

piBeacon - DIY Beacon with a Raspberry Pi

Created by Kevin Townsend



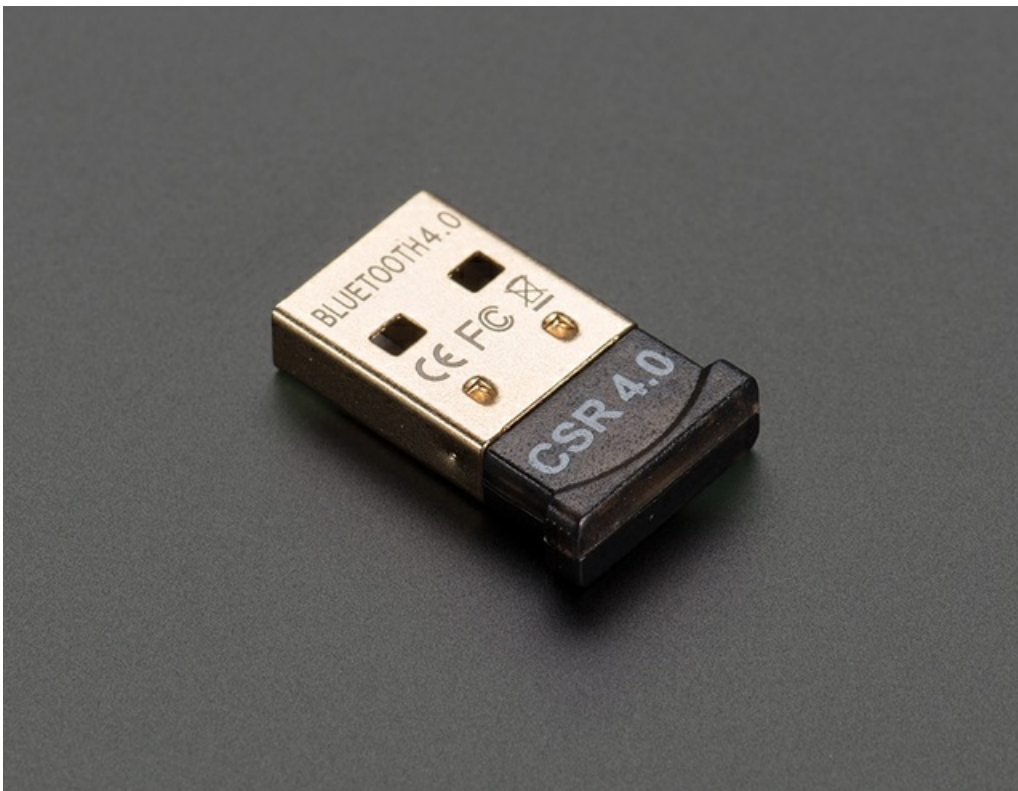
Last updated on 2018-08-22 03:38:45 PM UTC

Guide Contents

| | |
|--------------------------------------|----|
| Guide Contents | 2 |
| Overview | 3 |
| What You'll Need | 3 |
| Acknowledgement | 4 |
| What is a 'Beacon'? | 5 |
| How Does it Work in Practice? | 5 |
| Compiling Bluez | 6 |
| Check if you already have Bluez | 6 |
| 1. Install Required Libraries | 6 |
| 2. Download Bluez | 7 |
| 3. Unzip and Compile Bluez | 8 |
| 4. Insert the USB Module and Reset | 9 |
| Adding Beacon Data | 10 |
| 1. Check for your USB Module | 10 |
| 2. Enable the USB Device | 10 |
| 3. Enter the Beacon Advertising Data | 11 |
| Results on a Bluetooth Debugger | 11 |
| Testing it on iOS | 12 |



This learning guide will show you how you can take your Raspberry Pi (or almost any Linux-based device with a bit of poking and prodding) and turn it into an Beacon node using our [Bluetooth 4.0 USB Module](http://adafru.it/1327) and the open source Bluez stack.



What You'll Need

- [A Raspberry Pi](http://adafru.it/998) (any model should be OK)
- [A Bluetooth 4.0 USB Module](http://adafru.it/1327) (not every module works with Bluez, though ours definitely does!)
- A iOS 7.0 based device (recent iPhone/iPad/iPod Touch) to test with

- An iBeacon reader app from the App Store to test with (Try [Locate Beacon \(https://adafru.it/rnD\)](https://adafru.it/rnD), but any free iBeacon watcher ought to be OK!)

Acknowledgement

A big thanks to Tony Smith at The Register for putting together his [tutorial \(https://adafru.it/cYr\)](https://adafru.it/cYr) on configuring Bluez to transmit Beacon data!

What is a 'Beacon'?

'Beacons' are based on Bluetooth Low Energy (part of the new Bluetooth 4.0 standard), and at its heart is a way to advertise location specific data one-way, or provide basic indoor navigation via individual Beacon nodes.

The way it works is actually very simple. Any BLE device typically advertises a certain amount of data to let other devices (like your phone) know that they exist and they're available. The advertising packet that these devices send out might include information like key services offered by the device, a human-readable short devices name, etc.

Beacons take this short advertising frame, and appends a custom payload in the "Manufacturer Specific Data" field which includes a unique 128-bit UUID to identify companies or unique entities, as well as two 16-bit values ('Major' and 'Minor', or whatever you'd like to call them) that allow you to differentiate specific stores/premises (Major) and individual Beacon nodes (Minor).

That's basically it. All the rest of the magic is on the app side where your phone listens for these advertising frames, and when it detect something it estimates the distance to the node and displays some sort of alert.

It's terribly simple, but that's probably what makes it so interesting and also so inexpensive to implement!

How Does it Work in Praticce?

Essentially, all you need to do is insert a specific set of bytes into the optional **Manufacturer Specific Data** (<https://adafru.it/cYs>) field of the advertising packet on your Bluetooth Low Energy device.

Inside this field, you need the following values:

- **ID** (uint8_t) - This will always be 0x02
- **Data Length** (uint8_t) - The number of bytes in the rest of the payload = 0x15 (21 in dec)
- **128-bit UUID** (uint8_t[16]) - The 128-bit ID indentifying your company/store/etc
- **Major** (uint16_t) - The major value (to differentiate individual stores, etc.)
- **Minor** (uint16_t) - The minor value (to differentiate nodes withing one location, etc.)
- **TX Power** (uint8_t) - This value is used to try to estimate distance based on the RSSI value

For example, the following is a valid iBeacon payload (separators added for clarity sake):

```
02 | 15 | E2 0A 39 F4 73 F5 4B C4 A1 2F 17 D1 AD 07 A9 61 | 00 00 | 00 00 | C8
```

The only other missing piece is that, following the Bluetooth standard, the Manufacturer Specific Data needs to be preceded by the **Company Identifier** (<https://adafru.it/cYt>). The company identifier for Apple, for example, is **0x004C**, which we'll use for the example above.

Compiling Bluez

In order to use your Raspberry Pi to send out Beacon data in the advertising frame, we'll need to install a few open source tools, mainly **Bluez**

Check if you already have Bluez

If you already have a modern version of Bluez you do not need to do this step!

On your Raspberry Pi, try running

```
sudo apt-get install bluez
```

and then

```
dpkg --status bluez | grep '^Version:'
```

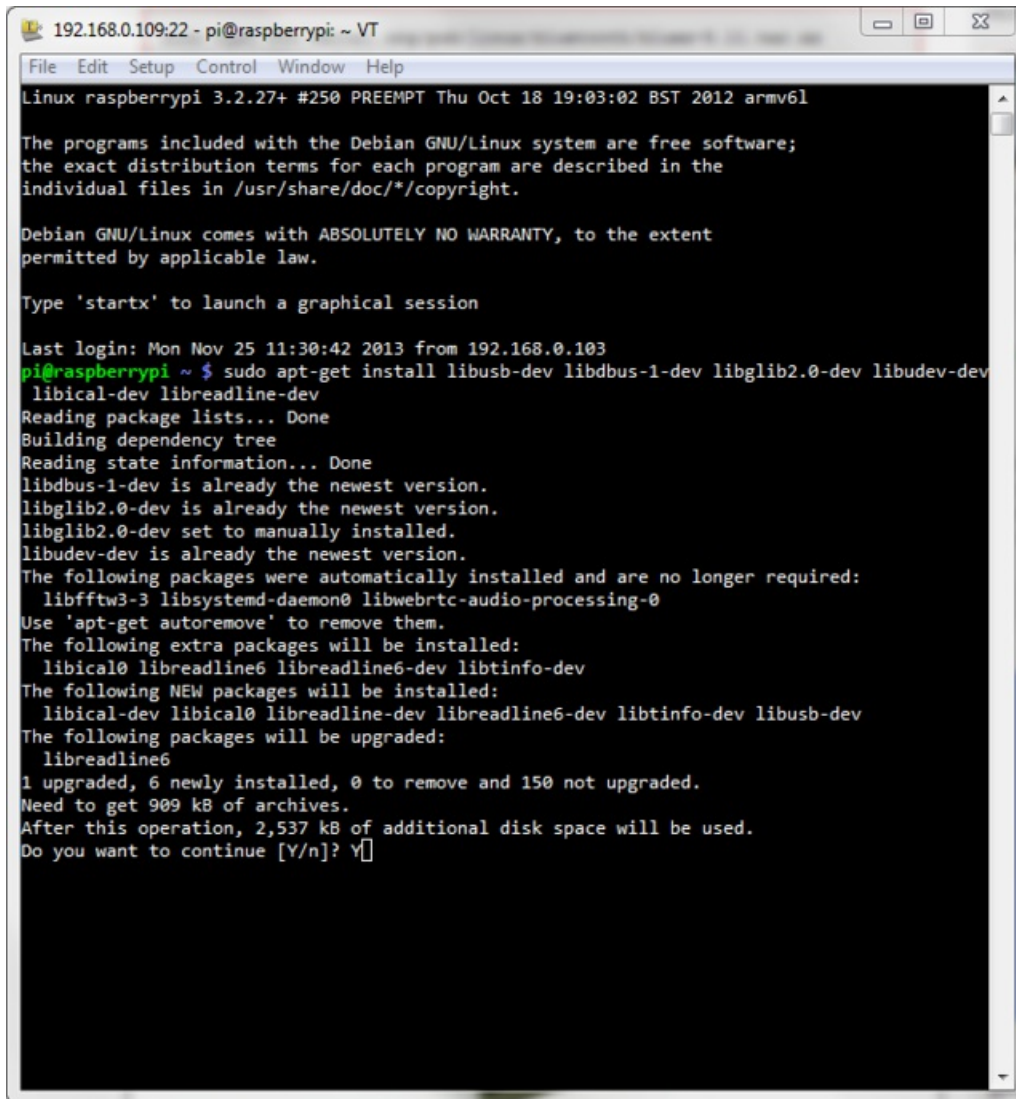
If you get something like **Version: 5.23-2+rpi2**

Where the version is greater than 5.11 you can skip this step.

1. Install Required Libraries

```
sudo apt-get install libusb-dev libdbus-1-dev libglib2.0-dev libudev-dev
sudo apt-get install libical-dev
sudo apt-get install libreadline-dev
```

You may need to type the above code in to make it work rather than doing a copy/paste, or manually remove a superfluous line feed between lines

A terminal window titled "192.168.0.109:22 - pi@raspberrypi: ~ VT" showing the output of a terminal session. The user has run the command "sudo apt-get install libusb-dev libdbus-1-dev libglib2.0-dev libudev-dev libical-dev libreadline-dev". The terminal output shows the package lists, dependency tree, and the packages to be installed. The user has responded with 'Y' to the confirmation prompt.

```
File Edit Setup Control Window Help
Linux raspberrypi 3.2.27+ #250 PREEMPT Thu Oct 18 19:03:02 BST 2012 armv6l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.

Type 'startx' to launch a graphical session

Last login: Mon Nov 25 11:30:42 2013 from 192.168.0.103
pi@raspberrypi ~ $ sudo apt-get install libusb-dev libdbus-1-dev libglib2.0-dev libudev-dev
libical-dev libreadline-dev
Reading package lists... Done
Building dependency tree
Reading state information... Done
libdbus-1-dev is already the newest version.
libglib2.0-dev is already the newest version.
libglib2.0-dev set to manually installed.
libudev-dev is already the newest version.
The following packages were automatically installed and are no longer required:
 libfftw3-3 libsystemd-daemon0 libwebRTC-audio-processing-0
Use 'apt-get autoremove' to remove them.
The following extra packages will be installed:
 libical0 libreadline6 libreadline6-dev libtinfo-dev
The following NEW packages will be installed:
 libical-dev libical0 libreadline-dev libreadline6-dev libtinfo-dev libusb-dev
The following packages will be upgraded:
 libreadline6
1 upgraded, 6 newly installed, 0 to remove and 150 not upgraded.
Need to get 909 kB of archives.
After this operation, 2,537 kB of additional disk space will be used.
Do you want to continue [Y/n]? Y
```

2. Download BlueZ

```
sudo mkdir bluez
cd bluez
sudo wget www.kernel.org/pub/linux/bluetooth/bluez-5.11.tar.xz
```

```
192.168.0.109:22 - pi@raspberrypi: ~/bluez VT
File Edit Setup Control Window Help
b) ...
Unpacking replacement libreadline6:armhf ...
Setting up libreadline6:armhf (6.2+dfsg-0.1) ...
Selecting previously unselected package libical0.
(Reading database ... 61347 files and directories currently installed.)
Unpacking libical0 (from .../libical0_0.48-2_armhf.deb) ...
Selecting previously unselected package libical-dev.
Unpacking libical-dev (from .../libical-dev_0.48-2_armhf.deb) ...
Selecting previously unselected package libtinfo-dev:armhf.
Unpacking libtinfo-dev:armhf (from .../libtinfo-dev_5.9-10_armhf.deb) ...
Selecting previously unselected package libreadline6-dev:armhf.
Unpacking libreadline6-dev:armhf (from .../libreadline6-dev_6.2+dfsg-0.1_armhf.deb) ...
Selecting previously unselected package libreadline-dev:armhf.
Unpacking libreadline-dev:armhf (from .../libreadline-dev_6.2+dfsg-0.1_armhf.deb) ...
Selecting previously unselected package libusb-dev.
Unpacking libusb-dev (from .../libusb-dev_2%3a0.1.12-20+nm1_armhf.deb) ...
Processing triggers for man-db ...
Setting up libical0 (0.48-2) ...
Setting up libical-dev (0.48-2) ...
Setting up libtinfo-dev:armhf (5.9-10) ...
Setting up libreadline6-dev:armhf (6.2+dfsg-0.1) ...
Setting up libreadline-dev:armhf (6.2+dfsg-0.1) ...
Setting up libusb-dev (2:0.1.12-20+nm1) ...
pi@raspberrypi ~ $ ls
Adafruit Desktop dev libnfc python_games
pi@raspberrypi ~ $ sudo mkdir bluez
pi@raspberrypi ~ $ cd bluez
pi@raspberrypi ~/bluez $ sudo wget www.kernel.org/pub/linux/bluetooth/bluez-5.11.tar.xz
--2013-12-02 11:24:12-- http://www.kernel.org/pub/linux/bluetooth/bluez-5.11.tar.xz
Resolving www.kernel.org (www.kernel.org)... 199.204.44.194, 198.145.20.140, 149.20.4.69
Connecting to www.kernel.org (www.kernel.org)|199.204.44.194|:80... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: https://www.kernel.org/pub/linux/bluetooth/bluez-5.11.tar.xz [following]
--2013-12-02 11:24:12-- https://www.kernel.org/pub/linux/bluetooth/bluez-5.11.tar.xz
Connecting to www.kernel.org (www.kernel.org)|199.204.44.194|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1092980 (1.0M) [application/x-xz]
Saving to: `bluez-5.11.tar.xz'

100%[=====>] 1,092,980 596K/s in 1.8s

2013-12-02 11:24:22 (596 KB/s) - `bluez-5.11.tar.xz' saved [1092980/1092980]

pi@raspberrypi ~/bluez $
```

3. Unzip and Compile Bluez

Next you need to actually build Bluez on the Pi. This step may take a while, but should work without any hiccups if you properly installed all the libraries in step one above:

```
sudo unxz bluez-5.11.tar.xz
sudo tar xvf bluez-5.11.tar
cd bluez-5.11
sudo ./configure --disable-systemd
sudo make
sudo make install
```



```
192.168.0.109:22 - pi@raspberrypi: ~/bluez/bluez-5.11 VT
File Edit Setup Control Window Help
CCLD tools/obex-server-tool
CC tools/bluetooth-player.o
CCLD tools/bluetooth-player
CC tools/obexctl.o
CCLD tools/obexctl
CC unit/test-eir.o
CC src/eir.o
CC src/glib-helper.o
CCLD unit/test-eir
CC unit/test-uuid.o
CCLD unit/test-uuid
CC unit/test-textfile.o
CC src/textfile.o
CCLD unit/test-textfile
CC unit/test-crc.o
CCLD unit/test-crc
CC unit/test-mgmt.o
CC src/shared/util.o
CC src/shared/mgmt.o
CCLD unit/test-mgmt
CC unit/test-sdp.o
CC src/sdpd-database.o
CC src/sdpd-service.o
CC src/sdpd-request.o
CCLD unit/test-sdp
CC unit/test-gdbus-client.o
CCLD unit/test-gdbus-client
CC unit/util.o
CC unit/test-gobex-header.o
CCLD unit/test-gobex-header
CC unit/test-gobex-packet.o
CCLD unit/test-gobex-packet
CC unit/test-gobex.o
CCLD unit/test-gobex
CC unit/test-gobex-transfer.o
CCLD unit/test-gobex-transfer
CC unit/test-gobex-apparam.o
CCLD unit/test-gobex-apparam
CC unit/test-lib.o
CCLD unit/test-lib
CC tools/hid2hci.o
CCLD tools/hid2hci
GEN tools/97-hid2hci.rules
pi@raspberrypi ~/bluez/bluez-5.11 $
```

4. Insert the USB Module and Reset

Once Bluez has been built, shut down your computer with `sudo shutdown -h now` and once its Halted, insert your [Bluetooth 4.0 USB Module \(http://adafru.it/1327\)](http://adafru.it/1327) and then restart the Raspberry Pi so that all of the changes we have made can take effect.

Adding Beacon Data

In the 'bluez-5.11' folder that we previously created, we can start entering the mandatory Beacon data and advertising it using `hcitool`, which we built when compiling Bluez.

1. Check for your USB Module

This should give you a list of devices on your system:

```
tools/hciconfig (if you compiled bluez)
```

or

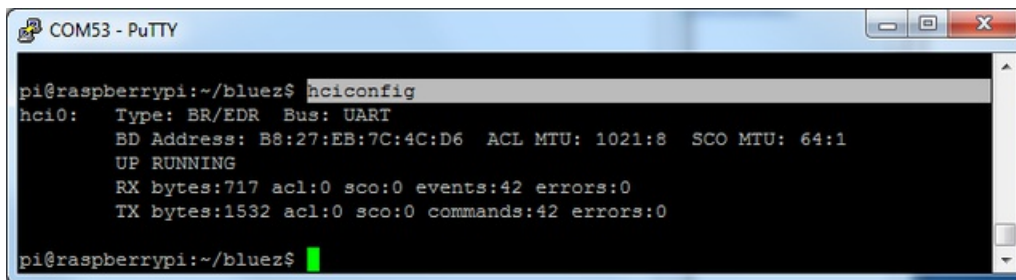
```
hciconfig (if you apt-get'd bluez)
```

If everything is properly configure you will see your Bluetooth 4.0 USB Module like this:

```
pi@raspberrypi ~/bluez/bluez-5.11 $ tools/hciconfig
hci0:  Type: BR/EDR  Bus: USB
      BD Address: 00:1A:7D:DA:71:12  ACL MTU: 310:10  SCO MTU: 64:8
      DOWN
      RX bytes:495 acl:0 sco:0 events:21 errors:0
      TX bytes:89 acl:0 sco:0 commands:21 errors:0

pi@raspberrypi ~/bluez/bluez-5.11 $
```

On a Raspberry Pi 3 you'll see the Bus is UART, not USB!



```
COM53 - PuTTY
pi@raspberrypi:~/bluez$ hciconfig
hci0:  Type: BR/EDR  Bus: UART
      BD Address: B8:27:EB:7C:4C:D6  ACL MTU: 1021:8  SCO MTU: 64:1
      UP RUNNING
      RX bytes:717 acl:0 sco:0 events:42 errors:0
      TX bytes:1532 acl:0 sco:0 commands:42 errors:0

pi@raspberrypi:~/bluez$
```

2. Enable the USB Device

Next you can enable the device with the following commands, turning off device scanning since this can cause problems when advertising.

If you're using the compiled bluez, add `tools/` before each call to `hciconfig`

```
sudo hciconfig hci0 up
sudo hciconfig hci0 leadv 3
sudo hciconfig hci0 noscan
```

Then run the `hciconfig` tool again and you should see that the device is marked as **UP** and **RUNNING**:

```
hciconfig
```

```

pi@raspberrypi ~/bluez/bluez-5.11 $ sudo tools/hciconfig hci0 leadv
pi@raspberrypi ~/bluez/bluez-5.11 $ sudo tools/hciconfig hci0 noscan
pi@raspberrypi ~/bluez/bluez-5.11 $ tools/hciconfig
hci0:  Type: BR/EDR  Bus: USB
      BD Address: 00:1A:7D:DA:71:12  ACL MTU: 310:10  SCO MTU: 64:8
      UP RUNNING
      RX bytes:1008 acl:0 sco:0 events:45 errors:0
      TX bytes:236 acl:0 sco:0 commands:45 errors:0

```

3. Enter the Beacon Advertising Data

The last thing to do is to enter the Beacon advertising data, which we can do with the following command (which should all be on one line):

```

sudo hcitool -i hci0 cmd 0x08 0x0008 1E 02 01 1A 1A FF 4C 00 02 15 E2 0A 39 F4 73 F5 4B C4 A1 2F 17 D1 AD

```

FF identifies the start of the Manufacturer Specific Data, 4C 00 is Apple's company ID (0x004C), and then you can see the rest of the Beacon payload until C8.

Results on a Bluetooth Debugger

Just to show that this actually works, you can see the results using a Bluetooth Low Energy sniffer below:

```

0x001A7DDA7112 (-74dBm)
  RSSI: -74dBm
  Address: 001A7DDA7112
  Address Type: Public
  Advertising Type: Connectable
  Bonded: False
  Advertising Data
    Flags: GeneralDiscoverable, LeAndBrEdrCapableController, LeAndBrEdrCapableHost
    ManufacturerSpecificData: 4C-00-02-15-E2-0A-39-F4-73-F5-4B-C4-A1-2F-17-D1-AD-07-A9-61-00-00-00-00-C8
  Scan Response Data

```

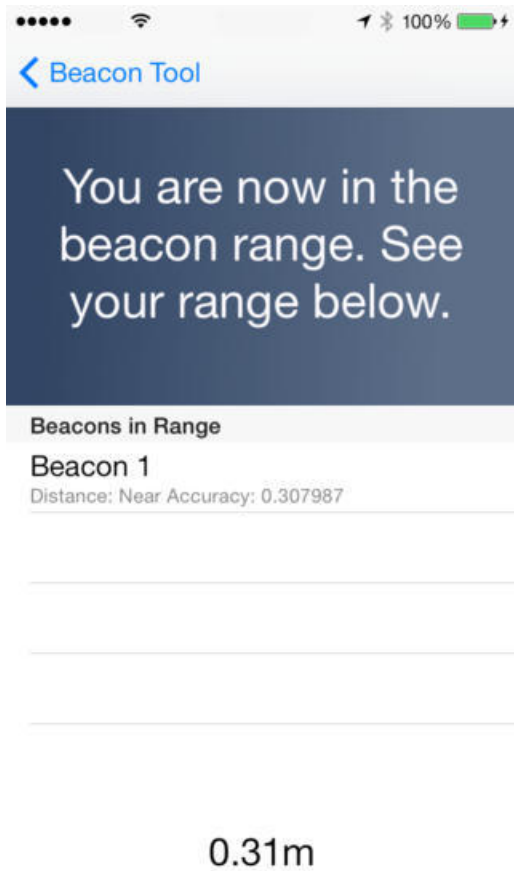
And here is the raw advertising frame from a different sniffer:

| Pkts | Time (us) | Channel | Access Address | Adv PDU Type | Adv PDU Header | | | AdvA | AdvData | CRC | RSSI (dBm) | FCS |
|------|-----------|---------|----------------|--------------|----------------|-------|-------|------------|---|----------|------------|-----|
| 1 | +0 | 0x25 | 0x8E99BE04 | ADV_IND | Type | TxAdd | RxAdd | PDU-Length | 02 01 1A 1A FF 4C 00 02 15 E2 0A 39 F4 73 F5 4B C4 A1 2F 17 D1 AD 07 A9 61 00 00 00 00 C8 | 0x00B8B5 | -64 | OK |
| | | | | | 0 | 0 | 0 | 36 | | | | |

Testing it on iOS

To test that this actually works you'll need an iOS 7 based iPad/iPhone/iPod Touch, and a Beacon application.

Start the app up, going into 'Listen' mode, and you should see a screen similar to the capture below, where the range will go in and out depending on your proximity to the node:



Beacon Toolkit only searches for specific Beacon UUIDs -- the same UUID used in this tutorial. If you use a different UUID, you will have to find a different tool to test on your iOS device.