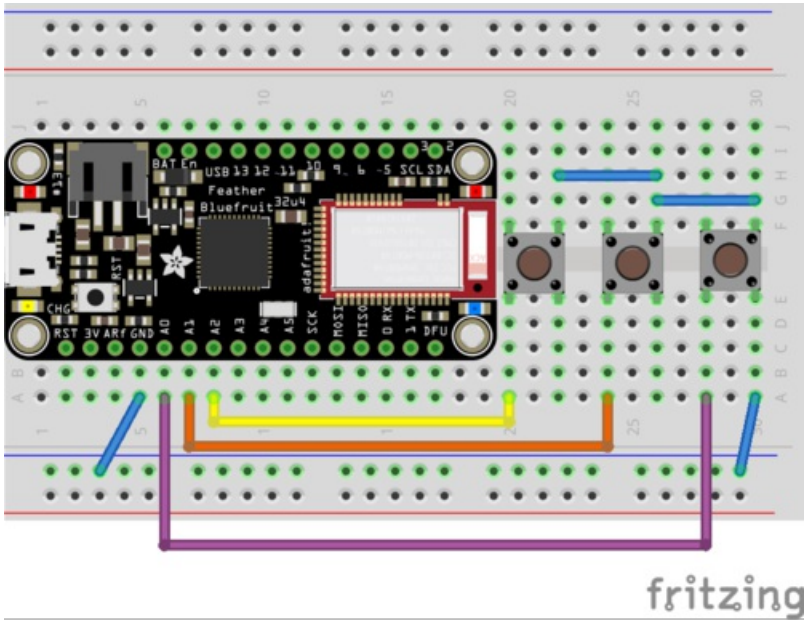
adafruit learning system

Using Bluefruit BLE to Give Disabled Users Switch Control Access to IOS Devices

Created by Chris Young



Last updated on 2018-08-22 03:58:42 PM UTC

Guide Contents

Overview

The touchscreen interface of modern smart phones and tablets is probably the easiest and most intuitive interface to a computing device that has ever been invented. However if you have a physical disability that prohibits you from operating a touchscreen, a world of useful applications that is at the fingertips of most people may be impossible for you to use. This is where adaptive technology (AT) comes in.

A special feature built into the iOS operating system that allows you almost complete access to all of the device's capabilities via a feature called "switch control". If you can operate at least one switch, it can be connected to a Bluetooth Low Energy (BLE) device and you can scan through rows and columns of icons and select the application you want to run. Once inside the app you can tap anywhere on the screen, scan through options to be selected, or type a message. The YouTube video below demonstrates how switch control works. This video was created on an iPad Mini 2 using the techniques described in this tutorial. It will work on any device running iOS 9 or greater.

Complete details on how to use switch control accessibility from iOS can be found on the Apple website at https://support.apple.com/en-us/HT201370 (https://adafru.it/udR)

There are some additional tutorial videos and other resources on the last page of this tutorial under "Final Thoughts" that may be of interest to you.

Commercially made Bluetooth switches can be purchased from a variety of companies who specialize in assistive technology but the prices can run from $180 up to $500 or more. We can use our maker skills, the power of Arduino compatible processors, and a few parts from the Adafruit store and build our own custom device for a fraction of that cost. Not only can a maker-built device save considerable money, if you have access to a 3D printer, laser cutter, or CNC devices you can create a device especially suited to the particular needs of the user. You have the flexibility that may not be available from a commercially built device.

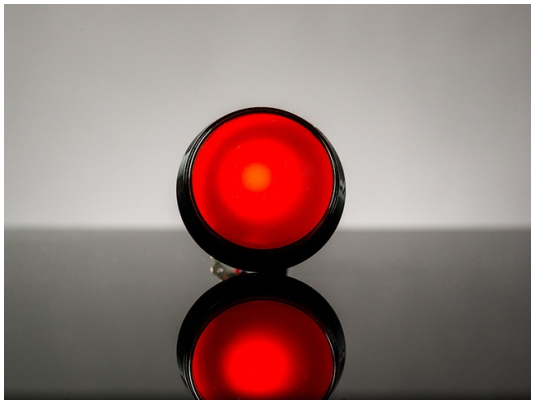
> NOTE: Older iOS devices and not support Bluetooth LE (also known as Bluetooth 4.0). Specifically iPhone prior to iPhone 4s and iPads prior to third-generation are reported to not support BLE.

Check this reference webpage link (https://adafru.it/uhd) or the specifications for your model to verify BLE compatibility.

Choosing Your Buttons

You will need to determine what type of button or buttons you want to use to activate the device. While it is possible to do switch control with a single switch, if the user is capable of pushing more than one button it will be much easier and faster. In single switch mode, iOS itself jumps from item to item and then you have to press a selection button to select that particular item. If you have 2 switches, one of them is used to manually advance the cursor to the next item while the other is used to actually select. Using three switches gives you the opportunity to scan forwards or backwards to the available options and not just continue to scan forward. The examples in this tutorial assume we are going to use three switches however we could use fewer or more depending on your needs and capabilities.

The type of switch that you use will depend upon the capabilities of your user. For people with plenty of strength but a lack of control such as users with cerebral palsy or those recovering from stroke you might want to consider this a large red arcade button.



Large Arcade Button with LED - 60mm Red

$5.95
IN STOCK

ADD TO CART

This is a 60 mm diameter button but there is also a 100 mm version and they come in not only red but, green, yellow, blue or white. If you want something a little smaller you might consider this 30 mm arcade style button.

Arcade Button - 30mm Translucent Red

$5.95
IN STOCK

ADD TO CART

It also comes in a variety of colors.

Keep in mind that fingers, hands, elbows are not the only options available. The arcade style button could also be mounted on a wheelchair headrest to be pushed with your head. You might want to consider a foot switch such as this one.

Foot switch

$7.50
IN STOCK

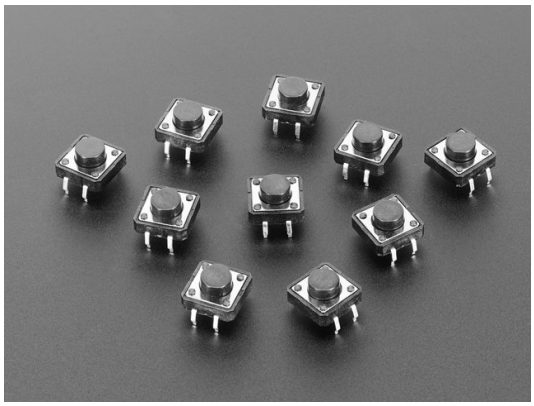
ADD TO CART

If the intended user has limited range of motion and possibly limited strength such as those with muscular dystrophy or ALS then you will probably want a smaller button with a soft touch. A variety of tactile pushbuttons are available from Adafruit such as these...

Colorful Square Tactile Button Switch Assortment - 15 pack

$5.95
IN STOCK

ADD TO CART

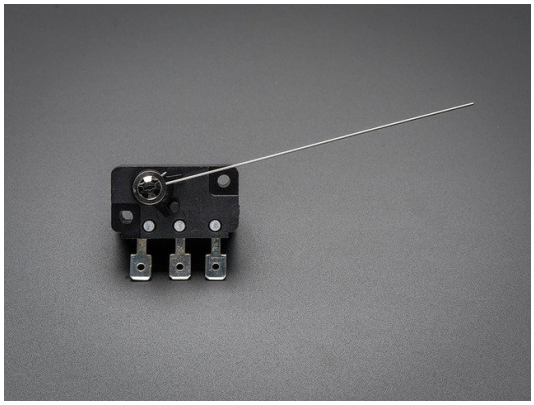


Tactile Switch Buttons (12mm square, 6mm tall) x 10 pack

$2.50
IN STOCK

ADD TO CART

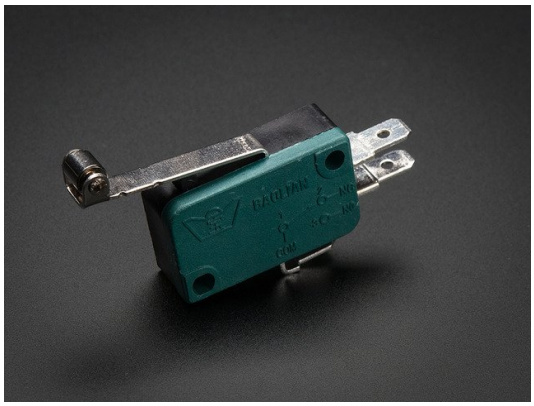
Personally I have a type of muscular dystrophy known as Spinal Muscular Atrophy and I only have limited use of my right hand. I use micro switches similar to these



Micro Switch w/Wire - Three Terminals

$2.95
OUT OF STOCK

OUT OF STOCK



Micro Switch w/Roller Lever - Three Terminal

$1.95
IN STOCK

ADD TO CART

Here is a link to all of the buttons sold by Adafruit https://www.adafruit.com/categories/235 (https://adafru.it/dSh) and all of the switches in the store https://www.adafruit.com/categories/155 (https://adafru.it/rcO)

Note that you want a momentary contact button or switch not a toggle switch, slide switch, or push off/push on power switch.

Although it's beyond the scope of this tutorial, you might want to use an analog joystick of some kind. There are several available in Adafruit store. You may not want an actual physical switch at all. Set up a photosensor and LED as if you pass your hand or finger between them it would break the beam and cause a signal to be sent to your Arduino. You could mount an accelerometer on a lever so that simply moving the position of the lever detects the change in angle. Alternatively you might want to consider some sort of capacitive touch solution. Here are a variety of capacitive touch devices available from Adafruit. https://www.adafruit.com/categories/60 (https://adafru.it/udV)

Also most Arduino compatible devices can be wired up to detect capacitive touch with nothing but a resistor and two input pins as described in this tutorial.

https://learn.adafruit.com/capacitive-touch-with-conductive-fabric-and-flora/ (https://adafru.it/udW)

Although that tutorial talks about the flora, the capacitive touch principle will work with most Arduino compatible AVR processors such as ATMega328 and ATMega 32u4. If you want to learn more about how capacitive touch devices work, we recommend this video "From the Desk of Lady Ada" (https://adafru.it/udX) where she describes capacitive touch in detail.

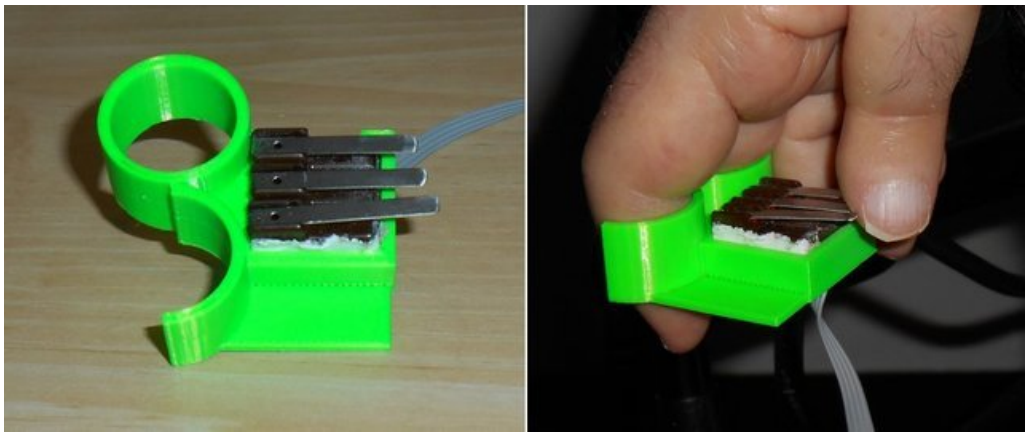
Professionally built adaptive technology devices refer to the switch devices as "AT Switches" for "Adaptive Technology Switches". The industry standard for connecting such devices is a 1/8 inch or 3.5 mm mono plug and jack. If you want your device to be compatible with this industry standard, you should put a 3.5 mm jack in your device and a 3.5 mm

plug on your switch. Here are some sources Digi-Key. Similar devices may be available at your local RadioShack or other suppliers.

Digi-Key		In line jack
Digi-Key		Panel mount jack
Digi-Key		In-line plug

Once you have selected the proper buttons, you need to consider where and how to mount them for the convenience of the user. If you have access to a 3D printer you may want to build a specialized enclosure for this project with the buttons mounted on the top. Here is a video from ATmakers.org (https://adafru.it/udY) that talks about how to mount commercially available AT switches on a standard camera mount. Use their solutions or use it to inspire you to create a mount of your own. https://youtu.be/Y-_8y-hHtM0 (https://adafru.it/udZ)

You may want to create a specialized piece to assist the user to hold onto micro switches in one hand. Here's a photo of a set of three micro switches in a 3D printed ring that helps me hold the buttons in my right hand where I can push the buttons with my thumb.



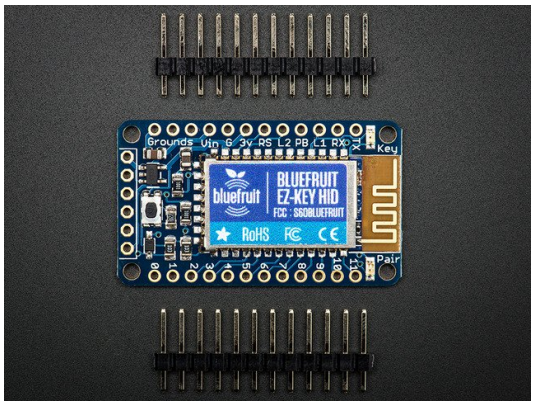
Each user will have different needs. You can use your maker skills to adapt and create something unique to the individual that may not be available in a commercially manufactured product.

Simplest Solutions

We need a device which can send single Bluetooth keypresses to your iOS device. Typically you might use the spacebar or perhaps a tab key to advance the scan to the next item in the enter key to select. However switch control is so flexible that any key can be set up.

You may not need to build a device at all, any Bluetooth keyboard that is capable of connecting to an iOS device can be used as a switch control device. If the user has sufficient dexterity and strength to push one or two buttons on a keyboard you can just pair that keyboard with your iOS device. You then configure the switch control on your iPhone or iPad to use whatever buttons on the keyboard that you can easily reach.

If you do want to build a device, the simplest one would use the Bluefruit EZ-Key - 12 Input Bluetooth HID Key. Without any extra programming you can connect up to 12 buttons to this device and use them to transmit keystrokes that can be used by iOS switch control.

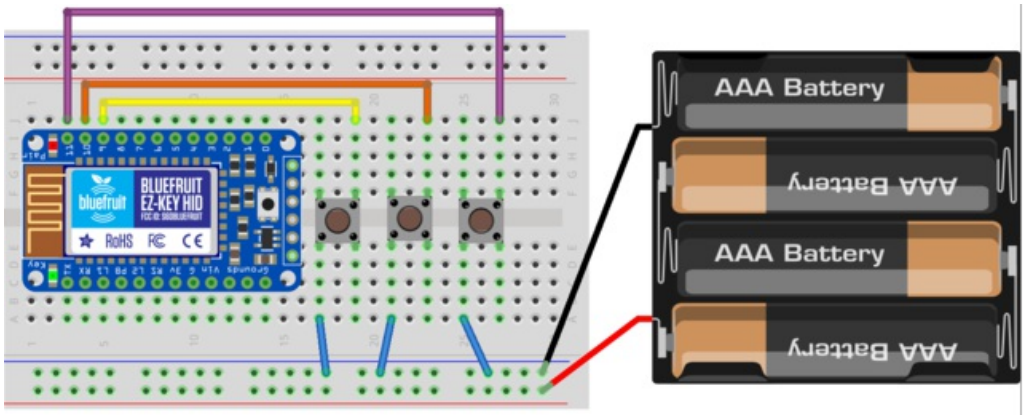


Bluefruit EZ-Key - 12 Input Bluetooth HID Keyboard Controller

$19.95
OUT OF STOCK

OUT OF STOCK

Below is a sample circuit connecting three pushbuttons to this device. You supply power anywhere from 3-16 volts. One side of each pushbutton is connected to ground and the other side is connected to one of the 12 input pins.



Complete details on this device are available in the product tutorial here...

Introducing Bluefruit EZ-Key (https://adafru.it/vwD)

No programming is required. This is the simplest device we can build. Although you can reconfigure this device in many ways such as redefining which keys can be transmitted, none of those advanced features for this device are

necessary. All you have to do is supply power and hookup some switches.

Theoretically you should be able to use any keypress to activate switch control. However on occasion if you're pressing the switches rapidly while typing, it will not interpret the keypress as a switch command. It might accidentally interpret as a character typed on the keyboard. For example on my feather based BLE device I use the letters "l", "r", and "s" for the left, right, and select functions. Occasionally while typing I will see one of those letters accidentally inserted into my text.

The switch locations 0-3 on the Adafruit EZ-Key will transmit arrow keys. If they glitch while typing something, it can really mess up your typing. That is why we have wired this to locations 9, 10, 11 because they correspond to the letters "a", "s", and "d" respectively. If you do get a glitch while typing, the worst that will happen is you will get an extraneous letter "a", "s", or "d" in your text message which you can easily delete.

If we need something more sophisticated than the EZ-Key, we can use one of several available Adafruit Feather BLE devices. In the a later section will give an example of using the Feather 32u4 BLE complete with schematics and source code and tips on how to customize it to make it as flexible as possible.

Configuring iOS Switch Control

In this section we will show you how to pair your BLE device to your iOS device and how to set up basic switch control using up to three buttons. Although this section is written with the EZ-Key HID device in mind, the process is virtually identical when using a Feather BLE or other devices. At the end of this page is a YouTube video that illustrates the process but we will describe it here for reference.

Pairing Your Device

On your iOS device go into **"Settings-> Bluetooth"** and turn on your BLE device. You should see the device listed in your available devices. Initially it might be described as simply **"Keyboard"** but eventually it will switch over to **"Adafruit EZ-Key"** followed by a four character identifier. You may have to press the **"pair"** button on the EZ Key. Tap on the device name to connect. Once you have paired your device you will not need to do it again. It should connect automatically thereafter.

Configuring the Switches

Now on your iOS device go into **"Settings-> General-> Accessibility-> Switch Control"**. Note that the **"Tap Behavior"** is set to **"Default"** and **"Scanning Style"** is set to **"Manual"**. If they are not configured that way, change them now.

Now select **"Switches"**. If you've not done this before it will say that there are **"0"** switches defined. Tap on **"Add new switch"** and then **"External"**. You will now be prompted to press one of your switches. Let's start with the **"Select"** switch. On your BLE device, press the switch briefly. If your iOS device recognizes it, it will prompt you to label it. In this case type the word **"Select"** and then click on **"Save"**.

Now you will be given a list of possible functions that can be assigned to that switch. They are listed in two categories **"Scanner"** and **"System"**

We want to assign the function **"Select Item"** to this particular switch. After selecting this, it will take us back to the **"Switches"** menu and we should add another switch. Repeat the process for the left and right switches. On the right switch we will assign the function **"Move to Next Item"**. Similarly add the left switch and assign it the function **"Move to Previous Item"**.

Now go back one level to **"Switch Control"** and turn on switch control with the slider button. All of the above had to be done manually using the iOS device's touchscreen. However from this point forward you will be able to do anything else by simply pushing the right combination of the three switches we have just configured. The only item that you cannot do under switch control is to add an additional switch. That must be done manually. Although we have configured three switches, you can define additional ones if you have the hardware to do so. Also note that switch control can be used with two switches or even one a single switch. If you are using just 2 switches then you need one for **"Move to Next Item"** and one as the **"Select"** switch.

Single switch control is possible under iOS. Your iOS device will constantly move the cursor rectangle from item to item and then you press the **"Select" s**witch when it is at the right place. Configuring single switch control is a bit difficult and we will not be covering how to use it in this tutorial but you can feel free to explore those options on your own. Under **"Scanning Method"** change it from **"Manual"** to **"Auto Scanning"** or **"Single Switch Step Scanning"**.

How to Use Switch Control

Once you have enabled switch control, you will notice that a blue rectangle will highlight an area of your screen. You should press the **"Right"** switch repeatedly and that rectangle will move from section to section on the screen. To make things go quicker, it will attempt to group together collections of objects on the screen. If you want to access something within the group then you press the **"Select"** switch. And then you can scan and select items within the group. If you overshoot the item that you want, you can press the **"Left"** switch and scan backwards.

Once you reach the item you want to choose, you press the **"Select"** switch but it does not automatically tap on that item. A small menu will pop up that will give you the option to tap on the selected item or perhaps to do some other function. Most of the time you want to tap and you should simply press the **"Select"** switch yet again to tap.

Instead of telling it to tap, you can optionally press the **"Right"** switch to explore other options. This brings up what is called the **"Scanner Menu"**. This will give you a variety of options such as pressing the home button, scrolling the screen in various directions, performing other touchscreen gestures such as pinch, or accessing device features such as the control center, notification area, volume controls etc. To have complete control of your iOS device, you must have some way to access this scanner menu.

Advanced Configuration

Because it might be annoying to have to constantly press **"Select"** twice in order to tap on a particular item, you may want to consider some advanced configuration options. The behavior we have described above is called **"Tap Behavior: Default"** as defined in the **"Switch Control"** main menu. The other varieties are **"Auto Tap"** and **"Always Tap"**.

When using Auto Tap, when you select an item it will briefly turn solid blue. If you do nothing for the next fraction of a second it will automatically tap for you. But if you quickly press the **"Select"** switch again, it will bring up the scanner menu. Note: we have not demonstrated the Auto Tap feature in the YouTube video at the end of this page. But you can try it out yourself.

The **"Always Tap"** behavior automatically taps on the item anytime you select it. In order to use the **"Always Tap"** behavior you must have an alternate way to access the scanner menu. You could decide to forgo the use of a **"Left"** switch and reassign its function to **"Scanner Menu"**. You could add a fourth switch to your BLE device and assign it the **"Scanner Menu"** function.

However our favorite solution to this problem is to make use of the **"Long Press"** feature. When this feature is enabled, each switch on your BLE device can have 2 functions. The default function is activated when you briefly press the switch. However if you hold the switch down for more than one second, an alternate function is activated. We recommend that you use the long press of your **"Select"** button to activate the **"Scanner Menu"** function.

> Note that depending on the physical ability of the user, you may not be able to accurately use short and long presses. In that situation, you will have to use either the default tap behavior or the auto tap behavior or assign an additional switch to the scanner menu function.

Under the **"Switch Control"** main menu select **"Long Press"** and turn it on. You also have the option to define how long the button needs to be depressed in order to qualify as a long press.

After you have configured your **"Select"** switch's long press function to activate the **"Scanner Menu"**, you can then change your **"Tap Behavior"** to use **"Always Tap"**. You will likely find this configuration is much easier to use than the default tap behavior.

You can also assign long press functions to your other buttons. Because pressing the **"Home"** button is something you will frequently do, we recommend assigning that function to the long press of your **"Right"** switch even though the "Home" function is available from the scanner menu.

In the demo video below we assigned a long press of the **"Left"** switch to activate the Siri digital assistant. You could assign it to some other function such as the notification center or whatever you want.

Except as noted, everything on this page is illustrated in the YouTube video below.

Arduino and Feather Based Solutions

Why Use an Arduino or Feather?

We were able to create a very useful solution based on the Adafruit EZ-Key Bluetooth BLE device but you might want to add additional capability. If you use an Arduino combined with a Bluetooth BLE friend (http://adafru.it/2633) you can add additional capability to the device. And a better solution yet is to use an Adafruit Feather 32u4 Bluefruit BLE (http://adafru.it/2829) or Adafruit Feather M0 Bluefruit BLE (http://adafru.it/2995). The Feather boards are small, inexpensive, and have the advantage of a built-in Lipo battery charger.

By using an Arduino or Feather platform, you have the option of adding other capabilities to the device. For example I have built a device that not only does BLE switch control of my iPhone, I can switch it into an additional mode where it serves as an infrared remote for my TV, cable box, Blu-ray etc. While we will not go into any details of how to create that complicated of a device in this tutorial, we do provide a multimode example sketch that illustrates the concept. Using a programmable platform gives you lots of options to use the device for more than just iOS switch control.

In our simple solution using the EZ-Key device, we were able to program 6 functions using just three switches by taking advantage of the long press feature. However if we allow for the possibility of pushing two or more buttons simultaneously, we can add even more functionality to the device. The use of more than one button simultaneously is called "chording" because it is like playing a chord on a piano by hitting more than one key at a time. By using a combination of short and long presses and short and long cords we can provide many more capabilities.

> NOTE: Depending upon the physical capabilities of your user and the location and types of switches that you use, it may be impossible or impractical to use a chord system.

Required Libraries and Source Code

There are three versions of the source code for this project. All of them are available in a GitHub repository in the link below. The three versions are:

- Simple: This is a simple version of the code that duplicates the functionality of the EZ-Key device we described earlier. You would use it if you are using a feather board to take advantage of the built-in battery charger but needed no other functionality.
- Multimode: This version shows how you can switch out of iOS Switch Control mode and use the feather board for some other functionality.
- Corded: This version shows how you can get more functionality by allowing for the pressing of multiple switches simultaneously.

All of the examples in this tutorial have been tested with both 32u4 and M0 feather devices. We recommend that you start with implementing the "simple" version first and then move on to the other versions if you need additional functionality.

You will need the Adafruit BLE library called "Adafruit_BluefruitLE_nRF51" available in the link below. All three sketches require the "BluefruitConfig.h" file that comes with all of the example sketches from that library. For more information on how to install this library visit this link (https://adafru.it/ue0).

Also to make the source code easier to read and understand, we have moved some of the common code from our three sketches into a separate file called "BluefruitRoutines.h".

Here are the links to the Adafruit BLE library and to our GitHub repository for this tutorial.

https://adafru.it/f4W

https://adafru.it/f4W

https://adafru.it/ue1

https://adafru.it/ue1

Here is the source code for the "BluefruitRoutines.h" file that is common to all three sample sketches. It creates an instance of the ble object and initializes the device. It renames it as "iOS Switch Control" so when you go to pair it with your iOS device it will be easily identifiable. This file also includes some debugging routines and error message routines. At the beginning of each sketch there is a "#define MY_DEBUG 1" which you can change to "0" to turn off debugging. We recommend you initially compile with debugging turned on and then once you have everything working properly, recompiling with debugging turned off.

NOTE: Debugging requires use of the serial monitor. If you do not have the serial monitor open, the sketch may not work if you have enabled debugging. The serial monitor is not necessary if debugging has been turned off.

```
/* This is all the blueprint specific code for all of our
 *  iOS switch control example code. We have moved it here to make
 *  the main source code more legible.
 */

#include <SPI.h>
#include "Adafruit_BLE.h"
#include "Adafruit_BluefruitLE_SPI.h"
#include "BluefruitConfig.h"
Adafruit_BluefruitLE_SPI ble(BLUEFRUIT_SPI_CS, BLUEFRUIT_SPI_IRQ, BLUEFRUIT_SPI_RST);

//Debug output routines
#if (MY_DEBUG)
  #define MESSAGE(m) Serial.println(m);
  #define FATAL(m) {MESSAGE(m); while (1);}
#else
  #define MESSAGE(m) {}
  #define FATAL(m) while (1);
#endif

void initializeBluefruit (void) {
  //Initialize Bluetooth
  if ( !ble.begin(MY_DEBUG))
  {
    FATAL(F("NO BLE?"));
  }
  //Rename device
  if (! ble.sendCommandCheckOK(F( "AT+GAPDEVNAME=iOS Switch Access" )) ) {
    FATAL(F("err:rename fail"));
  }
  //Enable HID keyboard
  if(!ble.sendCommandCheckOK(F( "AT+BleHIDEn=On" ))) {
    FATAL(F("err:enable Kb"));
  }
  //Add or remove service requires a reset
  if (! ble.reset() ) {
    FATAL(F("err:SW reset"));
  }

}
```

Simple Feather Code

The sample sketch presumes that you have three switches connected to the feather board. We have chosen to use pin numbers **"A0"**, **"A1"**, and **"A2"** to represent the **"Previous"**, **"Select"**, and **"Next"** functions. Although these are "analog" capable pins we are using them as standard digital input pins with the INPUT_PULLUP parameter. We chose them because they are adjacent to a ground pin which makes it easy to wire them up and it is unlikely that they will conflict with any other use of the device. It also gives you the option to rewrite the code to use an analog joystick rather than momentary press switches.

The function `readSwitches()` reads all three switches and turns them into a single unsigned byte value.

The main loop repeatedly calls this function and depending on the results initiates transmission of a keypress via the BLE connection.

The function `pressKeyCode(c)` handles the transmission of a keypress. It first sends a BLE command to press and hold the proper key. Then it repeatedly calls `readSwitches()` until it detects that the switch has been released. Then it sends a release command via BLE. In this way we can correctly process long keypresses. There is a `delay(100)` that helps to denounce the switches. If you find you're getting accidental double presses you could increase this value or if it is processing your presses too slowly you could decrease it.

NOTE: The codes being sent by the BLE command are not standard ASCII character codes. They are special keyboard press codes. For more information on the `AT+BLEKEYBOARDCODE=` BLE command see this following link (https://adafru.it/ue2).

Here is the source code for the "simple" sample program.

```
//iOS switch control example
//Simple version
#define MY_DEBUG 1
#include "BluefruitRoutines.h"

//Pin numbers for switches
#define PREVIOUS_SWITCH A0
#define SELECT_SWITCH A1
#define NEXT_SWITCH A2

//Actions
#define DO_PREVIOUS 1
#define DO_SELECT   2
#define DO_NEXT     4

uint8_t readSwitches(void) {
  return (~(digitalRead(PREVIOUS_SWITCH)*DO_PREVIOUS
      + digitalRead(SELECT_SWITCH)*DO_SELECT
      + digitalRead (NEXT_SWITCH)*DO_NEXT)
     ) & (DO_PREVIOUS+ DO_SELECT+ DO_NEXT);
}

//Translate character to keyboard keycode and transmit
void pressKeyCode (uint8_t c) {
  ble.print(F("AT+BLEKEYBOARDCODE=00-00-"));
  uint8_t Code=c-'a'+4;
  if (Code<0x10)ble.print("0");
  ble.print(Code,HEX);
  ble.println(F("-00-00-00-00"));
  MESSAGE(F("Pressed."));
  delay(100);//de-bounce
  while (readSwitches()) { //wait for button to be released
    /*do nothing*/
  };
  ble.println(F("AT+BLEKEYBOARDCODE=00-00"));
  MESSAGE(F("Released"));
}

void setup() {
#if(MY_DEBUG)
  while (! Serial) {}; delay (500);
  Serial.begin(9600); Serial.println("Debug output");
#endif
  pinMode(SELECT_SWITCH, INPUT_PULLUP);
  pinMode(NEXT_SWITCH, INPUT_PULLUP);
  pinMode(PREVIOUS_SWITCH, INPUT_PULLUP);
  initializeBluefruit();
}

void loop() {
  uint8_t i=readSwitches();
  switch (i) {
    case DO_PREVIOUS: pressKeyCode('p'); break;
    case DO_SELECT:   pressKeyCode('s'); break;
    case DO_NEXT:     pressKeyCode('n'); break;
  }
}
```

You should compile and upload the sketch and open your serial monster. You can see initialization steps that the program does. Then try pressing and releasing each of the three switches and note how it first transmits the keypress and then transmits the release signal.

You should pair the BLE device to your iOS device and configure switch control as described in the previous section of this tutorial. It should function virtually identically to the description of the EZ-Key device we described earlier.

Once you have everything working properly you should disable debugging by editing the line near the top to read `#define MY_DEBUG 0`

Multimode Feather Code

As we mentioned earlier, one of the advantages of the using an Arduino or feather based system is it allows you to use the device for more than just iOS switch control. Personally I have implemented a multiuse device which not only controls my iPhone using switch control, it can also be changed into another mode where it is used as an infrared TV remote. While we aren't going to go into the complexities of writing such an application, we have included this "multimode" sample sketch that shows you how you can switch in and out of iOS switch control and do something else. We recommend you implement the "simple" version first and get it working before attempting this version.

A global variable called `Mode` is initialized to `SWITCH_MODE`. This makes the program operate identically to the "simple" version we showed you earlier. However if you hold down one of the switches longer than 10 seconds (the value `EXIT_LIMIT`) then it switches into an alternate mode by setting `Mode= OTHER_MODE;` Our sample code while using `OTHER_MODE` simply blinks the pin 13 LED five times and then reverts to `SWITCH_MODE`. You can come up with something much more useful to do in that alternate mode. Perhaps you would have code that would turn them off or on environmental controls, operate pan and tilt servos to help you take a photo with your iPhone, or some other useful function. Whenever you want to go back to iOS Switch Control you simply set `Mode=SWITCH_MODE`

This sketch depends on both of the standard `BluefruitConfig.h` and our `BluefruitRoutines.h` file. Here is the source code for the main sketch.

```
//iOS switch control example
//Multiple mode version
#define MY_DEBUG 1
#include "BluefruitRoutines.h"

//Pin numbers for switches
#define PREVIOUS_SWITCH A0
#define SELECT_SWITCH A1
#define NEXT_SWITCH A2

//Actions
#define DO_PREVIOUS 1
#define DO_SELECT   2
#define DO_NEXT     4

//Flag to tell if you are using switch control or other functions
uint8_t Mode;
#define SWITCH_MODE 1
#define OTHER_MODE 2
#define EXIT_LIMIT 10000ul  //Time limit for switching modes

uint8_t readSwitches(void) {
  return (~(digitalRead(PREVIOUS_SWITCH)*DO_PREVIOUS
      + digitalRead(SELECT_SWITCH)*DO_SELECT
      + digitalRead (NEXT_SWITCH)*DO_NEXT)
     ) & (DO_PREVIOUS+ DO_SELECT+ DO_NEXT);
}

//Translate character to keyboard keycode and transmit
void pressKeyCode (uint8_t c) {
  uint32_t Start=millis();
  ble.print(F("AT+BLEKEYBOARDCODE=00-00-"));
  uint8_t Code=c-'a'+4;
  if (Code<0x10)ble.print("0");
  ble.print(Code,HEX);
  ble.println(F("-00-00-00-00"));
  MESSAGE(F("Pressed "));
```

```
  MESSAGE(F("Pressed."));
  delay(100);//de-bounce
  while (readSwitches()) { //wait for button to be released
    /*do nothing*/
  };
  if( (millis()-Start) > EXIT_LIMIT) {
    Mode= OTHER_MODE;
  }
  ble.println(F("AT+BLEKEYBOARDCODE=00-00"));
  MESSAGE(F("Released"));
}

// This reads the switches and decides what keypresses
// to send to the BLE device.
void doSwitchMode (void) {
  uint8_t i=readSwitches();
  switch (i) {
    case DO_PREVIOUS: pressKeyCode('p'); break;
    case DO_SELECT:   pressKeyCode('s'); break;
    case DO_NEXT:     pressKeyCode('n'); break;
  }
}

// This routine can perform some alternate function other than
// iOS switch control. It will be engaged if you hold a button
// for a very very long time.
void doOtherMode (void) {
  /*Insert your code here*/
  MESSAGE(F("Doing other mode."));
  //For demonstration purposes we will just blink the LED pin 13
  //a few times and then go back into switch control mode.
  pinMode(13, OUTPUT);
  for(uint8_t i=0;i<5;i++ ) {
    digitalWrite(13, HIGH); delay(1000);
    digitalWrite(13, LOW);  delay(1000);
  }
  Mode=SWITCH_MODE;
  MESSAGE(F("Returning to switch mode"));
}

void setup() {
#if(MY_DEBUG)
  while (! Serial) {}; delay (500);
  Serial.begin(9600); Serial.println("Debug output");
#endif
  pinMode(SELECT_SWITCH, INPUT_PULLUP);
  pinMode(NEXT_SWITCH, INPUT_PULLUP);
  pinMode(PREVIOUS_SWITCH, INPUT_PULLUP);
  initializeBluefruit();
  Mode=SWITCH_MODE;
}

void loop() {
  switch(Mode) {
    case SWITCH_MODE: doSwitchMode(); break;
    case OTHER_MODE:  doOtherMode(); break;
  }
}
```

You should compile and upload the sketch and open your serial monster. You can see initialization steps that the program does. Then try pressing and releasing each of the three switches and note how it first transmits the keypress and then transmits the release signal.

Try holding down one of your switches for more than 10 seconds. Note how it switches into the alternate mode and blinks the pin 13 LED. After five blinks, it will revert back to switch control mode.

If you have not done so already, you should pair the BLE device to your iOS device and configure switch control as described in the earlier section of this tutorial. It should function virtually identically to the description of the EZ-Key device we described earlier.

Once you have everything working properly you should disable debugging by editing the line near the top to read `#define MY_DEBUG 0`

Chorded Feather Code

Using three switches with their short and long keypresses you can have 6 different functions defined for your switch control. But you may wish to add even more functionality without adding more switches. You can do so by using a "chord" system. This means that you press more than one switch simultaneous like you are playing a chord on a piano keyboard.

Suppose you have three switches we will label **"1"**, **"2"**, & **"3"**. You can get an additional function by pressing **"1"** & **"2"** simultaneously or similarly **"2"** & **"3"** simultaneously. In my own personal application my switches are 3 small lever micro switches that I push with my thumb. I can bridge my thumb across two adjacent switches easily. I can also bridge all three switches and press them all simultaneously. However in that configuration I cannot press **"1"** & **"3"** simultaneously because I'm only pushing switches with with my thumb. Depending on your switch configuration and your user's capability you may be able to play the "13" chord and/or you may not be able to press all three simultaneously. Given that I can press each of the three switches individually plus the "12", "23", and "123" combinations that gives me 6 combos. If we include both long and short presses that gives me up to 12 possible switches. I use the "123" combination to switch out of switch control mode and into my IR remote control mode. You can configure this anyway you want.

Altogether using 3 switches in all combinations and long and short presses you can get 14 different combinations. Currently there are only 13 possible functions that you can control in iOS switch control and you may not want to assign switch combinations to all of them. In our previous examples we got by really well with just six functions. However if you have a multimode application, you may want to use some of those extra capabilities in your other application.

There is however a problem using short and long keypresses when using a chorded system. In our previous code, the instant that you pressed any switch, it would send the keypress signal immediately and not send the release signal until you released the switch. However the possibility of pressing 2 switches at precisely the exact same instant is infinitesimal. That means when you tried to do a two switch chord, one of the two switches would get transmitted before you could get to the other one.

When using a chorded system, you cannot transmit the BLE signal upon pressing the switch. When you detect that a switch has been pressed, you have to wait to see if other switches are going to be pressed as well. You have to do a logical "or" of all of the switches that are pressed until such time that all of the switches have been released. Then and only then can you transmit the proper BLE signal.

For example if I press "1" and while holding it press "2" then I release "1" and then finally release "2" I want this to be transmitted as a "12" signal. But I can't transmit that signal until everything has been released.

This means that I cannot use the iOS switch control long press feature. Instead I need to keep track of how long the switch presses were in my Arduino/feather sketch and instead send a different code. So we will not be configuring iOS Switch Control to use long presses. Everything will be considered a momentary short press as far as our iOS device is concerned. However within our Arduino code we will keep track of short and long presses and send different commands based on how long the switches were pressed.

For example in the simple version of the sketch, when you press the **"Select"** switch it would send the letter "s". Then when you release that switch however long you held it, it would send a release command. The switch control software in iOS would determine whether it was in short or long press and act accordingly. In this chorded system, if you press the switch briefly it will send "s" but if you press it a long time it will send "l". The iOS software treats it as if you pressed an entirely different switch for a short time.

Here is the sample "chord" code.

```
//iOS switch control example
```

```
//Multiple mode version
#define MY_DEBUG 1
#include "BluefruitRoutines.h"

//Pin numbers for switches
#define PREVIOUS_SWITCH A0
#define SELECT_SWITCH A1
#define NEXT_SWITCH A2

//Actions
#define DO_PREVIOUS 1
#define DO_SELECT 2
#define DO_NEXT   4
#define DO_SPLIT  (DO_PREVIOUS+DO_NEXT)
#define DO_DOWN   (DO_SELECT+DO_NEXT)
#define DO_UP     (DO_PREVIOUS+DO_SELECT)
#define DO_ALL    (DO_PREVIOUS+DO_SELECT+DO_NEXT)
#define DO_LONG   8
#define DO_END    (DO_PREVIOUS+DO_LONG)
#define DO_LONG_SEL (DO_SELECT+DO_LONG)
#define DO_HOME   (DO_NEXT+DO_LONG)
#define DO_RIP    (DO_SPLIT+DO_LONG)
#define DO_PGUP   (DO_UP+DO_LONG)
#define DO_PGDN   (DO_DOWN+DO_LONG)
#define DO_EXIT   (DO_ALL+DO_LONG)

//Flag to tell if you are using switch control or other functions
uint8_t Mode;
uint8_t Prev, Buttons;
#define SWITCH_MODE 1
#define OTHER_MODE 2

uint8_t readSwitches(void) {
  return (~(digitalRead(PREVIOUS_SWITCH)*DO_PREVIOUS
      + digitalRead(SELECT_SWITCH)*DO_SELECT
      + digitalRead (NEXT_SWITCH)*DO_NEXT)
     ) & (DO_PREVIOUS+ DO_SELECT+ DO_NEXT);
}

uint8_t getChord() {
  uint32_t Delta, Start;
  uint8_t Sample;
  Prev=Buttons; //Save previous button state
  Buttons=readSwitches();//Get current button state
  if (Buttons) {
    if(Buttons!=Prev) {//If it's different, restart timer
      Delta=0; Start=millis();
    }
    do {
      delay(50);
      //Take another sample and logical bitwise "or" together
      Buttons |= (Sample = readSwitches());
      Delta=millis()-Start;//Compute difference
    } while (Sample);
    if (Delta > 1000)  Buttons |= 8;//Greater than one second is a long
  } else {
    Prev=0;
  }
  return  Buttons;
};
```

```
// This reads the switches and decides what keypresses
// to send to the BLE device.
void doSwitchMode (void) {
  uint8_t i=getChord();
  if (i) {
    ble.print(F("AT+BleKeyboard="));
    switch (i) {
      case DO_PREVIOUS: ble.println("p"); MESSAGE(F("PREV")); break;
      case DO_SELECT:   ble.println("s"); MESSAGE(F("SELECT")); break;
      case DO_NEXT:     ble.println("n"); MESSAGE(F("NEXT")); break;
      case DO_SPLIT:    ble.println("v"); MESSAGE(F("SPLIT")); break;
      case DO_DOWN:     ble.println("d"); MESSAGE(F("DOWN")); break;
      case DO_UP:       ble.println("u"); MESSAGE(F("UP")); break;
      case DO_PGUP:     ble.println("t"); MESSAGE(F("PAGE UP")); break;
      case DO_LONG_SEL: ble.println("l"); MESSAGE(F("LONG SEL")); break;
      case DO_PGDN: ble.println("b"); MESSAGE(F("PAGE DOWN")); break;
      case DO_RIP:  ble.println("i"); MESSAGE(F("RIP")); break;
      case DO_END:  ble.println("e"); MESSAGE(F("END")); break;
      case DO_HOME: ble.println("h"); MESSAGE(F("HOME")); break;
      case DO_ALL:
      case DO_EXIT: ble.println("a"); MESSAGE(F("ALL3"));
          Mode=OTHER_MODE; delay(500); break;
    }
  }
}

// This routine can perform some alternate function other than
// iOS switch control. It will be engaged if you hold a button
// for a very very long time.
void doOtherMode (void) {
  /*Insert your code here*/
  MESSAGE(F("Doing other mode."));
  //For demonstration purposes we will just blink the LED pin 13
  //until we press the exit code
  digitalWrite(13, HIGH); delay(1000);
  digitalWrite(13, LOW);  delay(1000);
  if(getChord()== DO_ALL) {
    Mode=SWITCH_MODE;
    MESSAGE(F("Returning to switch mode"));
  }
}

void setup() {
#if(MY_DEBUG)
  while (! Serial) {}; delay (500);
  Serial.begin(9600); Serial.println("Debug output");
#endif
  pinMode(SELECT_SWITCH, INPUT_PULLUP);
  pinMode(NEXT_SWITCH, INPUT_PULLUP);
  pinMode(PREVIOUS_SWITCH, INPUT_PULLUP);
  pinMode(13, OUTPUT);//Blink LED for other mode
  initializeBluefruit();
  Mode=SWITCH_MODE;
  Prev,  Buttons=0;
}

void loop() {
  switch(Mode) {
    case SWITCH_MODE: doSwitchMode(); break;
```

```
    case OTHER_MODE:  doOtherMode(); break;
  }
}
```

Note: This code is based on the "multimode" version of the sketch.

When configuring your iOS device, you will not configure long press versions of each of the three switches. Instead you will have as many as 13 switches to configure.

Of course you need not actually use every combination. You only need to configure as many as you would reasonably use.

This card uses a different command to send keypresses over the BLE connection. It does not use a separate press and release command. It does it all with one command. We recommend that you use unique lowercase letters when using this command in switch control applications. We have not tested other types of keystrokes and do not recommend using anything other than lowercase letters.

Upload the sketch and open the serial monitor. Try pressing different combinations of switches for different amounts of time. We have set it up so that if you press all three switches simultaneously for either a short or long period of time, it will switch you into the other mode which will blink the pin 13 LED. Unlike the previous multimode example, we will stay in this other mode and tell you again press all three switches simultaneously. Then it will revert back into normal iOS switch control mode.

Again our "other mode" is not very useful but it's just to demonstrate the concept.

Once you have everything working properly you should disable debugging by editing the line near the top to read `#define MY_DEBUG 0`

Final Thoughts

For most users the EZ-Key or the Simple Feather versions of this project would give you excellent access to your iOS device. Also note that such access can be obtained by using commercially made BLE devices and AT switches. However as a maker and hobbyist we can go above and beyond those simple ideas to create really powerful and effective devices using the higher concepts such as a multimode application possibly with chorded functionality.

Use your imagination and creativity to come up with the ultimate device which will give the maximum benefit to a disabled user.

Such a device has allowed me to go beyond simple things like posting on Facebook or playing iPhone games. Recently I was hospitalized and on a ventilator where I could not talk. I was able to type messages into the "Notes" application on my iPhone to talk to the doctors and nurses who were caring for me. I would've been lost without this device.

Naturally this is a simple solution that could have unknown bugs. On occasion my iPhone has spontaneously disconnected from the BLE device or accidentally turned off of switch control so you should be careful not to rely on such devices under "life critical" circumstances. But for everyday use, and when needing to communicate under difficult circumstances like I encountered, this is an excellent solution.

We recently come across an excellent iBook by iOS switch control user Christopher Hills. This is a free download on iTunes. Unfortunately like all iBooks it can only be read on a iOS device such as iPhone, iPad, or iPod touch but if you didn't have one of those you would not need his book anyway. This is a very detailed guide on how to use iOS switch control and is a must read for anyone who's using this feature.
https://itunes.apple.com/us/book/handsfree/id1040716154 (https://adafru.it/vDe)

Addendum: Here are some additional YouTube videos I created related to iOS switch control.