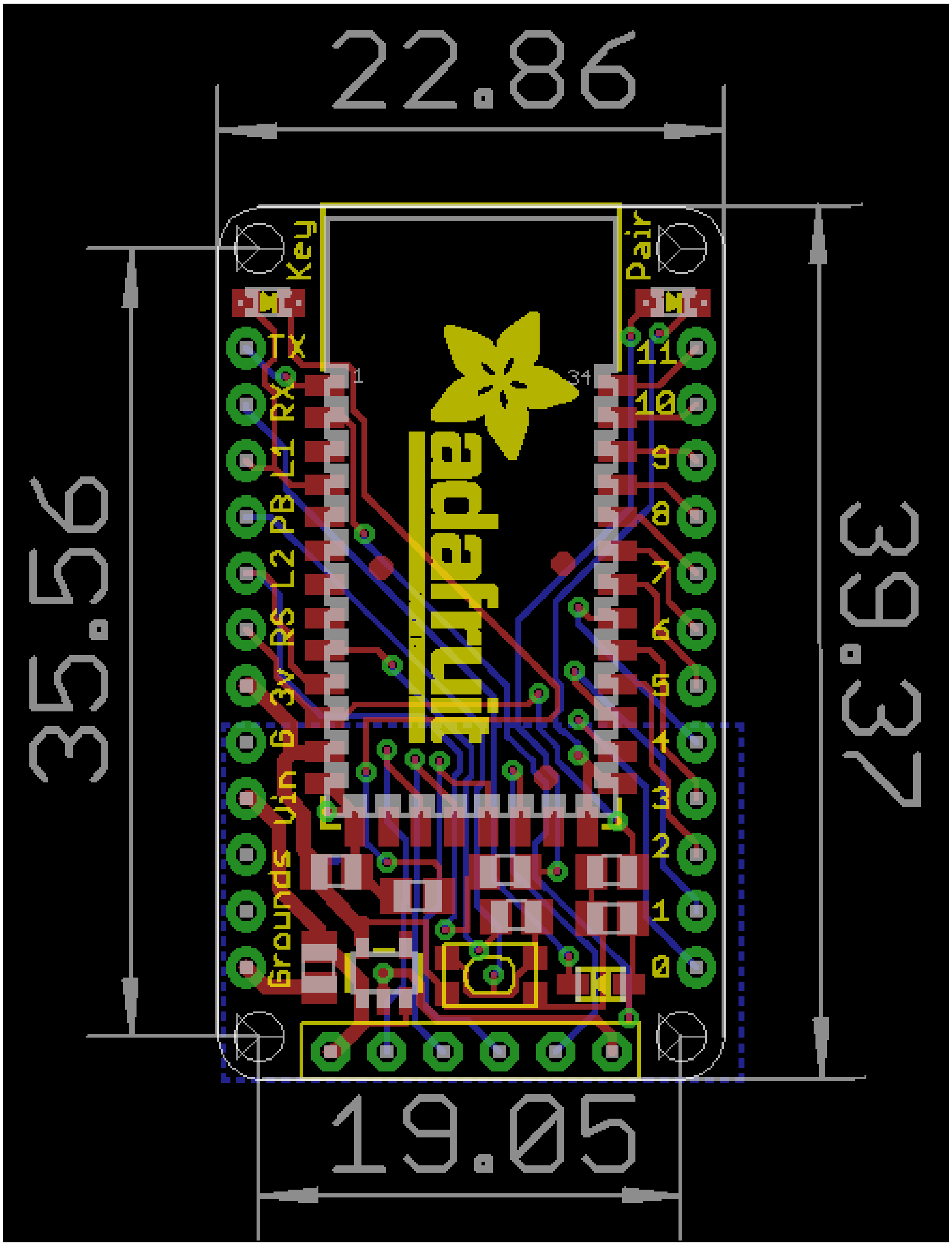
Files

- [Fritzing object in Adafruit Fritzing library](http://adafru.it/c7M) (http://adafru.it/c7M)
- [EagleCAD PCB files on GitHub](http://adafru.it/rAu) (http://adafru.it/rAu)

Schematic



Fabrication Print


22.86
35.56
39.37
19.05
Key
Pair
TX
RX
L1
PB
L2
RS
3v
G
Vin
Grounds
11
10
9
8
7
6
5
4
3
2
1
0
1
34
adafruit