BleuIO

Getting Started Guide

www.bleuio.com

A product of Smart Sensor Devices

Table of Contents

How it works

Introduction

Thank you for purchasing the Smart USB Dongle 2.0!
In this user manual we will show you how to start using the Smart USB Dongle 2.0, what it can do and how to use it.

Smart USB Dongle 2.0 is a Bluetooth Low Energy (BLE) capable USB Dongle that, with simple AT-Commands, can easily be used to create new BLE applications.

When in the Central role it can scan and present a list of nearby advertising Peripherals, connect to a connectable peripheral and if the SPS profile is present in the peripheral it can receive and transmit data.

When in the Peripheral role it can start and stop advertising, set custom advertising- and response data, allow a Central device to connect to it and receive and transmit data via the SPS profile.

It is also possible to update the firmware via the USB port.

Requirements

The following hardware and software elements are required to use the Smart USB Dongle 2.0:

- The Smart USB Dongle 2.0.
- A Windows OS Computer with a functioning USB port.
- A terminal software such as RealTerm, Putty or Teraterm.

A target device such as:

- A Mobile Phone with Bluetooth and an App such as BLE Scanner, LightBlue or DSPS from Dialog Semiconductor
- A Computer with Bluetooth and a BLE scanning application installed
- Another Smart USB Dongle 2.0 connected to a Computer

Recommended Port Setup

Recommended	Port setup:
Baudrate:	57600
Data bits:	8
Parity:	None
Stop bits:	1
Flow controll:	None

How to use

Windows

Step 1

Figure 1: Connect the dongle to your computer.

Step 2

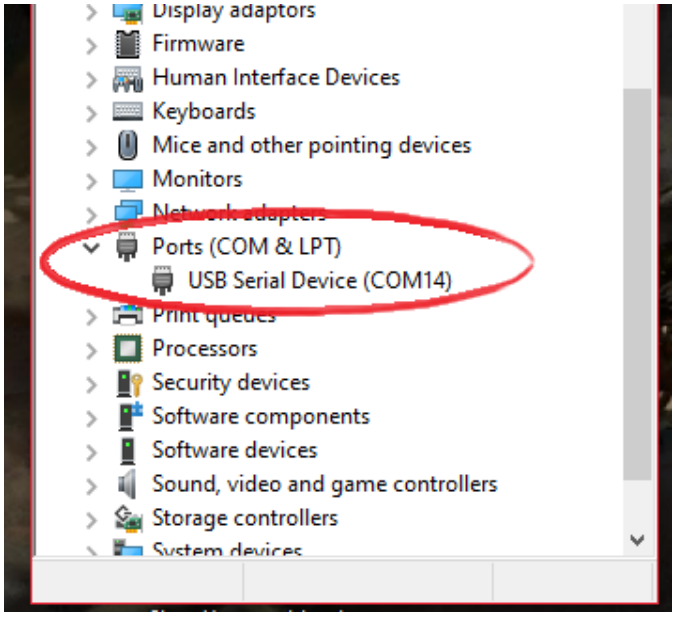


*Figure 2: Open Device Manager and check under **Ports (COM & LPT)** that a new COM port has popped up.*

You can open up Device Manager to see that the dongle is properly connected. The Smart USB Dongle 2.0 includes a bootloader which allows you to easily update the firmware (or flash your own application to the dongle, more on that *HERE*). When starting up, the dongle will open up a COM port for the bootloader for 10 seconds to allow you to update the firmware (or flash your own application). Afterwards it will close that port and open a new port for the Smart USB Dongle 2.0 application which is the one we're interested in here.

NOTE: The COM port number may vary and may not be the same as in the picture above.

Step 3

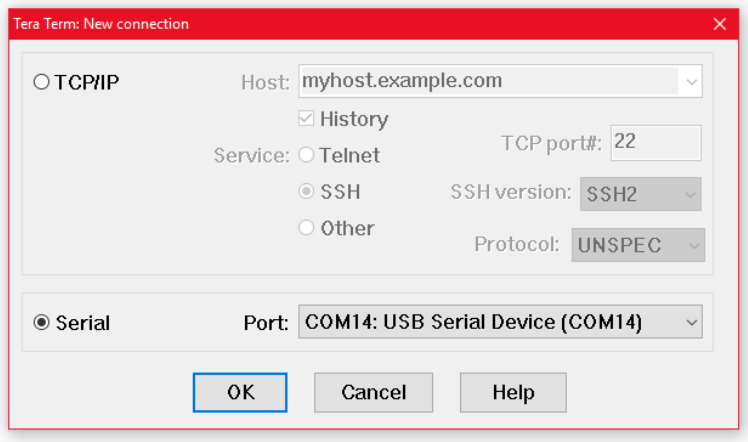


Figure 3: Open your terminal software and select the COM port your dongle is plugged into.

Step 4

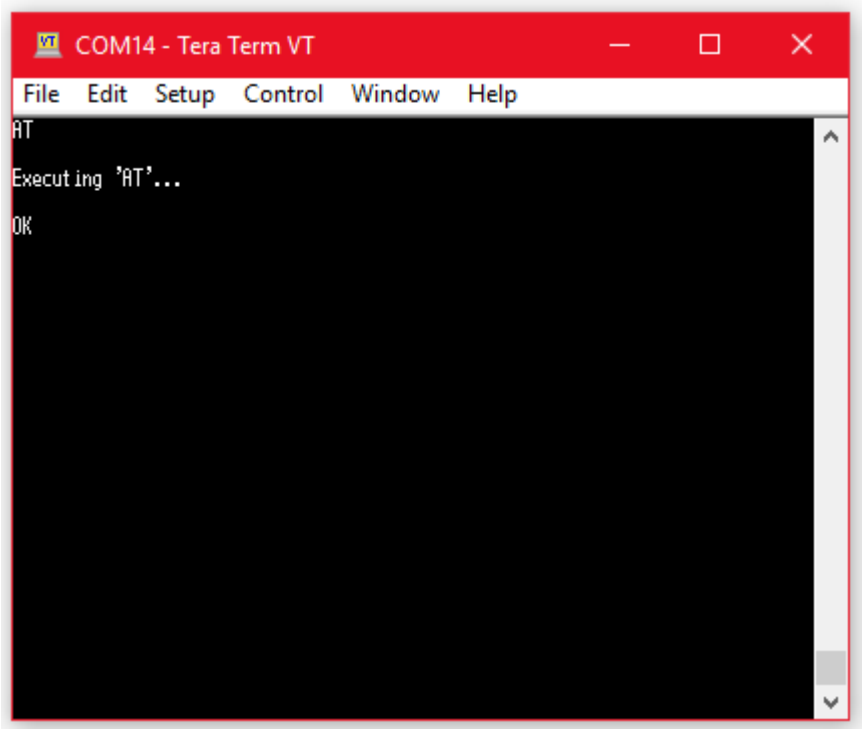


Figure 4: Try typing an AT-Command, for example AT.

If you get an OK response that means the dongle is working.

Done

Great job! You are now ready to start using Smart USB Dongle 2.0!

Now check out the different *AT Commands available* or how to get started *using scripts*.

Linux / Raspberry Pi

Step 1

Figure 1: Connect the dongle to your Linux / RaspberryPi.

Step 2

```
pi@raspberrypi:~ $ ls /dev
autofs           loop6            ram7     tty24  tty51      vcs4
block            loop7            ram8     tty25  tty52      vcs5
btrfs-control    loop-control     ram9     tty26  tty53      vcs6
bus              mapper           random   tty27  tty54      vcsa
cachefiles       mem              raw      tty28  tty55      vcsa1
char             memory_bandwidth rfkill   tty29  tty56      vcsa2
console          mmcblk0          serial1  tty3   tty57      vcsa3
cpu_dma_latency  mmcblk0p1        shm      tty30  tty58      vcsa4
cuse             mmcblk0p2        snd      tty31  tty59      vcsa5
disk             mqueue           stderr   tty32  tty6       vcsa6
fb0              net              stdin    tty33  tty60      vcsm
fd               network_latency  stdout   tty34  tty61      vcsm-cma
full             network_throughput tty    tty35  tty62      vcsu
fuse             null             tty0     tty36  tty63      vcsu1
gpiochip0        ppp              tty1     tty37  tty7       vcsu2
gpiochip1        ptmx             tty10    tty38  tty8       vcsu3
gpiomem          pts              tty11    tty39  tty9       vcsu4
hidraw0          ram0             tty12    tty4   ttyAMA0    vcsu5
hidraw1          ram1             tty13    tty40  ttyprintk  vcsu6
hwrng            ram10            tty14    tty41  uhid       vhci
initctl          ram11            tty15    tty42  uinput     video10
input            ram12            tty16    tty43  urandom    video11
kmsg             ram13            tty17    tty44  v4l        video12
log              ram14            tty18    tty45  vchiq      watchdog
loop0            ram15            tty19    tty46  vcio       watchdog0
loop1            ram2             tty2     tty47  vc-mem     zero
loop2            ram3             tty20    tty48  vcs
loop3            ram4             tty21    tty49  vcs1
loop4            ram5             tty22    tty5   vcs2
loop5            ram6             tty23    tty50  vcs3
```

Figure 2: Run: ls /dev.

To identify which device name the dongle is connected to, you will need to run:

```
1.  ls /dev
```

You might need to do it twice, once before you connect the dongle and once after to be able to identify which one is the device name. The Smart USB Dongle 2.0 includes a bootloader which allows you to easily update the firmware (or flash your own application to the dongle, more on that *HERE*). When starting up, the dongle will open up a COM port for the bootloader for 10 seconds to allow you to update the firmware (or flash your own application). Afterwards it will close that port and open a new port for the Smart USB Dongle 2.0 application which is the one we're interested in here. You can run:

```
1.  lsusb
```

It should list a device with the ID: 2dcf:6001 when the bootloader is active but change to 2dcf:6002 after 10 seconds when the application is running.

NOTE: The device name may vary and may not be the same as in the picture above.

Step 3

You will need a serial communication program to communicate with the dongle. For this tutorial we will be using Minicom. You can get Minicom by running:

```
1.  sudo apt-get install minicom
```

Now, to start using the dongle run the following command if, for example, your dongle is connected to the device name ttyACM0:

```
1.  minicom -b 9600 -o -D /dev/ttyACM0
```

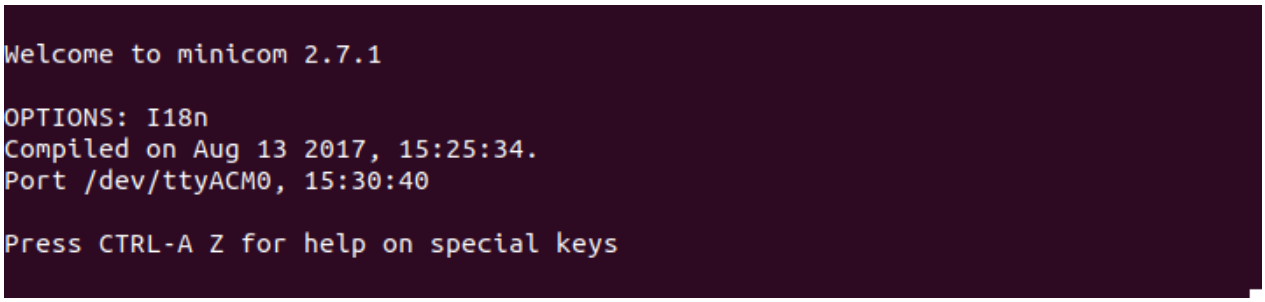


Figure 3: Open your terminal software.

Step 4

Now try typing an AT-Command. For example **AT** .
If you get an OK response that means the dongle is working.

Done

Great job! You are now ready to start using Smart USB Dongle 2.0!

Now check out the different *AT Commands available* or how to get started *using scripts*.

Mac OS X

Step 1

First connect the dongle to your Mac.

Step 2

To identify which device name the dongle is connected to, you will need to run:

```
1.  ls /dev/cu.*
```

You should see something like:

```
1.  $ ls /dev/cu.*
2.  /dev/cu.Bluetooth-Modem     /dev/cu.iPhone-WirelessiAP
3.  /dev/cu.Bluetooth-PDA-Sync  /dev/cu.usbmodem123456781
```

The dongle should show up as: **/dev/cu.usbmodem123456781** if it shows up with a different name use that instead.
When starting up, the dongle will open up a COM port for the bootloader for 10 seconds to allow you to update the firmware (or flash your own application). Unfortunately this is not as of yet available on Mac. Afterwards it will close that port and open a new port for the Smart USB Dongle 2.0 so you might need to wait a few seconds before doing the next step.

Step 3

You can either use the Mac OS X built in Terminal and **screen** or an external communication program to communicate with the dongle. For this tutorial we will show how to get started using terminal and **screen** and also a popular serial communication software called **Minicom** also via the terminal.
With Screen:

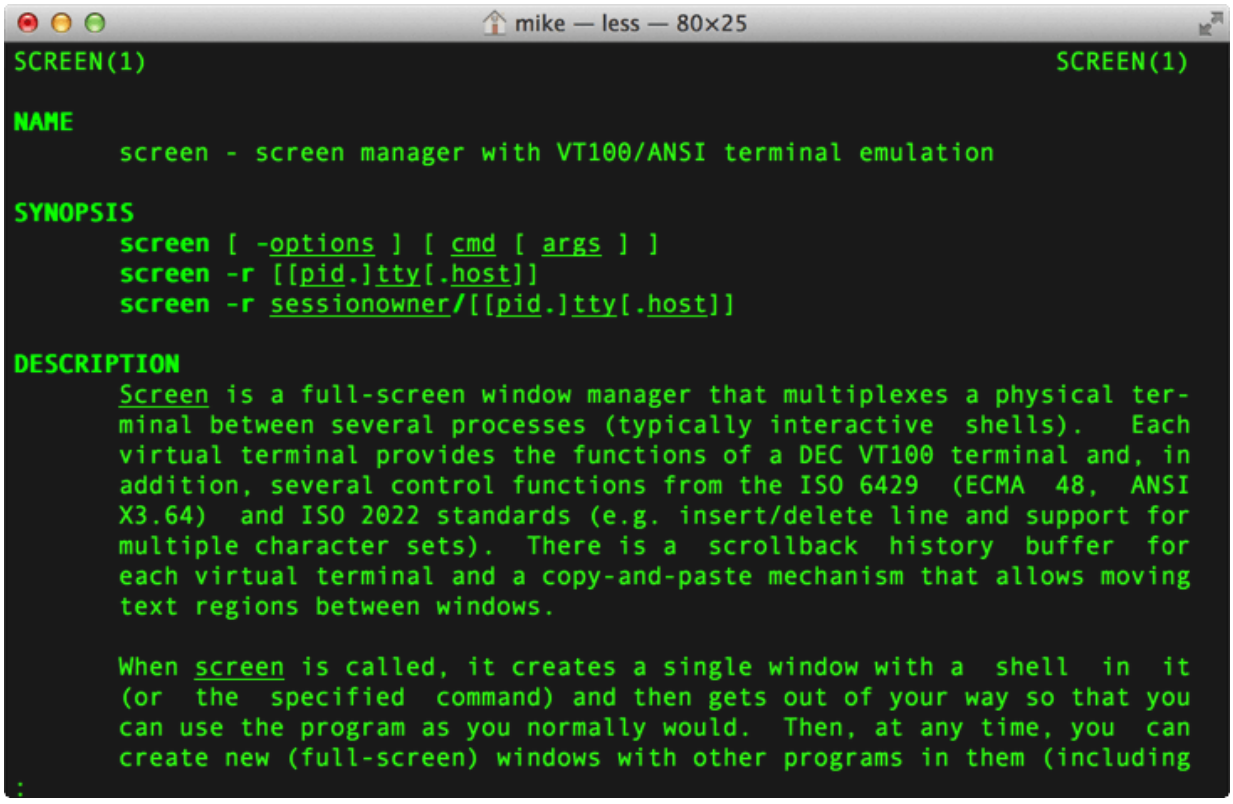


Figure 1: Terminal and screen.

Simply type:

```
1.  screen /dev/cu.usbmodem123456781 9600
```

With Minicom:
If you have Homebrew* you can get Minicom by simply running:

```
1.  sudo brew install minicom
```

(If you don't have Homebrew **click here to find out how to install it**)
Now you need to setup Minicom by running:

```
1.  minicom -s
```

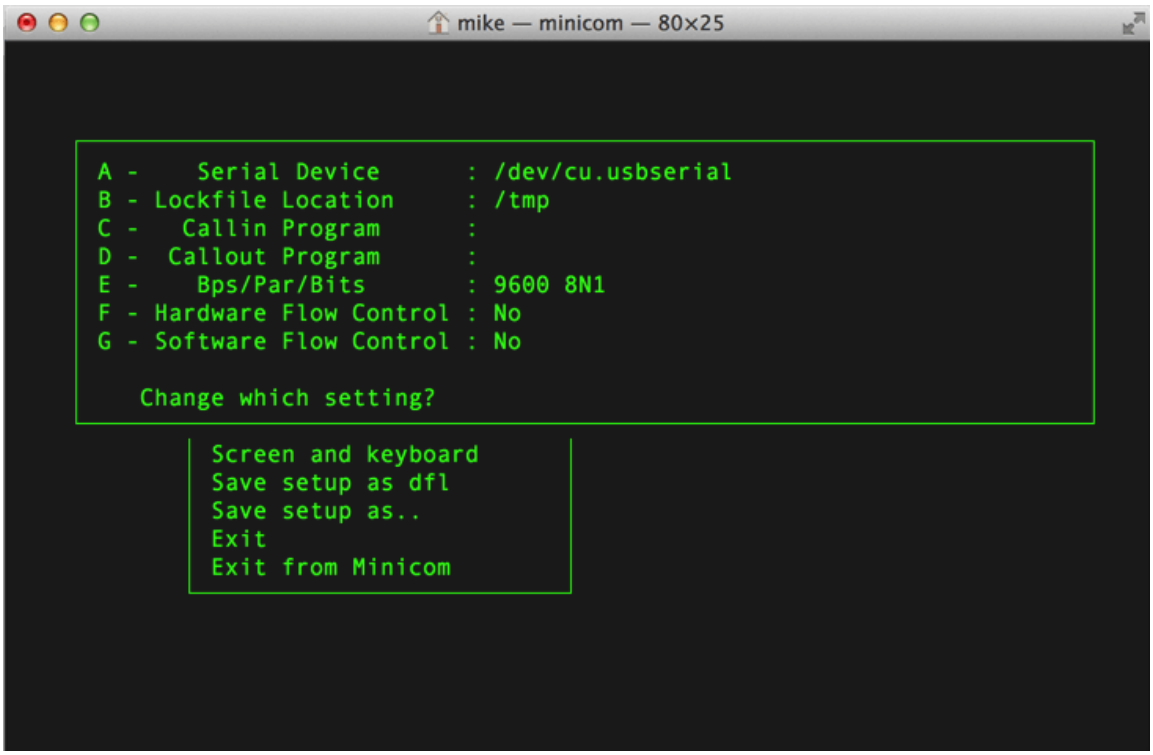


Figure 2: Minicom settings.

Go to *Serial port setup* > *Serial Device* and type in: **/dev/cu.usbmodem123456781** in the field and press enter. Then *Save setup as dfl*.
Now, to start using the dongle just run:

```
1.  minicom
```

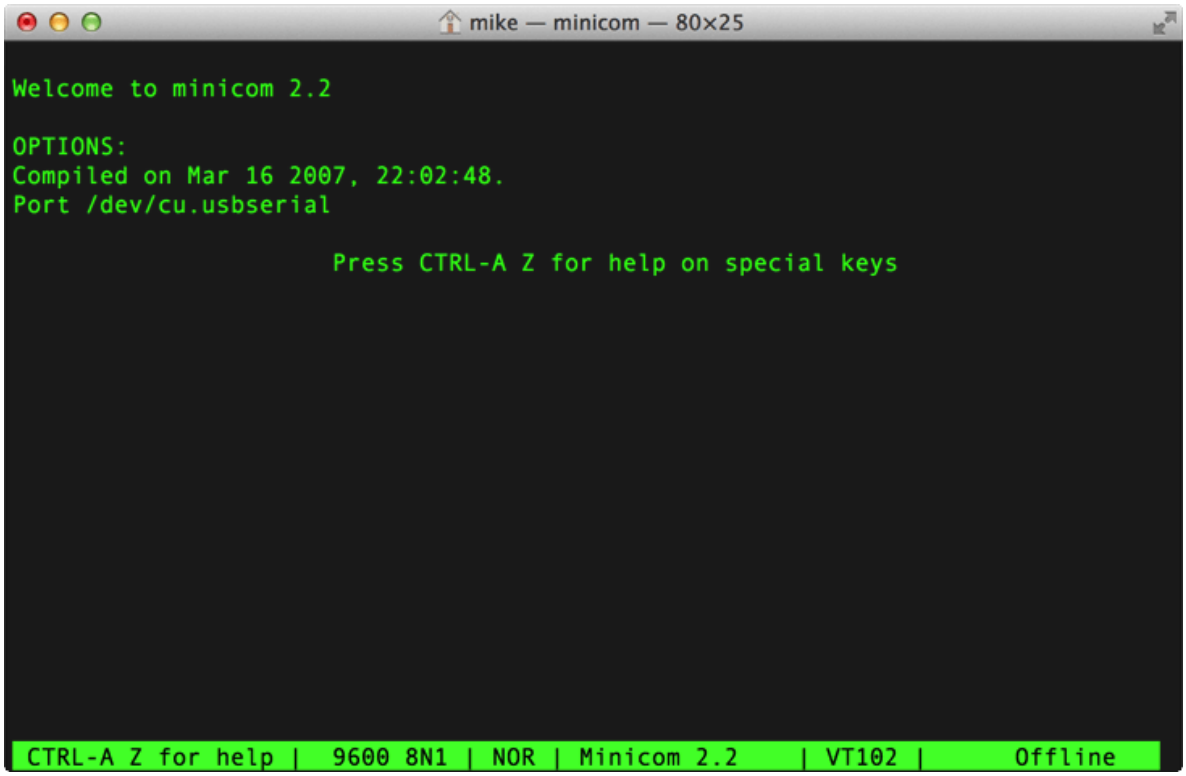


Figure 3: Minicom ready to use.

Step 4

Now try typing an AT-Command. For example **AT** .
If you get an OK response that means the dongle is working.

Done

Great job! You are now ready to start using Smart USB Dongle 2.0!

Now check out the different *AT Commands available* or how to get started *using scripts*.

Issues

Showing advertising and/or response data, either when setting or querying, has been temporarily disabled.

Firmware Updates

Introduction

The Smart USB Dongle 2.0 comes with a bootloader to allow you to update the firmware or flash your own application to the dongle. To flash the dongle you will need a image file containing the new firmware or your own application and a host USB loader application (that you can get
HERE for Windows (host_usb_updater.zip) or

HERE for Linux (host_usb_updaterlinux.zip)).

USB loader application for Mac is unfortunately not available at the moment.

Latest Firmware

[2020-06-30] v1.0.7 Firmware

Version 1.0.7	Download	Release Notes

Older Firmwares

Version 1.0.6	Download	Release Notes
Version 1.0.5	**Download**	**Release Notes**
Version 1.0.4	**Download**	**Release Notes**
Version 1.0.3	**Download**	**Release Notes**

How to update your firmware

Linux side

- Run with

```
sudo ./host_usb_updater /dev/ttyACM0 ./example_firmware.img
```

means USB-CDC driver of Linux. It can be changed according to test machine.

- Sometimes a modemmanager in Linux system like Ubuntu might interrupt the USB-CDC communication. In that case you can try disabling it.

Windows side

- Open the command prompt in the directory where the ***host_usb_updater.exe*** is located.
- Run with

```
host_usb_updater.exe 24 example_firmware.img -verbose
```

- The 24 means COM port number of USB-CDC device. You can see the COM port number on Windows device manager. COM port number may vary on different machines. - Debug message can be enabled by -verbose option.

NOTE: For simplicity's sake it is recommended to keep the image-file and ***host_usb_updater.exe*** in the same folder otherwise you will need to provide the path to where image file is stored relative to the ***host_usb_updater.exe***.

Troubleshooting

General issues

The Smart USB Dongle 2.0 is connected but doesn't show up.

- Try another USB port.
- Unplug all other devices connected to your USB ports and try connecting the dongle again.
- On older machines: Make sure that you have the correct CDC drivers installed.
- Try connecting it to another computer.

Bootloader issues

Bootloader fails to upload image file.

- Verify that you are trying to upload a valid image file.
- Try if you can upload a different image file.
- Make sure you are trying to upload the image file during the time window where the bootloader is open.
- Make sure you have closed down anything, like serial communication programs (e. g. TeraTerm or Minicom), that might be connected to the dongle before trying to upload a new image.
- For Linux users: Try disabling the modemmanager. As it may lock up the com port for awhile and not leaving enough time to use the boodloader before the main app starts up.

Firmware issues

I have flashed my own firmware but it doesn't start.

- Make sure you have changed the crystal setup in ***custom_config_qpsi.h*** as the Smart USB Dongle 2.0 doesn't have a 32KHz crystal:

Change

```
1.
    #define dg_configUSE_LP_CLK                     LP_CLK_32768 // 32KHz. Usually the default in most
     Dialog Semiconductor projects.
```

To

```
1.  #define dg_configUSE_LP_CLK                     LP_CLK_RCX // 16Khz.
```

AT COMMANDS

AT

Basic AT-Command.

Command	Syntax
AT	AT

Example:

```
AT
Executing 'AT'...

OK
```

ATI

Device information query. Returns firmware version, hardware type and unique organization identifier.

Command	Syntax
ATI	ATI

Example:

```
ATI
Executing 'ATI'...

OK
SSD005 Dongle DA14683 v 1.0.1
```

ATR

Trigger platform reset.

Command	Syntax
ATR	ATR

Example:

```
ATR
Executing 'ATR'...

OK
```

AT+ADVDATA

Sets or queries the advertising data. Data must be provided as hex string. The content will take effect only after advertising is restarted.
If you want to send several data types separate it with a space.

Command	Syntax
AT+ADVDATA	
	AT+ADVDATA=*xx:xx:xx:xx:xx:xx*
	AT+ADVDATA=*(xx:xx:xx:xx) (xx:xx:xx:xx:xx)*

Example:

```
AT+ADVDATA=04:09:43:41:54
Executing 'AT+ADVDATA=04:09:43:41:54'...


OK
```

AT+ADVDATAI

Sets advertising data in a way that let's it be used as an iBeacon.

Command	Syntax
AT+ADVDATAI	AT+ADVDATAI=*(UUID)(MAJOR)(MINOR)(TX)*

Example:

```
AT+ADVDATAI=ebbaaf47-0e4f-4c65-8b08-dd07c98c41ca0000000000

Executing 'AT+ADVDATAI=ebbaaf47-0e4f-4c65-8b08-dd07c98c41ca0000000000'...

OK
```

AT+ADVSTART

Starts advertising. Advertising interval can optionally be specified in milliseconds (100 to 3000ms). Time_ms controlls how for how long the dongle will advertise. Set to 0 for unlimited. Returns ERROR if advertising is already active or if the device is in central role.

Command	Syntax
AT+ADVSTART	AT+ADVSTART
	AT+ADVSTART=*(mode);(intv_min);(intv_max);(time_ms);*

Supported modes:

Code	Mode
0:	Non-connectable mode
1:	Undirected mode
2:	Directed mode
3:	Directed Low Duty Cycle mode

Example:

```
AT+ADVSTART=1;200;300;20;
Executing 'AT+ADVSTART'...

OK


ADVERTISING....
```

AT+ADVSTOP

Stops advertising. Returns ERROR if not already advertising.

Command	Syntax
AT+ADVSTOP	AT+ADVSTOP

Example:

```
AT+ADVSTOP
Executing 'AT+ADVSTOP'...

OK

STOPPING ADVERTISING...


ADVERTISING STOPPED.
```

AT+ADVRESP

Sets or queries scan response data. Data must be provided as hex string. Changes to take effect after restart of advertising.

Command	Syntax
AT+ADVRESP	
	AT+ADVRESP=*(xx:xx:xx:xx:xx:xx:xx)*

Example:

```
AT+ADVRESP=04:09:43:41:54
Executing 'AT+ADVRESP'...

OK
```

AT+CENTRAL

Sets the device Bluetooth role to central role. Advertising must be stopped and any connection must be terminated before the role change is accepted.

Command	Syntax
AT+CENTRAL	AT+CENTRAL

Example:

```
AT+CENTRAL
Executing 'AT+CENTRAL'...

OK
```

AT+GAPCONNECT

Initiates a connection with a specific slave device. Upon succesfully connecting atempts to display device name, advertising and response data and the services of the peripheral. The local device must be in central role.

Command	Syntax
AT+GAPCONNECT	AT+GAPCONNECT=*slave_address*

Example:

```
AT+GAPCONNECT=FD:37:13:D0:6D:02
Executing 'AT+GAPCONNECT=FD:37:13:D0:6D:02'...

OK

CONNECTING...
CONNECTED
```

AT+GAPDISCONNECT

Disconnects from a peer Bluetooth device. This command can be used in both central and peripheral role.

Command	Syntax
AT+GAPDISCONNECT	AT+GAPDISCONNECT

Example:

```
AT+GAPDISCONNECT
Executing 'AT+GAPDISCONNECT'...

OK

DISCONNECTED
```

AT+GAPSCAN

Starts a Bluetooth device scan with or without timer set in seconds. Only accepted when device is in central role and not connected. The scan will continue indefinitly in no parameters set or until amount of time set is reached. Scan will abort if user presses CTRL+C.

Command	Syntax
AT+GAPSCAN	AT+GAPSCAN
	AT+GAPSCAN=*seconds*

Example:

```
AT+GAPSCAN
Executing 'AT+GAPSCAN'...

OK

SCANNING...

[01] Device: [1]30:63:C5:D0:B1:DE  RSSI: -38
[02] Device: [0]D0:76:50:80:0A:98  RSSI: -75 (closebeacon.com)
[03] Device: [1]27:5D:B8:2E:96:B0  RSSI: -51
[04] Device: [1]5E:CE:CF:C5:20:BB  RSSI: -84


SCAN COMPLETE
```

AT+GAPSTATUS

Reports the Bluetooth role.

Command	Syntax
AT+GAPSTATUS	AT+GAPSTATUS

Example:

```
AT+GAPSTATUS
Executing 'AT+GAPSTATUS'...

OK

PERIPHERAL
```

AT+GATTCREAD

Read attribute of remote GATT server. Can only be used in Central role and when connected to a peripheral.

Command	Syntax
AT+GATTCREAD	AT+GATTCREAD=*handle param*

Example:

```
AT+GATTCREAD=001B
Executing 'AT+GATTCREAD=001B'...

OK

handle_evt_gattc_read_completed: conn_idx=0000 handle=001b status=0

Value read: HELLO
Hex: 0x48454C4C4F
Size: 5
```

AT+GATTCWRITE

Write attribute to remote GATT server in ASCII. Can only be used in Central role and when connected to a peripheral.

Command	Syntax
AT+GATTCWRITE	AT+GATTCWRITE=*(handle param) (msg)*

Example:

```
AT+GATTCWRITE=001B
Executing 'AT+GATTCWRITE=001B HELLO'...

OK

DATA WRITTEN: HELLO
```

AT+GATTCWRITEB

Write attribute to remote GATT server in Hex. Can only be used in Central role and when connected to a peripheral.

Command	Syntax
AT+GATTCWRITEB	AT+GATTCWRITEB=*(handle param) (msg)*

Example:

```
AT+GATTCWRITE=001B
Executing 'AT+GATTCWRITE=001B 0101'...

OK

DATA WRITTEN: 0101
Size: 2
```

AT+PERIPHERAL

Sets the device Bluetooth role to peripheral. Any connection must be terminated before the role change is accepted.

Command	Syntax
AT+PERIPHERAL	AT+PERIPHERAL

Example:

```
AT+PERIPHERAL
Executing 'AT+PERIPHERAL'...

OK
```

AT+SCANTARGET

Scan a target device. Displaying it's advertising data as it updates.

Command	Syntax
AT+SCANTARGET	AT+SCANTARGET=*(slave address)*

Example:

```
1.  AT+SCANTARGET=00:00:00:00:00:01
2.  Executing 'AT+SCANTARGET=00:00:00:00:00:01'...
3.
4.  OK
5.
6.  SCANNING...
7.
8.  [01] Device Data: DataXYZ
9.  [01] Device Data: DataXYZ
10. [01] Device Data: DataXYZ
11. [01] Device Data: DataXYZ
12. [01] Device Data: DataXYZ
```

AT+SPSSEND

Send a message or data via the SPS profile. Without parameters it opens a stream for continiously sending data.

Command	Syntax
AT+SPSSEND	AT+SPSSEND=*(message)*
	AT+SPSSEND

Example:

```
1.  AT+SPSSEND=HELLO
2.  Executing 'AT+SPSSEND=HELLO'...
3.
4.  OK
```

--H

- ***Shows all AT-Commands.***

Command	Syntax
--H	--H

Example:

```
--H

AT                                  [A] Basic AT-Command.
ATI                                 [A] Device information query.
AT+ADVDATA                          [P]
...
```

Scripts & Tutorials

Getting Started

To use these scripts you will need to have **Python installed**. Both Python2 and Python3 should work but the scripts are made with Python3 in mind. But there are comments where the script need to be edited to work with Python2. You will also need to install the module **pySerial**. The easiest way to install it is through **pip** (which you should already have after installing Python) by running:
Python2:

```
1.  pip install pyserial
```

Python3:

```
1.  python3 -m pip install pyserial
```

Now you can download the script you want to use, from our GITHUB PAGE. Then open up the command prompt inside the directory where the script you want to run is. Then run:

```
1.  python scriptname.py
```

NOTE: You will need to manually change the COM port in the script to which ever your dongle uses. The number may vary from machine to machine.

Follow the video tutorial to learn how to get started using the Smart USB Dongle 2.0.

https://www.youtube.com/watch?v=6EYSGS0oZ3s

iBeacon

Introduction

iBeacon technology allows Mobile Apps to understand their position on a micro-local scale, and deliver content to users based on location. It is a Bluetooth Low Energy technology.

BLE Advertising uses a one-way communication method. Beacons that want to be discovered can Advertise self-contained packets of data in set intervals. Smartphones collect these packets, which can be used for various applications to trigger things like push messages, prompts or app actions.

Beacons ideal for indoor location tracking because a standard BLE has a broadcast range of up to 100 meters. With an iBeacon network, any retailer, app, platform or brand will be able to locate a customer and send contents and advertisements to customers on their smartphones.

You can set up your own IBeacon easily using Smart USB dongle 2.0 and Python script. This device works equally well in Windows 10, Linux or macOS.

iBeacon are defined by the Apple company including following parameters: UUID, Major and Minor

You can build your own beacon by defining your own parameter values.

What is UUID?

UUID stands for Universally Unique Identifier. It contains 32 hexadecimal digits, split into 5 groups, like this:

```
5f2dd896-b886-4549-ae01-e41acd7a354a0203010400
```

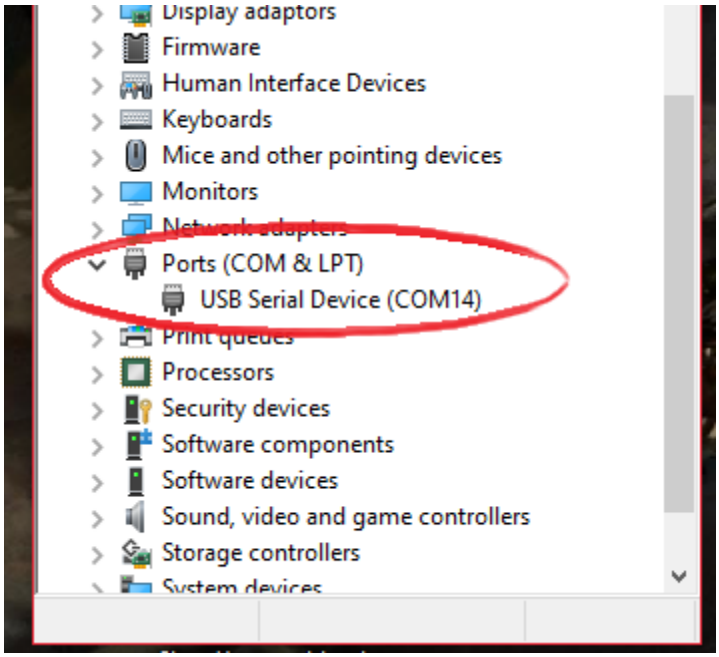
The UUID is a standard identifying system which allows a 'unique' number to be generated for a beacon network. The purpose of the UUID is to identify iBeacons in your network, from all other possible beacons in networks not in your control.

What are Major and Minor values?

Major and Minor values are numbers assigned to your iBeacons, in order to identify individual iBeacon within your UUID network. Minor and Major are unsigned integer values between 0 and 65535. The iBeacon standard requires both a Major and Minor value to be assigned.

How to use

Connect the Bluetooth USB Adapter to your computer. It opens a virtual serial port (COM port) that you can use to send commands to and from the Bluetooth USB Adapter.



Control Bluetooth USB Adapter using predefined commands. *List of AT Commands*
The Smart USB Dongle 2.0 includes a bootloader which allows you to easily update the firmware (or flash your own application to the dongle)

After connecting, you can use the following sample python script to set up your own iBeacon.

```
import serial
import time

connecting_to_dongle = 0
print("Connecting to dongle...")
# Trying to connect to dongle until connected. Make sure the port and baudrate is the same as your dongle.
# You can check in the device manager to see what port then right-click and choose properties then the Port Settings
# tab to see the other settings
while connecting_to_dongle == 0:
    try:
        console = serial.Serial(
                port='COM14',
                baudrate=57600,
                parity="N",
                stopbits=1,
                bytesize=8,
                timeout=0
                )
        if console.is_open.__bool__():
            connecting_to_dongle = 1
    except:
        print("Dongle not connected. Please reconnect Dongle.")
        time.sleep(5)


print("\n\nConnected to Dongle.\n")
print("\n Welcome to the iBeacon example!\n\n")


new_input = 1
while 1 and console.is_open.__bool__():
    # get keyboard input once
    if (new_input == 1):
        # Python 2 users
        # input = raw_input("Enter the UUID... ")
        new_input = input("Enter the UUID (x) string with Major (j), Minor (n) and TX (t) (format: "
                          "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxxjjjjnnnntt): ")
        time.sleep(0.1)
        # sends the commands to the dongle. Important to send the \r as that is the return-key.
        console.write(str.encode("AT+ADVDATAI="))
        console.write(new_input.encode())
        console.write('\r'.encode())
        time.sleep(0.1)
        console.write(str.encode("AT+ADVSTART=0;200;3000;0;"))
        console.write('\r'.encode())
        out = ''
        # let's wait one second before reading output (let's give device time to answer)
        time.sleep(1)
    while console.inWaiting() > 0:
        out += console.read(console.inWaiting()).decode()
    else:
        if not out.isspace():
            # We make sure it doesn't print the same message over and over again by setting [out] to blankspace
            # after printing once and check for blankspace before print again
            print(">>" + out)
            out = "  "
```

Full source also available on **GitHub**.

Youtube Tutorial

Follow the Youtube tutorial.

https://www.youtube.com/watch?v=CixqMR3leGs

Eddystone Beacon

Getting started

Copy the following script and save it as **eddystone_example.py** on your local directory. You can also download the source code from our **GitHub** page.

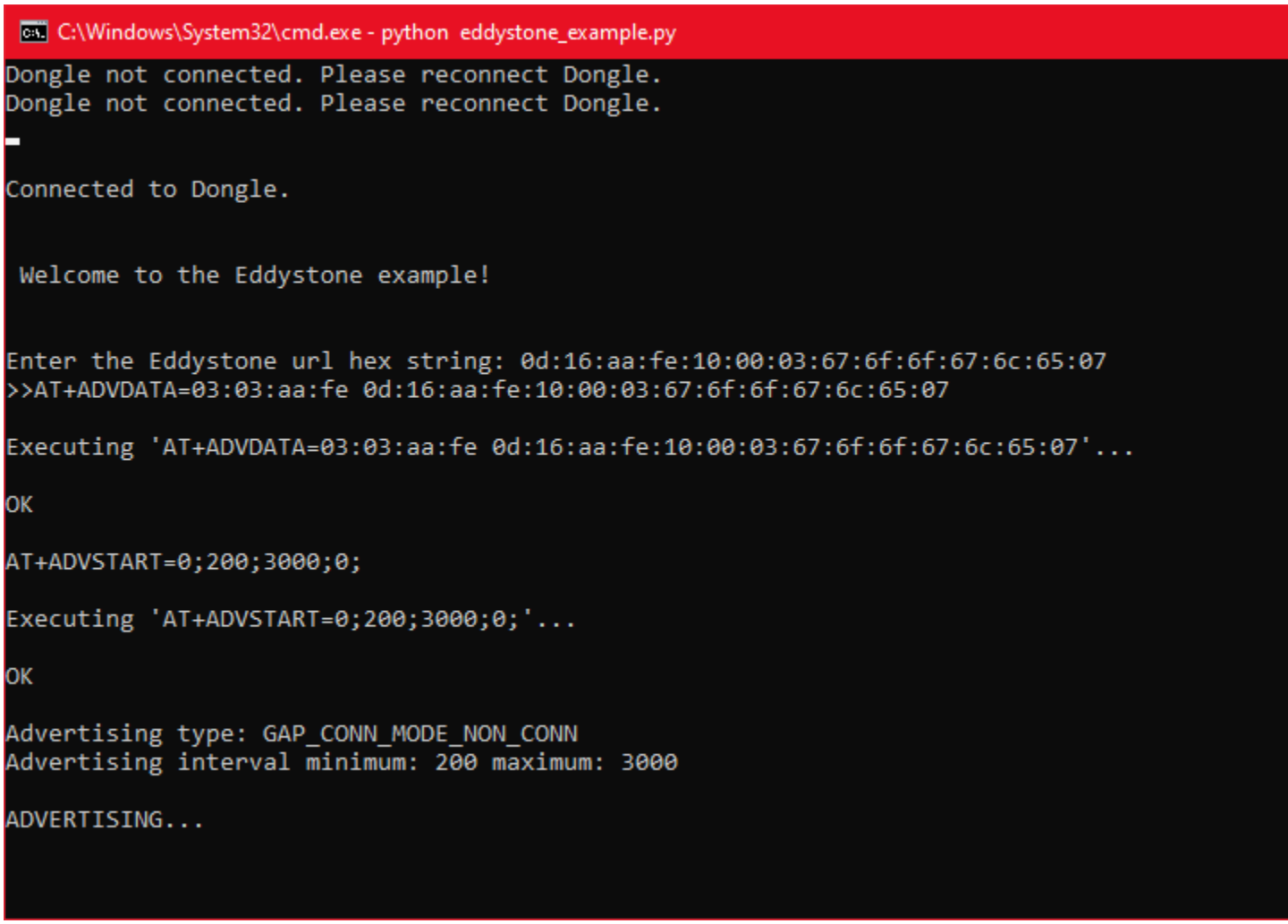
```
1.  import serial
2.  import time
3.
4.  connecting_to_dongle = 0
5.  print("Connecting to dongle...")
6.  # Trying to connect to dongle until connected. Make sure the port and baudrate is the same as your
     dongle.
7.  # You can check in the device manager to see what port then right-
    click and choose properties then the Port Settings
8.  # tab to see the other settings
9.  while connecting_to_dongle == 0:
10.     try:
11.         console = serial.Serial(
12.                 port='COM14',
13.                 baudrate=57600,
14.                 parity="N",
15.                 stopbits=1,
16.                 bytesize=8,
17.                 timeout=0
18.                 )
19.         if console.is_open.__bool__():
20.             connecting_to_dongle = 1
21.     except:
22.         print("Dongle not connected. Please reconnect Dongle.")
23.         time.sleep(5)
24.
25.
26. print("\n\nConnected to Dongle.\n")
27. print("\n Welcome to the Eddystone example!\n\n")
28.
29.
30. new_input = 1
31. while 1 and console.is_open.__bool__():
32.     # get keyboard input once
33.     if (new_input == 1):
34.         # Python 2 users
35.         # input = raw_input("Enter the Eddystone url hex string: ")
36.         new_input = input("Enter the Eddystone url hex string: ")
37.         time.sleep(0.1)
38.         # sends the commands to the dongle. Important to send the \r as that is the return-key.
39.         console.write(str.encode("AT+ADVDATA=03:03:aa:fe "))
40.         console.write(new_input.encode())
41.         console.write('\r'.encode())
42.         time.sleep(0.1)
43.         console.write(str.encode("AT+ADVSTART=0;200;3000;0;"))
44.         console.write('\r'.encode())
45.         out = ''
46.         # let's wait one second before reading output (let's give device time to answer)
47.         time.sleep(1)
48.     while console.inWaiting() > 0:
49.         out += console.read(console.inWaiting()).decode()
50.     else:
51.         if not out.isspace():
52.             # We make sure it doesn't print the same message over and over again by setting [out]
    to blankspace
53.             # after printing once and check for blankspace before print again
54.             print(">>" + out)
```

```
55.             out = "  "
```

Then go to the directory you saved the script and open a command propt. Then run the script by typing:

```
1.  python eddystone_example.py
```

The script will now ask you for a eddystone url hex string.



I will now show you how to prepare one.

Eddystone url hex string

Let's say we want to send **http://google.com**
The url hex string will then looks like this: **0d:16:aa:fe:10:00:03:67:6f:6f:67:6c:65:07**
0d - Length of the following string in hex. **0D** is 13 and as you see there is 13 bits after this one.
It is important that this correspondes with your length otherwise it will not run.

Keep in mind that you have a maximum of 17 bytes worth of space for your url.

Hex	Description
16	Service Data data type value (No need to change this)
aa	16-bit Eddystone UUID (No need to change this)
fe	16-bit Eddystone UUID (No need to change this)
10	Frame Type = URL (No need to change this)

Hex	Description
00	TX Power.
03	https:// ***(Eddystone has a list of 1 byte codes for popular prefixes and sufixes)***
67	hex for 'g'
6f	hex for 'o'
6f	hex for 'o'
67	hex for 'g'
6c	hex for 'l'
65	hex for 'e'
07	.com (Eddystone has a list of 1 byte codes for popular prefixes and sufixes)

Here is two helpful sites that can help you autogenerate an eddystone hex url string:

https://www.mkompf.com/tech/eddystoneurl.html
https://yencarnacion.github.io/eddystone-url-calculator/
Just keep in mind that you will still need to put a colon **(:)** between every byte when you put it in the script for it to work.

Supported prefix list

Hex	Expansion
00	http://www.
01	https://www.
02	http://
03	https://

Supported sufix list

Hex	Expansion
00	.com/
01	.org/
02	.edu/
03	.net/
04	.info/
05	.biz/
06	.gov/
07	.com
08	.org
09	.edu

Hex	Expansion
0a	.net
0b	.info
0c	.biz
0d	.gov
0e to 20	Reserved for Future Use
7F to FF	Reserved for Future Use

Full source also available on **GitHub**.

Scan Example

Scan example

In this tutorial, we're going to set up the dongle using Python script to scan for nearby Bluetooth devices. For a quick setup, copy the following script and save it on your local directory. You can also get the source code from our **GitHub**. page.

```
1.  import serial
2.  import time
3.
4.  connecting_to_dongle = 0
5.  print("Connecting to dongle...")
6.  # Trying to connect to dongle until connected. Make sure the port and baudrate is the same as your
     dongle.
7.  # You can check in the device manager to see what port then right-
    click and choose properties then the Port Settings
8.  # tab to see the other settings
9.  while connecting_to_dongle == 0:
10.     try:
11.         console = serial.Serial(
12.                 port='COM14',
13.                 baudrate=57600,
14.                 parity="N",
15.                 stopbits=1,
16.                 bytesize=8,
17.                 timeout=0
18.                 )
19.         if console.is_open.__bool__():
20.             connecting_to_dongle = 1
21.     except:
22.         print("Dongle not connected. Please reconnect Dongle.")
23.         time.sleep(5)
24.
25.
26. print("\n\nConnected to Dongle.\n")
27. print("\nWelcome to the Bluetooth device Scanning example!\n\n")
28.
29.
30. new_input = "NEW-INPUT"
31. while 1 and console.is_open.__bool__():
32.     # get keyboard input once
33.     if (new_input == "NEW-INPUT"):
34.         # Python 2 users
35.         # input = raw_input("Select:\n1) If you... ")
36.         new_input = input("Select:\n1) If you'd like to scan for devices without a timer to stop.\
    n2)"
37.                           " If you'd like to scan for devices for a selected period of time.\n"
38.                           "3) If you'd like to scan a specific device.\n>>")
39.         if new_input == "1":
40.             time.sleep(0.1)
41.             # sends the commands to the dongle. Important to send the \r as that is the return-
    key.
42.             console.write(str.encode("AT+CENTRAL"))
43.             console.write('\r'.encode())
44.             time.sleep(0.1)
45.             console.write(str.encode("AT+GAPSCAN"))
46.             console.write('\r'.encode())
47.         elif new_input == "2":
48.             time.sleep(0.1)
49.             # sends the commands to the dongle. Important to send the \r as that is the return-
    key.
50.             console.write(str.encode("AT+CENTRAL"))
51.             console.write('\r'.encode())
```

```
            input_time = input("Please select amount of time the scanning should continue: ")
            while not input_time.isdigit():
                input_time = input("Sorry, unacceptable time.\n"
                                   "Please select amount of time the scanning should continue: ")

            console.write(str.encode("AT+GAPSCAN="))
            console.write(input_time.encode())
            console.write('\r'.encode())
        elif new_input == "3":
            time.sleep(0.1)
            # sends the commands to the dongle. Important to send the \r as that is the return-key.
            console.write(str.encode("AT+CENTRAL"))
            console.write('\r'.encode())
            time.sleep(0.1)
            input_adress = input("Please enter type ([0] or [1]) and the address (xx:xx:xx:xx:xx:xx) of the device you "
                                 "\nwish to scan (format:[x]xx:xx:xx:xx:xx:xx): ")
            console.write(str.encode("AT+SCANTARGET="))
            console.write(input_adress.encode())
            console.write('\r'.encode())
        else:
            print("That was not a choice. Please choose one of the options.")
            new_input="NEW-INPUT"
        # let's wait one second before reading output (let's give device time to answer)
        time.sleep(1)
        out = ""
    while console.inWaiting() > 0:
        out += console.read(console.inWaiting()).decode()
    else:
        if not out.isspace():
            # We make sure it doesn't print the same message over and over again by setting [out] to blankspace
            # after printing once and check for blankspace before print again
            print(out + " ")
            out = " "
```

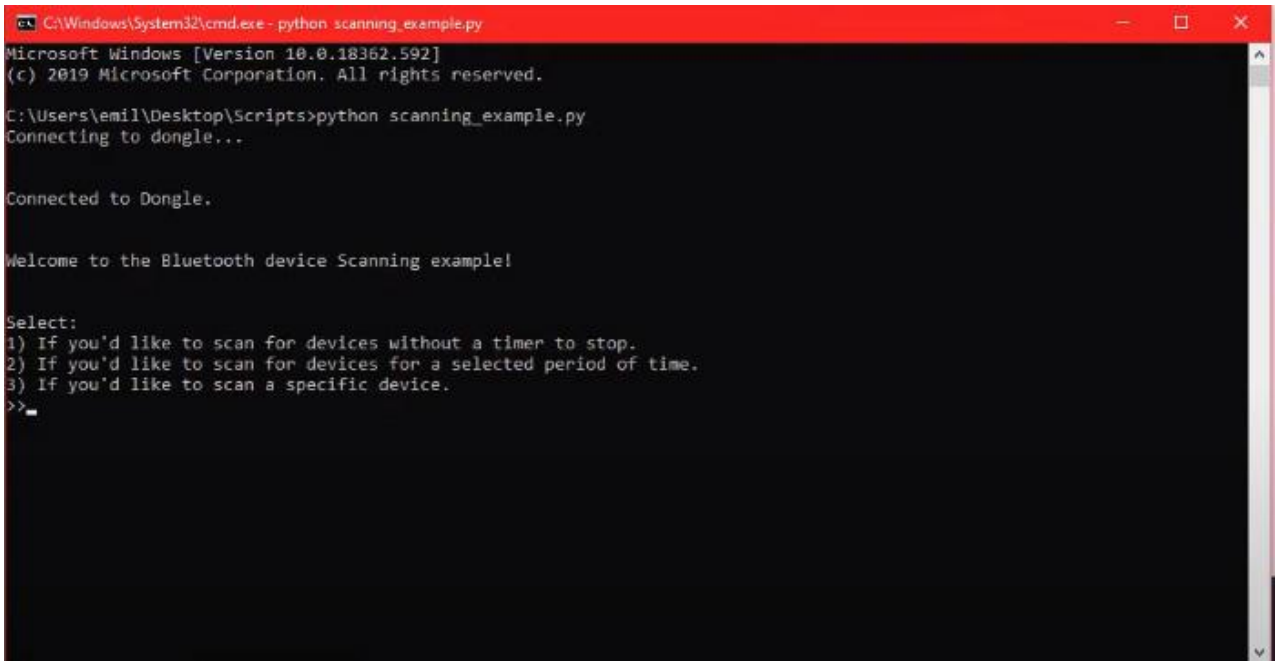
Open up the command prompt in the directory where the script is located. Start the script by typing

```
python scriptname.py
```

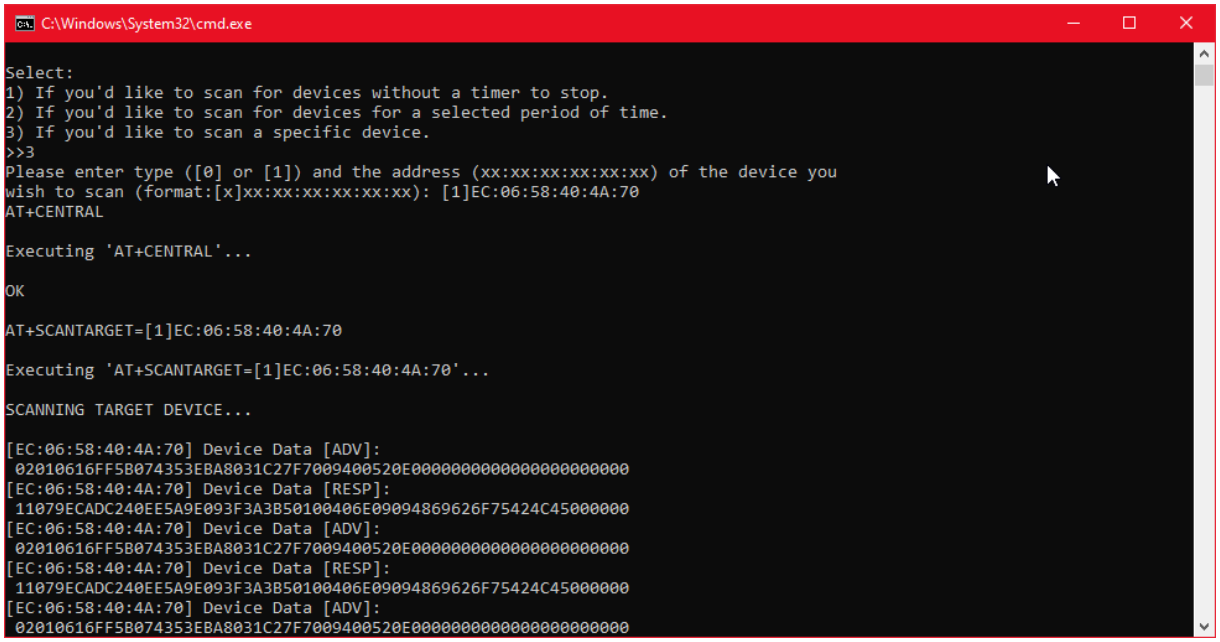
and press Enter.

You should now be prompted to choose between three options.

- Scan without time limit. (requires no input)
- Scan with a time limit. (requires an input a number representing the time in seconds.)
- Scan a specific devices advertising data. (requires an address to the desired Bluetooth device which we can get from the other two options)



Option 3 will show the full advertising and response data of 31 bytes from the selected device.



And that is how we scan for Bluetooth devices. If you want to stop the script, you can simply press control C.

Youtube Tutorial

Follow the Youtube tutorial.

https://www.youtube.com/watch?v=oidvEd5QoXg

Scan and Store Example

Scan and Store example

In this tutorial, we're going to set up the dongle using Python script to scan for nearby Bluetooth devices with a specific Manufacturer Specific (MFS) ID and then store the results together with a timestamp and the MAC Address in a simple text file. For a quick setup, copy the following script and save it on your local directory. You can also get the source code from our **GitHub**. page.

```
1.  import serial
2.  import time
3.  from datetime import datetime
4.
5.  # Name of file that will be created and store the saved data
6.  file_name = "SavedData.txt"
7.  connecting_to_dongle = 0
8.  print("Connecting to dongle...")
9.
10. # Trying to connect to dongle until connected. Make sure the port and baudrate is the same as your
     dongle.
11. # You can check in the device manager to see what port then right-
    click and choose properties then the Port Settings
12. # tab to see the other settings
13. while connecting_to_dongle == 0:
14.     try:
15.         console = serial.Serial(
16.                 port='COM14',
17.                 baudrate=57600,
18.                 parity="N",
19.                 stopbits=1,
20.                 bytesize=8,
21.                 timeout=0
22.                 )
23.         if console.is_open.__bool__():
24.             connecting_to_dongle = 1
25.     except:
26.         print("Dongle not connected. Please reconnect Dongle.")
27.         time.sleep(5)
28.
29.
30. print("\n\nConnected to Dongle.\n")
31. print("\n Welcome to the Scan and Store example!\n\n")
32.
33. # Method for parsing and writing to file
34. def write_data_to_file(out_data):
35.     addr_string = ""
36.     data_string = ""
37.     now = datetime.now() # Generating a timestamp
38.     current_time = now.strftime("%H:%M:%S") # Formatting the timestamp
39.     out_data = out_data.replace('\r','') # Remove return.
40.     out_data = out_data.replace('\n','') # Remove new line.
41.     fo = open(file_name, "a")
42.     for i in range(2,21): # Reading the MAC-Address and saving it into addr_string
43.         addr_string += out_data[i]
44.     for x in range(41, 102): # Here the advertising/response data gets stored in the data_string v
    ariable
45.         data_string += out_data[x]
46.     fo.write("{")
47.     fo.write("["+current_time+"]")
48.     fo.write(addr_string)
49.     fo.write(":")
50.     fo.write(data_string)
```

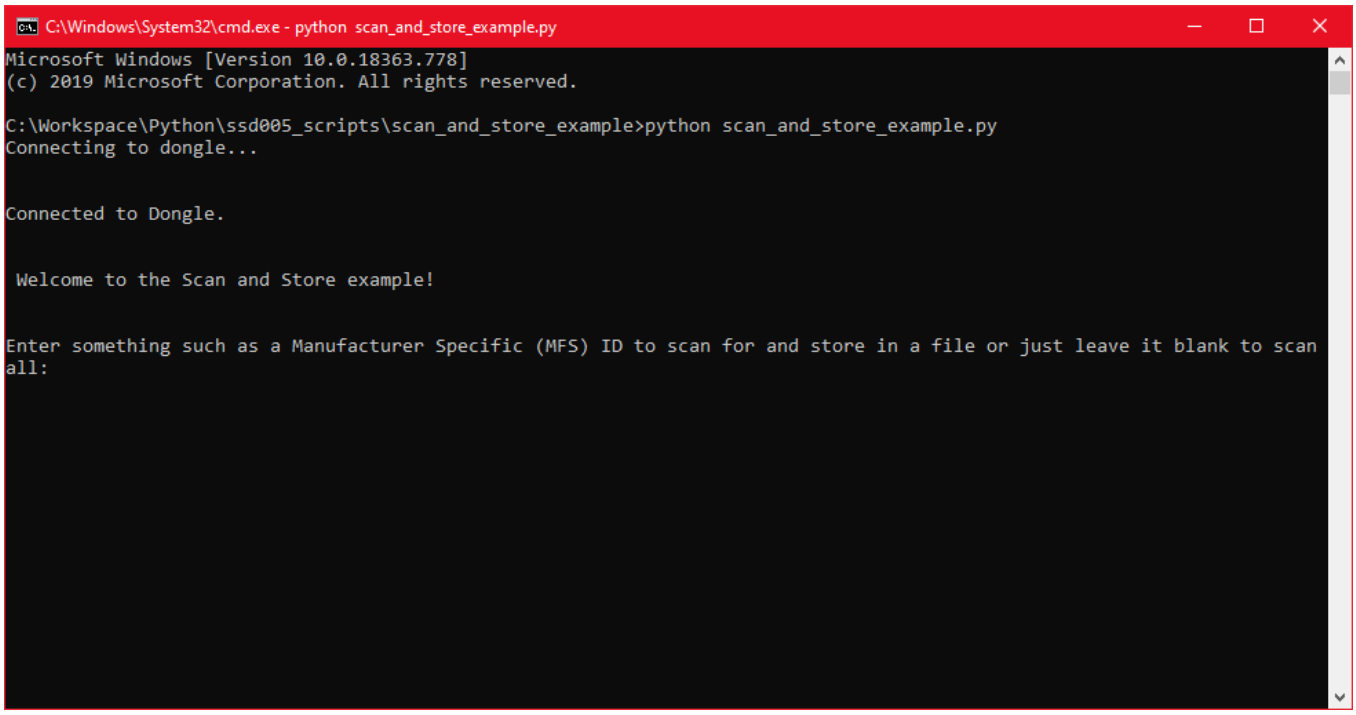
```
    fo.write("}\n")
    fo.close()


new_input = 1
try:
    while 1 and console.is_open.__bool__():
        # get keyboard input once
        if (new_input == 1):
            # Python 2 users
            # input = raw_input("Enter something such as a Manufacturer Specific (MFS) ID to scan for and store in a file or just leave it blank to scan all: ")
            new_input = input("Enter something such as a Manufacturer Specific (MFS) ID to scan for and store in a file or just leave it blank to scan all: ")
            time.sleep(0.1)
            # sends the commands to the dongle. Important to send the \r as that is the return-key.
            console.write(str.encode("AT+CENTRAL"))
            console.write('\r'.encode())
            time.sleep(0.1)
            console.write(str.encode("AT+FINDSCANDATA="))
            console.write(new_input.encode())
            console.write('\r'.encode())
            out = ''
            # Let's wait one second before reading output (let's give device time to answer)
            time.sleep(1)
            print("\nCollecting data...\nPress Ctrl-C to stop.")
        while console.inWaiting() > 0:
            out += console.read(console.inWaiting()).decode()
        else:
            if not out.isspace():
                # We make sure it doesn't print the same message over and over again by setting [out] to blankspace
                # and check for blankspace and that [out] isn't anything else before writing to file
                if not out.__contains__("AT+") and not len(out) <= 106:
                    write_data_to_file(out)
                out = "  "
except KeyboardInterrupt:
    exit()
```

Open up the command prompt in the directory where the script is located. Start the script by typing

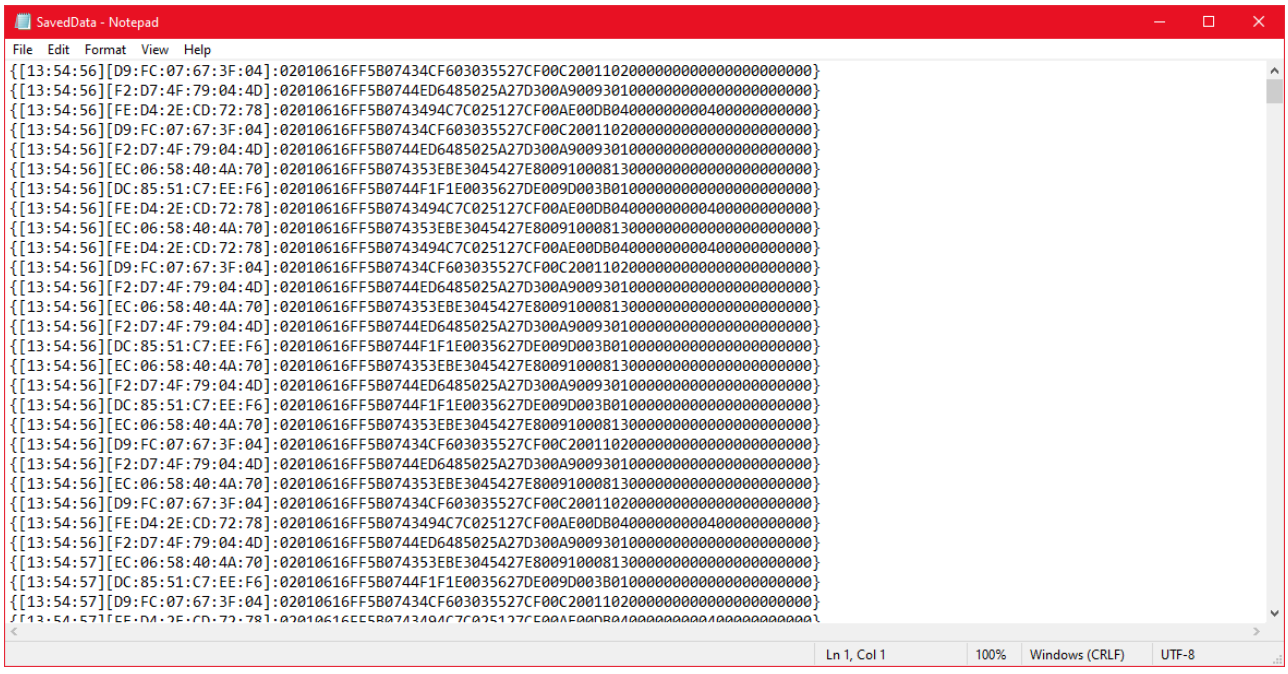
```
python scan_and_store_example.py
```

and press Enter.

You should now be prompted to enter what you want to scan for. For example a Manufacturer Specific ID (if you want to scan for Smart Sensor Devices products for example you enter the Flag for MFS Data **FF** and then our MFS ID: **5B07**).



The script will then generate a *.txt file in the same directory with the name entered as the ***file_name*** variable in the script. For example: ***"SavedData.txt"***.
The script will run until you stop it. You can then have a look in the text file...



Note that the script will not work if the text file is open or altered during runtime.

And that is how we store data that we scanned from Bluetooth devices. If you want to stop the script, you can simply press control C.

SPS Script

SPS Echo example

In this tutorial, we're going to set up two dongles a using Python script to send data back and forth between them.

In short:

1. One dongle will take on the Central role and the other will take on the Peripheral role.
2. Then they will connect to each other.
3. The Central dongle will then start off sending a message; "Echo".
4. The Peripheral dongle will then receive the message and send it back to the Central dongle which in turn will receive it and send it back and so forth until the script is stopped.

For a quick setup, copy the following script and save it on your local directory. You can also get the source code from our **GitHub**. page.

```
1.  import serial
2.  import time
3.
4.  target_dongle_mac_address = "[0]40:48:FD:E5:2D:05" # Change this to the peripheral's mac address.
5.  your_com_port = 'COM14' # Change this to the com port your dongle is connected to.
6.
7.  # Global
8.  connecting_to_dongle = True
9.  counter = 0
10. msg = ""
11. latest_msg = ""
12. error_counter = 0
13.
14. print("Connecting to dongle...")
15. # Trying to connect to dongle until connected. Make sure the port and baudrate is the same as your
     dongle.
16. # You can check in the device manager to see what port then right-
    click and choose properties then the Port Settings
17. # tab to see the other settings
18. while connecting_to_dongle:
19.     try:
20.         console = serial.Serial(
21.             port=your_com_port,
22.             baudrate=57600,
23.             parity="N",
24.             stopbits=1,
25.             bytesize=8,
26.             timeout=0
27.         )
28.         if console.is_open.__bool__():
29.             connecting_to_dongle = False
30.     except:
31.         print("Dongle not connected. Please reconnect Dongle.")
32.         time.sleep(5)
33.
34. # This script will send data from one dongle to another which in turn will echo it back and forth
    between the dongles.
35. print("\n\nConnected to Dongle.\n")
```

```
print("\nWelcome to the Serial Port Service (SPS) example!\n\n")
print("\nRemember to setup the Peripheral dongle first so Central has something to connect to.\n\n")

# Python 2 users
# input = raw_input("Choose \n1) for Peripheral...
role_input = input("Choose: \n1) for Peripheral Role\n2) for Central role\n>> ")
while not (role_input == "1" or role_input == "2"):
    role_input = input("Please choose 1 or 2.\nChoose: \n1) for Peripheral Role \n2) for Central role\n>>  ")

connected = "0"
while 1 and console.is_open.__bool__():
    if role_input == "1":
        print("Starting advertising.")
        time.sleep(0.1)
        # Sends the commands to the dongle. Important to send the \r as that is the return-key.
        console.write(str.encode("AT+ADVSTART"))
        console.write('\r'.encode())
        while connected == "0":
            dongle_output = console.read(console.in_waiting)
            time.sleep(2)
            print("Awaiting connection to Central...")
            if not dongle_output.isspace():
                # We make sure it doesn't print the same message over and over again by resetting [out] to blankspace
                # after printing once and check for blankspace before print again
                print(dongle_output.decode())
                if dongle_output.__contains__(str.encode("\r\nCONNECTED.\r\n")):
                    # Opens Serial Stream
                    console.write(str.encode("AT+SPSSEND"))
                    console.write('\r'.encode())
                    connected = "1"
                    print("Connected!")
                dongle_output = " "
    elif role_input == "2":
        # This is what will be sent back and forth between the dongles.
        # You can change this message to whatever you like.
        msg = "Echo"
        # Sends the commands to the dongle. Important to send the \r as that is the return-key.
        console.write(str.encode("AT+CENTRAL"))
        console.write('\r'.encode())
        time.sleep(0.1)
        print("Putting dongle in Central role and trying to connect to other dongle.")
        while connected == "0":
            # Sends the commands to the dongle. Important to send the \r as that is the return-key.
            time.sleep(0.5)
            console.write(str.encode("AT+GAPCONNECT="))
            console.write(str.encode(target_dongle_mac_address))
            console.write('\r'.encode())
            dongle_output2 = console.read(console.in_waiting)
            time.sleep(2)
            print("Trying to connect to Peripheral...")
            if not dongle_output2.isspace():
                # We make sure it doesn't print the same message over and over again by resetting [out] to blankspace
                # after printing once and check for blankspace before print again
                print(dongle_output2.decode())
                if dongle_output2.decode().__contains__("CONNECTED."):
                    # Opens Serial Stream
                    console.write(str.encode("AT+SPSSEND"))
                    console.write('\r'.encode())
                    connected = "1"
                    print("Connected!")
                    time.sleep(2)
                    # We wait a bit then sends the starting msg to the other dongle.
                    console.write(str.encode(msg))
```

```
                    counter += 1
                dongle_output2 = " "
    while connected == "1":
        dongle_output3 = console.read(console.in_waiting)
        # Let's wait 2 seconds to make sure we have had a chance of receiving anything before proc
    eeding
        time.sleep(2)
        if not dongle_output3.isspace():
            # Here we check if we receive a message.
            if dongle_output3.__contains__(str.encode("\r\n[Received]: ")):
                print(dongle_output3)
                # Sometimes some extra bytes will be sent along with the message that the decode()
     function can't handle
                # so we put it in a try-
    except block so that it wont crash the script. We also save the last message as
                # a fallback in case of error.
                try:
                    # Here we convert the msg to a string and strip it of all newlines and carriag
    e returns.
                    msg = str(dongle_output3.decode("ascii"))
                    msg = msg.replace('[Received]: ', '')
                    msg = msg.replace("\r\n", "")
                    latest_msg = msg
                except:
                    # In case of an error we use the last received message instead and increment t
    he error counter.
                    msg = latest_msg
                    error_counter += 1
                print("This is what we will send: " + msg)
                # Encodes the string msg than sends it back to the other dongle.
                console.write(str.encode(msg))
                counter += 1
                # Just a little counter to show how many messages we've sent.
                print(str(counter) + " message(s) sent. " + str(error_counter) + " error(s).")
                # In case we get disconnected from the other dongle we go back to the previous sta
    te of waiting for a
                # connection (Peripheral) / trying to reconnect (Central).
            if dongle_output3.__contains__(str.encode("\r\nDISCONNECTED.")):
                print("Disconnected!")
                # Send an Escape to abort stream.
                console.write(0x1B)
                connected = "0"
            dongle_output3 = ""
        msg = ""
```

Connect a dongle each into two computers that has Python installed.

Change the COM-port in the script on both computers to match the ones your dongles is actually connected to. For the one that will be set up as the Central dongle you will also need to change the target_dongle_mac_address variable to the MAC address of your Peripheral dongle.

You can get the MAC address by scanning for the other dongle while it's advertising. To learn more about how to scan check out our **scanning tutorial**.
Open up the command prompt, on both computers, in the directory where the script is located. Start the script by typing

```
1.  python sps_example.py
```

and press Enter.

You should now be prompted to enter 1 or 2 depending on what role the dongle should have. Set one as Peripheral and the other as Central. It is advisable to setup the Peripheral first as the Central will need someone to connect to.

You should now see in the terminal how the dongles send and recieve data to and from each other.

The script will run until you stop it.

And that is an example of how we can send data between two dongles. If you want to stop the script, you can simply press control C.

Full source also available on **GitHub**.