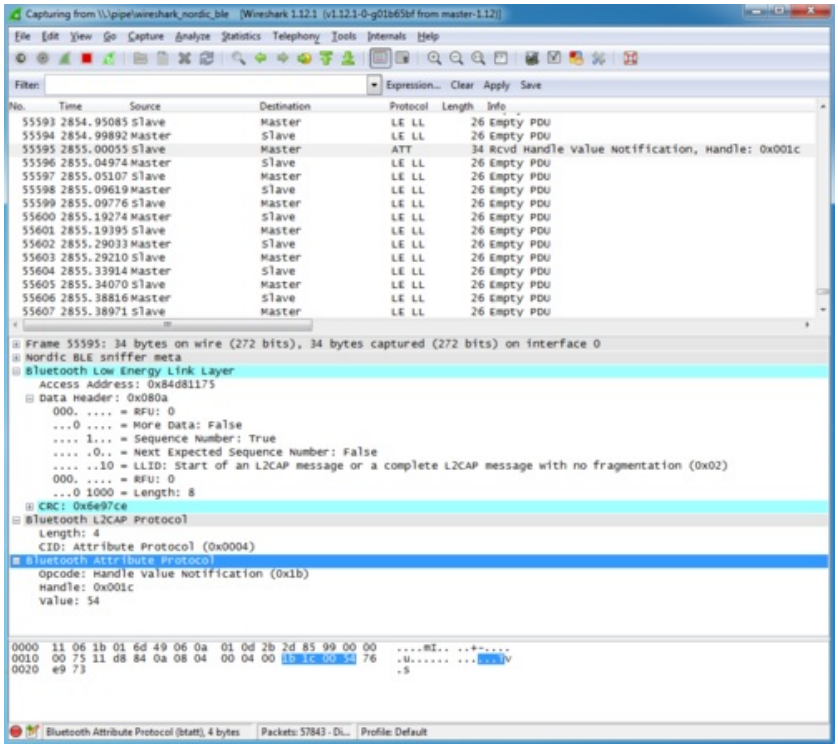
adafruit learning system

Introducing the Adafruit Bluefruit LE Sniffer

Created by Kevin Townsend



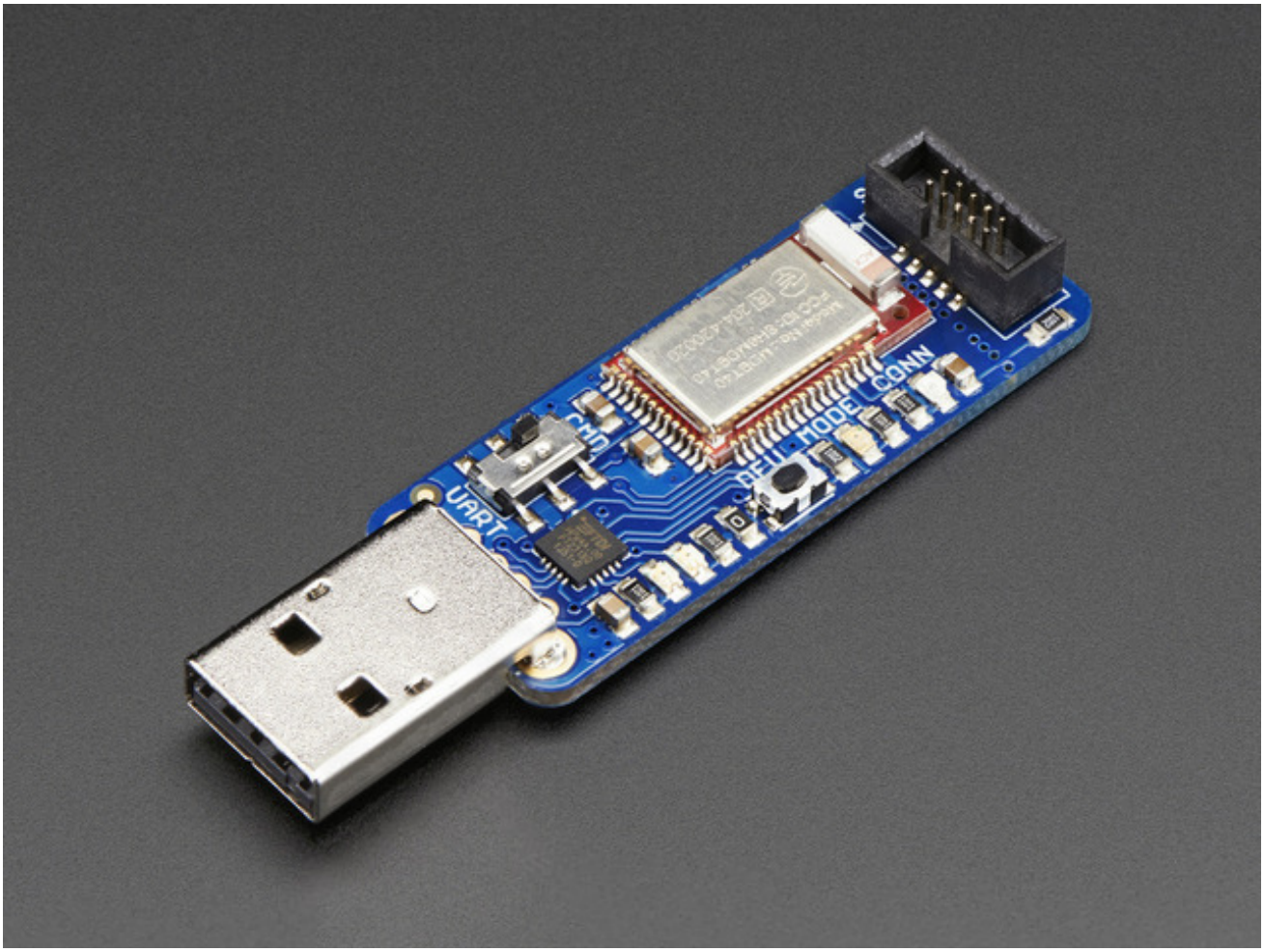
Last updated on 2015-12-08 10:30:13 PM EST

Guide Contents

Introduction

Using a special firmware image provided by Nordic Semiconductors and the open source network analysis tool Wireshark, the Bluefruit LE Sniffer (http://adafru.it/edE) can be used as a low cost Bluetooth Low Energy sniffer.

NOTE: This product can only be used to sniff Bluetooth Low Energy devices. It will not work with classic Bluetooth devices or transactions.



Since nRF-Sniffer is a passive solution that is simply scanning packets over the air, there is the possibility of missing packets using this tool (or any other passive sniffing solution). In order to capture as many packets as possible, be sure to run the sniffer on a USB bus that isn't busy and avoid running it in a virtual machine since this can introduce significant latency over USB.

FTDI Driver Requirements

Before you can start talking to the sniffer, you'll need to install a standard FTDI driver for the FT231x located on the device.

Find the appropriate FTDI VCP installer on the FTDI Driver Download Page (http://adafru.it/aJv), install it on you system, and then insert the sniffer in any USB port on your system.

Currently Supported VCP Drivers:

		Processor Architecture							
Operating System	**Release Date**	**x86 (32-bit)**	**x64 (64-bit)**	**PPC**	**ARM**	**MIPSII**	**MIPSIV**	**SH4**	**Comments**
Windows*	2014-09-29	Available as setup executable Contact support1@ftdichip.com if looking to create customised drivers		-	-	-	-	-	2.12.00 WHQL Certified Available as setup executable Release Notes
Linux	2009-05-14	1.5.0	1.5.0	-	-	-	-	-	All FTDI devices now supported in Ubuntu 11.10, kernel 3.0.0-19 Refer to TN-101 if you need a custom VCP VID/PID in Linux
Mac OS X	2012-08-10	2.2.18	2.2.18	2.2.18	-	-	-	-	Refer to TN-105 if you need a custom VCP VID/PID in MAC OS

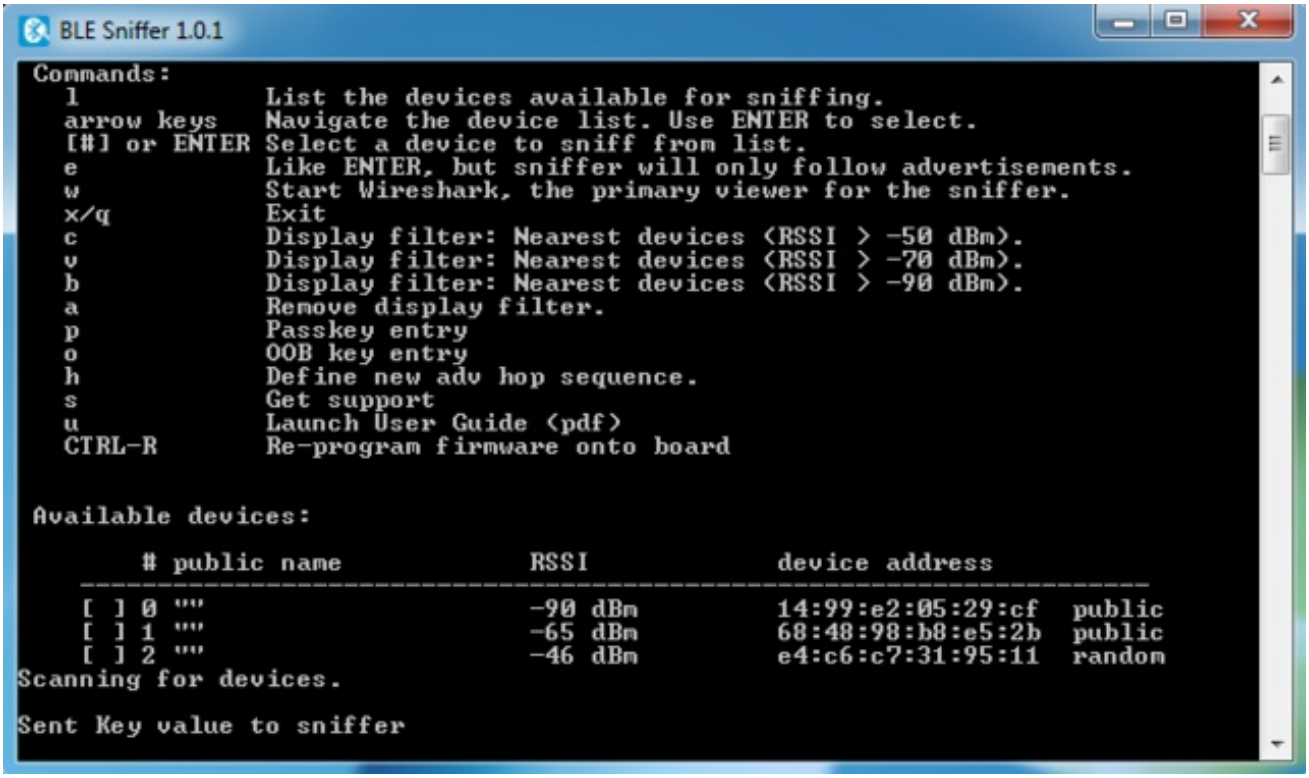
Using the Sniffer

There are currently two ways to use the sniffer:

Nordic's nRF Sniffer Utility (Windows only)

If you are on Windows, the best user experience will be had by using the official Nordic nRFSniffer application, available as a download from Nordic Semiconductors after creating a 'My Pages' account, and regitering your device using the product ID located on the Bluefruit LE Sniffer packaging.

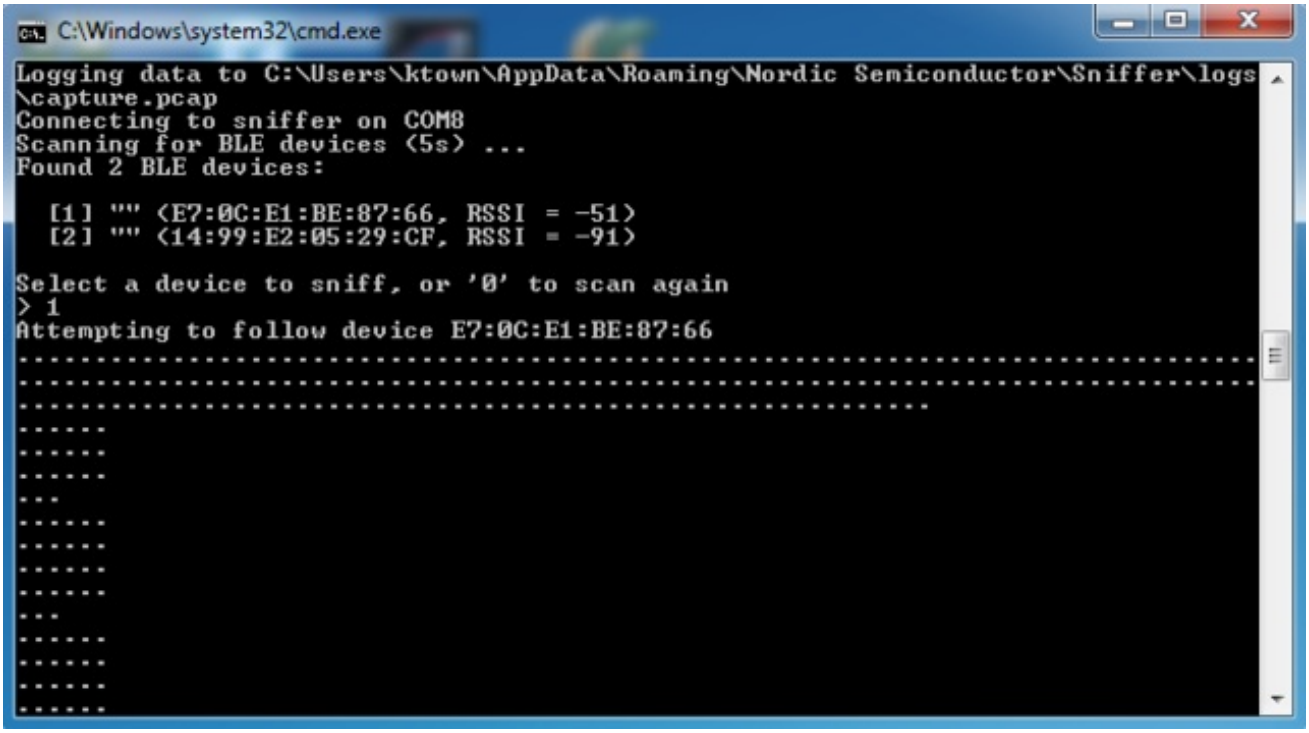
More information on using Nordic's nRF Sniffer application (http://adafru.it/k6F).



Python API (Cross-Platform, no Registration)

If you are not using Windows, or don't wish to create a MyPages account, the alternative is to use a Python interface to communicate with the nRFSniffer firmware, which will log any traffic to a libpcap file that can be opened directly in Wireshark. This has been tested on OS X 10.10, Ubuntu 14.04 and Windows 7, but it currently doesn't support streaming data directly into Wireshark via named pipes (though this is possible with some platform-specific effort).

More information on using the Python API (http://adafru.it/k7a).


C:\Windows\system32\cmd.exe
Logging data to C:\Users\ktown\AppData\Roaming\Nordic Semiconductor\Sniffer\logs
\capture.pcap
Connecting to sniffer on COM8
Scanning for BLE devices (5s) ...
Found 2 BLE devices:
[1] "" (E7:0C:E1:BE:87:66, RSSI = -51)
[2] "" (14:99:E2:05:29:CF, RSSI = -91)
Select a device to sniff, or '0' to scan again
> 1
Attempting to follow device E7:0C:E1:BE:87:66

Nordic nRF Sniffer

The following guide will walk you through downloading, installing and using the official nRF Sniffer application fro Nordic Semiconductors.

Getting the Sniffer Utility

The Bluefruit LE Sniffer comes pre-flashed with the special sniffer firmware image, but you'll need to go to Nordic's website and download the **nRF-Sniffer** package to capture the data on Windows and push it out into Wireshark for packet by packet analysis.

Go to the nRF Sniffer product page (http://adafru.it/ezV) and click the 'downloads' tab, then download the latest version of the utility, as shown below, and unzip it:

SOFTWARE

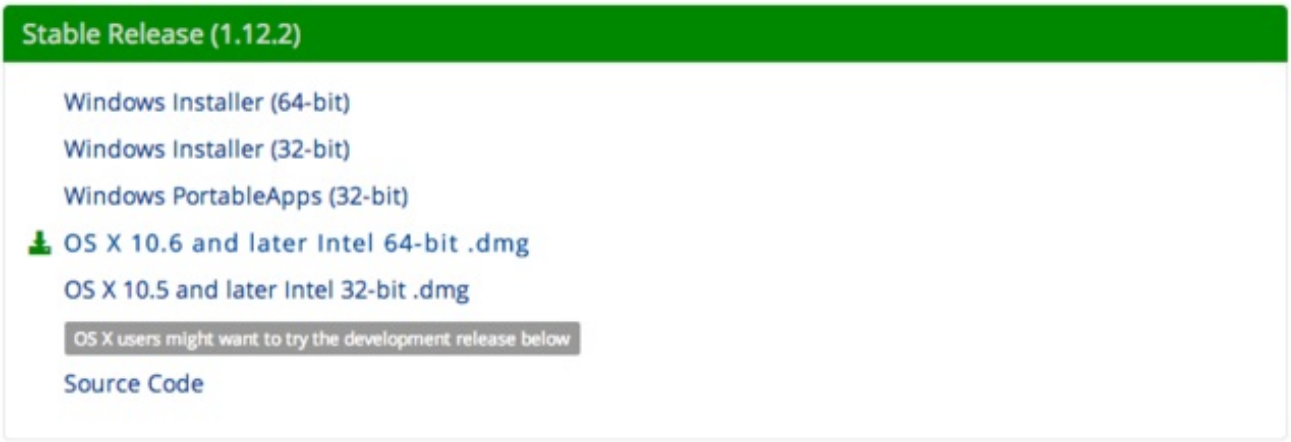
Code	Name	Version
nRF-Sniffer	nRF Sniffer (First Production release) - PC Software and Device Firmware that allow you to see all Bluetooth low energy packets on the air between two devices.	1.0.1

Inside this downloaded file you'll find the sniffer executable, which will open up the command-line tool when you click on it.

Getting Wireshark

In order to use the sniffer utility you'll also need to download Wireshark (http://adafru.it/ecp), preferably verison 1.12.1 (the same one used in this tutorial).

Simply select the 32-bit or 64-bit Windows Installer and install it on your machine using the default settings:



Make sure that you install the libpcap library when installing Wireshark. Any log files captured by the python library are in libpcap format, and will require this library to work.

Running the Sniffer

Now that everything is installed, you can get started using the Bluefruit LE Sniffer and the sniffer bridge SW that pushes any sniffed data out into Wireshark ...

Select the Sniffer Target

The nRF-Sniffer can only sniff one device at a time, so the first step is getting the sniffer running and then selecting the device that you want to debug.

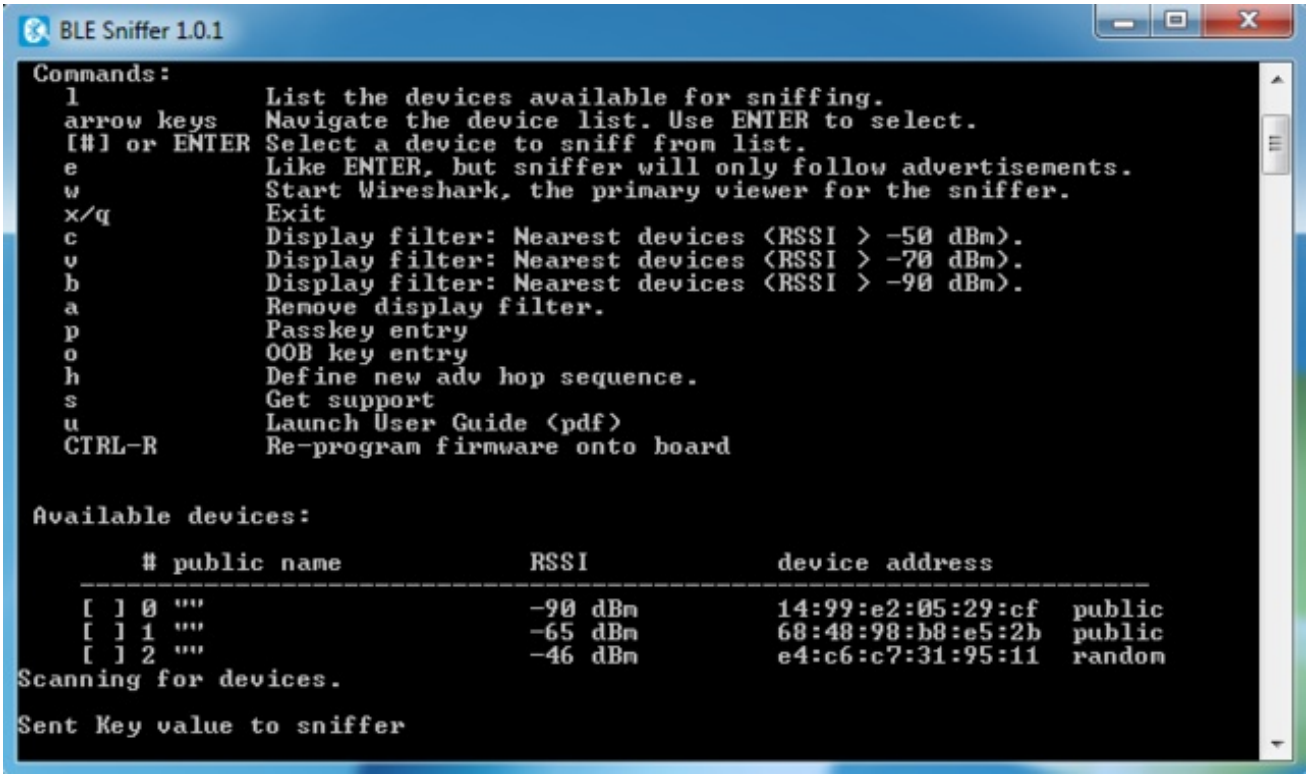
Start nRF-Sniffer by running the ble-sniffer_win executable (for example: ble-sniffer_win_1.0.1_1111_Sniffer.exe).

This will try to detect the device running the nRF-Sniffer firmware over a UART COM port.

If the board isn't detected right away type **'f'** to erase any previous com port settings, or try removing and then re-inserting the sniffer while the console application is running.

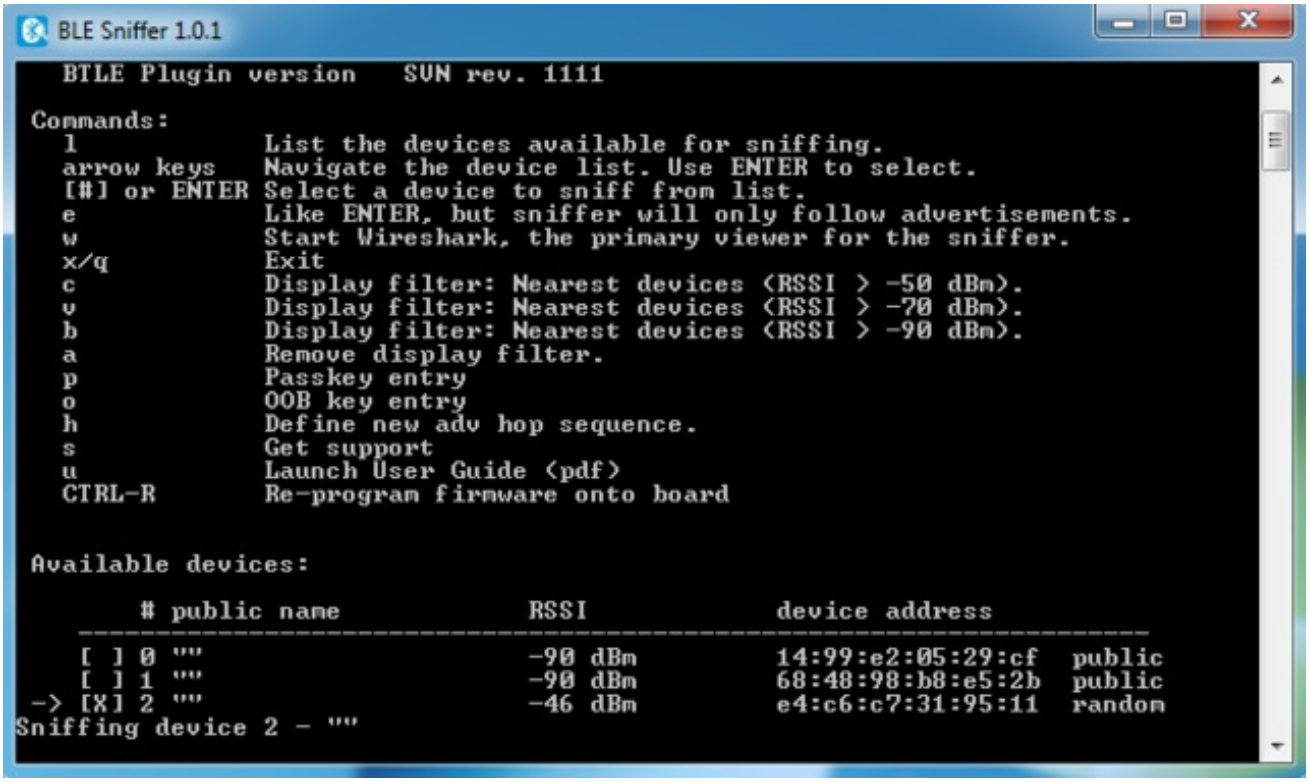
Once the sniffer is found, you should see a list of all BLE devices that were detected in listening range:

If you see a warning in the application about your firmware being out of date and requesting to update it, IGNORE THE WARNING. The Adafruit boards run a slightly modified version of the sniffer firmware, which causes the tool to think it is out of date.



In this particular case, we'll select device number 2, which is a BLEFriend running the standard UART firmware.

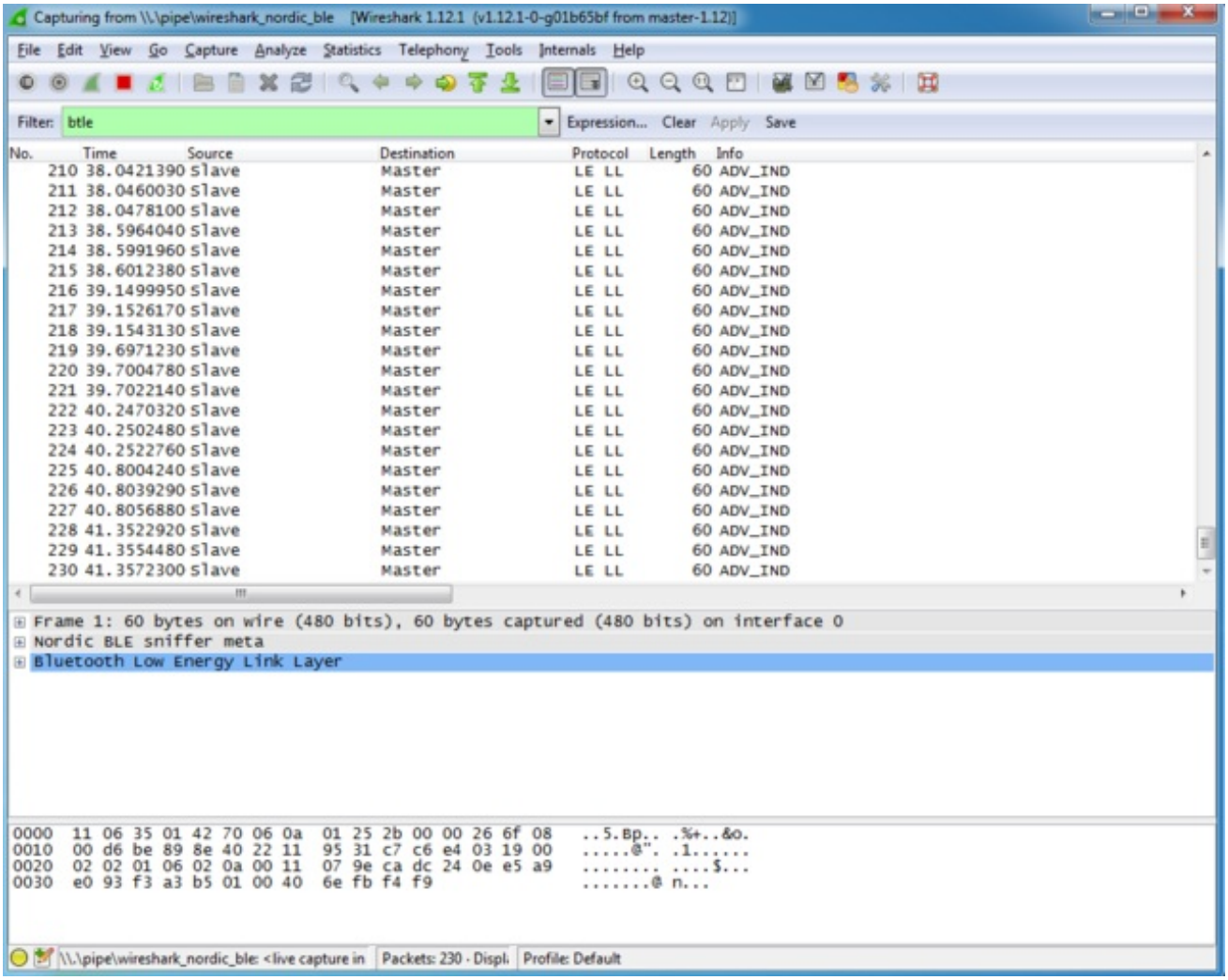
Type the device number you want to sniffer (in this case **'2'**), and you should see the device highlighted in the list, similar to the image below:



At this point you can type **'w'**, which will try to open wireshark and start pushing data out via a dedicate pipe created by the nRF-Sniffer utility.

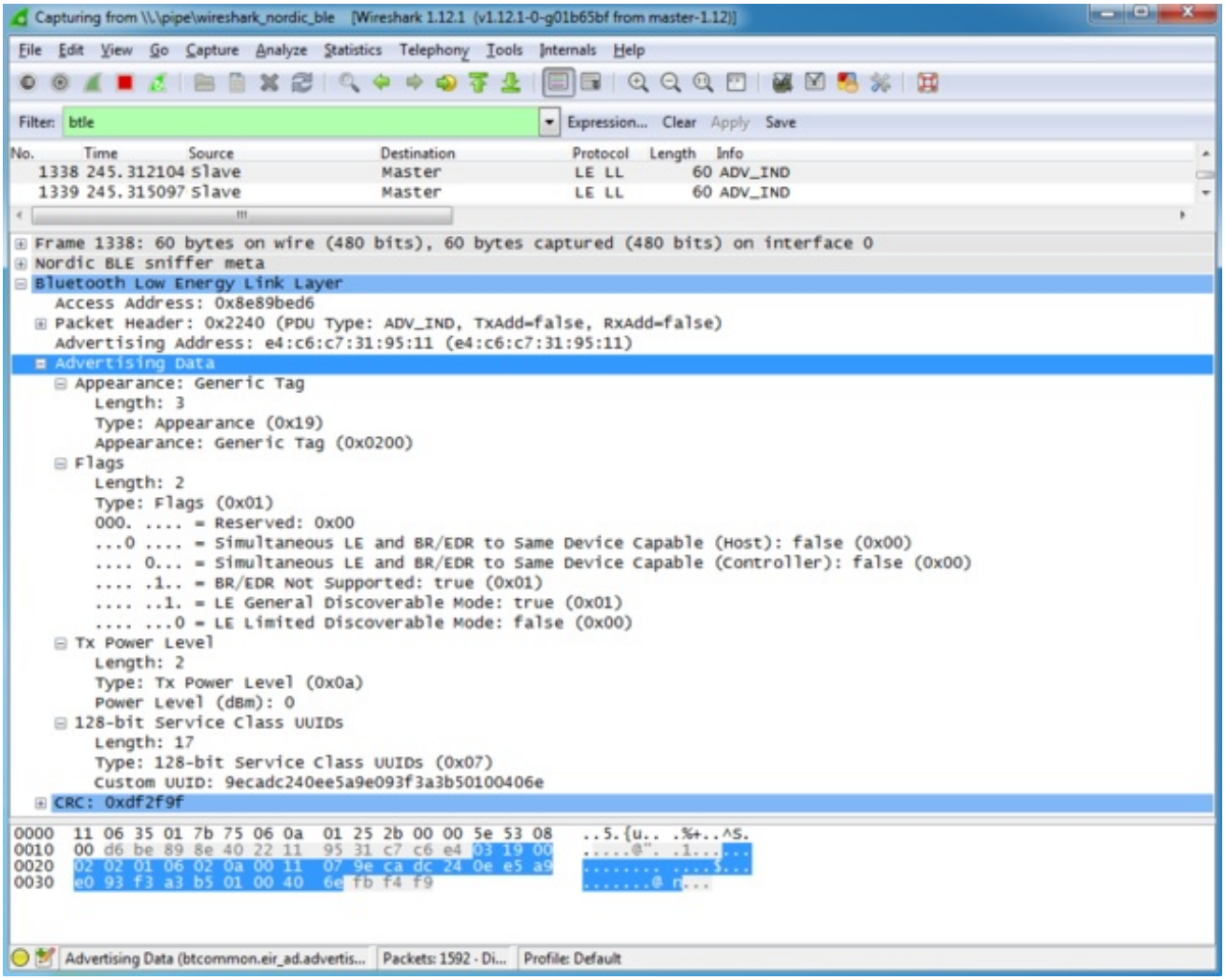
Working with Wireshark

Once Wireshark has loaded, you should see the advertising packets streaming out from the selected BLE device at a regular intercal, as shown in the image below:



One of the key benefits of WireShark as an analysis tool is that it understands the raw packet formats and provides human-readable displays of the raw packet data.

The main way to interact with BLE data packets is to select one of the packets in the main window, and then expand the **Bluetooth Low Energy Link Layer** treeview item in the middle of the UI, as shown below:



Clicking on the **Advertising Data** entry in the treeview will highlight the relevant section of the raw payload at the bottom of the screen, but also provides human readable information about the payload that can save you a lot of time trying to debug or reverse engineer a device.

We can see, for example, that the device is advertising itself as a Bluetooth Low Energy only device ('BR/EDR Not Supported'), with a TX Power Level of 0dBm, and a single service is being advertised using a 128-bit UUID (the UART service in this case).

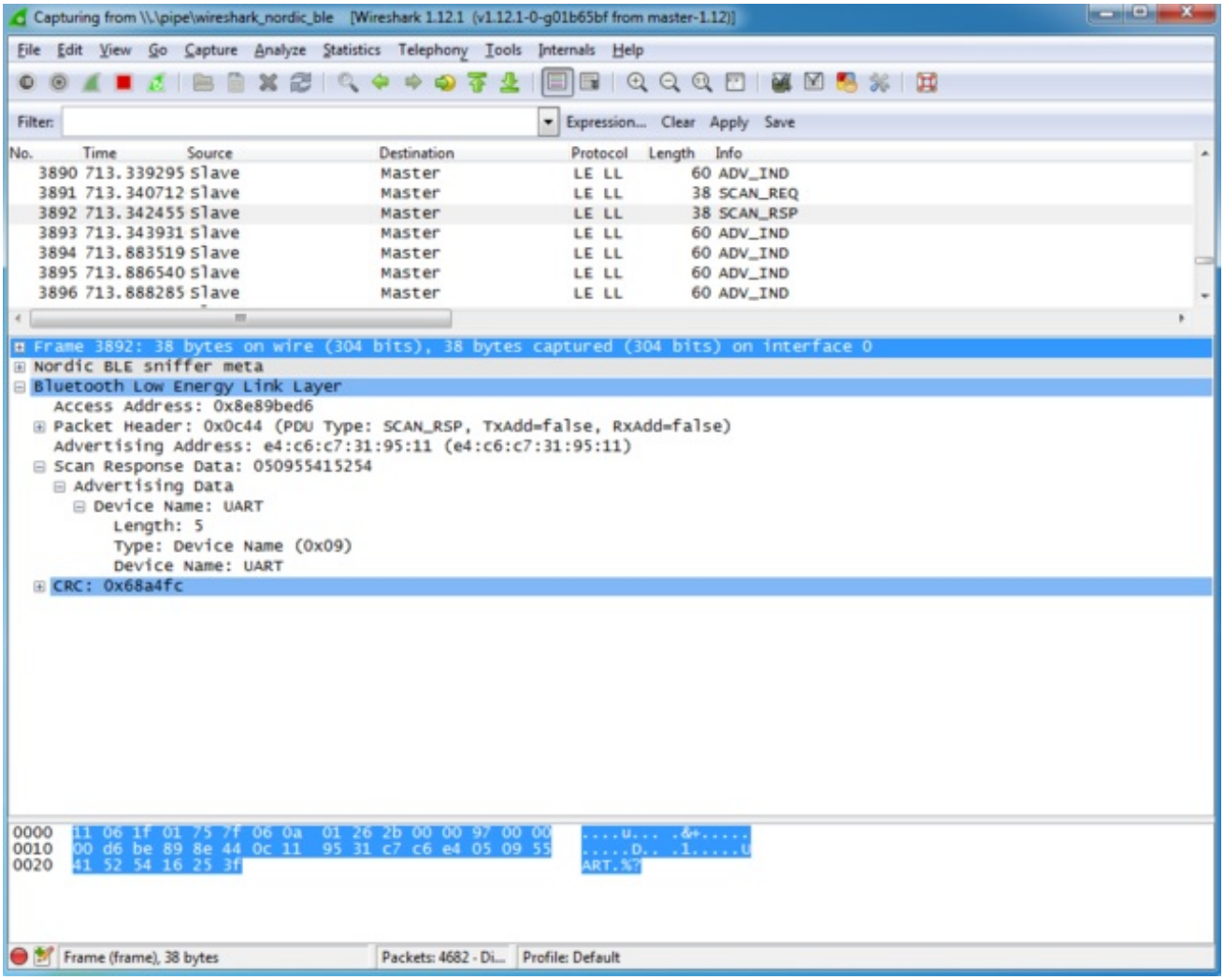
Capturing Exchanges Between Two Devices

If you wish to sniff data being exchanged between two BLE devices, you will need to establish a connection between the original device we selected above and a second BLE device (such as an iPhone or an Android tablet with BLE capabilities).

The nRF-Sniffer firmware is capable is listening the all of the exchanges that happen between these devices, but can not connect with a BLE peripheral or central device itself (it's a purely passive device).

Scan Response Packets

If you open up **nRF UART** on an Android or iOS device, and click the **Connect** button, the phone or tablet will start scanning for devices in range. One of the side effects of this scanning process is that you may spot a new packet in Wireshark on an irregular basis, the 'SCAN_REQ' and 'SCAN_RSP' packets:



The **Scan Response** is an optional second advertising packet that some Bluetooth Low Energy peripherhals use to provide additional information during the advertising phase. The normal mandatory advertising packet is limited to 31 bytes, so the Bluetooth SIG includes the possibility to request a second advertising payload via the **Scan Request**.

You can see both of these transactions in the image above, and the **Device Name** that is included in the Scan Response payload (since the 128-bit UART Service UUID takes up most of the free space in the main advertising packet).

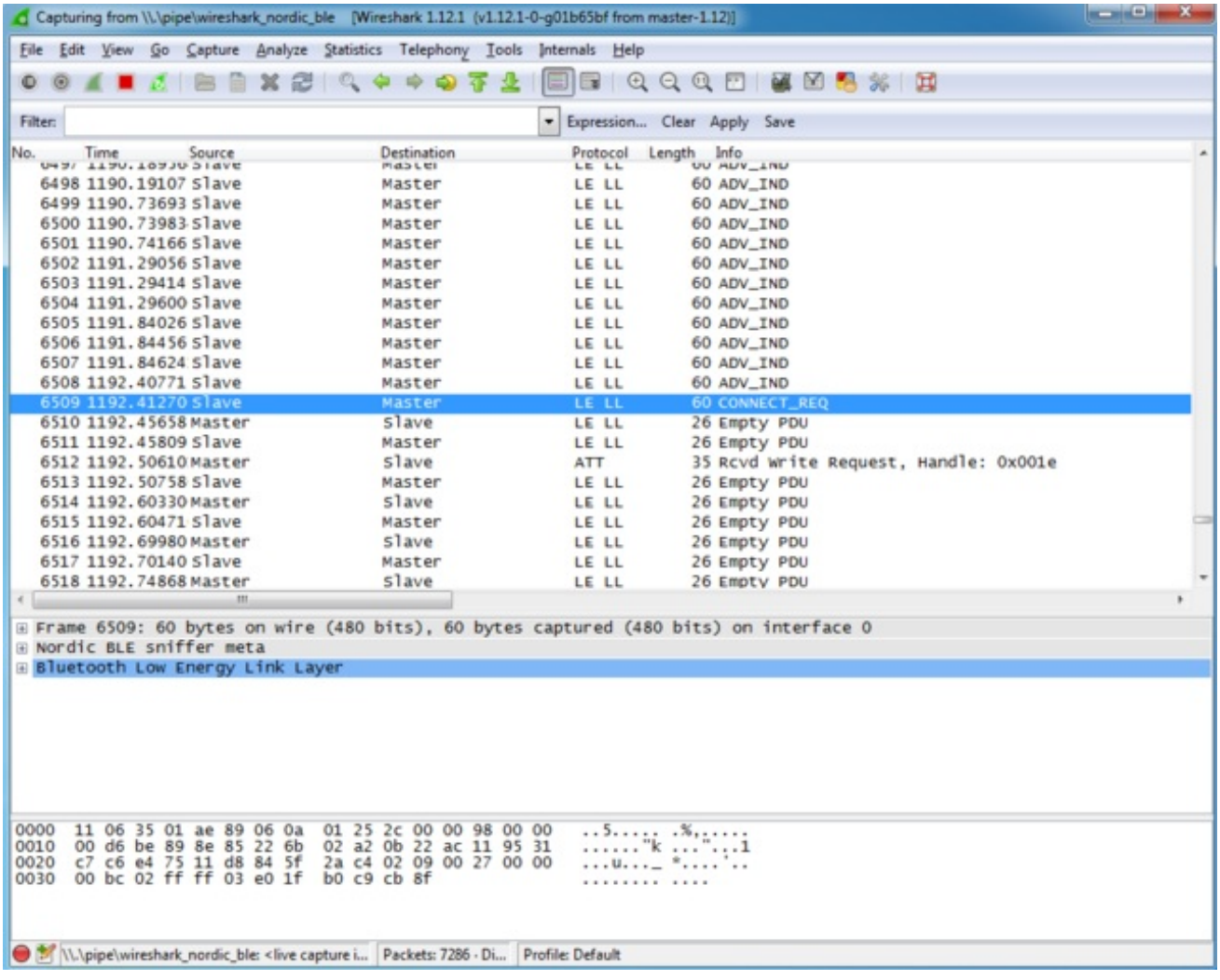
For more information on Scan Responses and the advertising process in Bluetooth Low Energy see

our Introduction to Bluetooth Low Energy Guide (http://adafru.it/iCo).

Connection Request

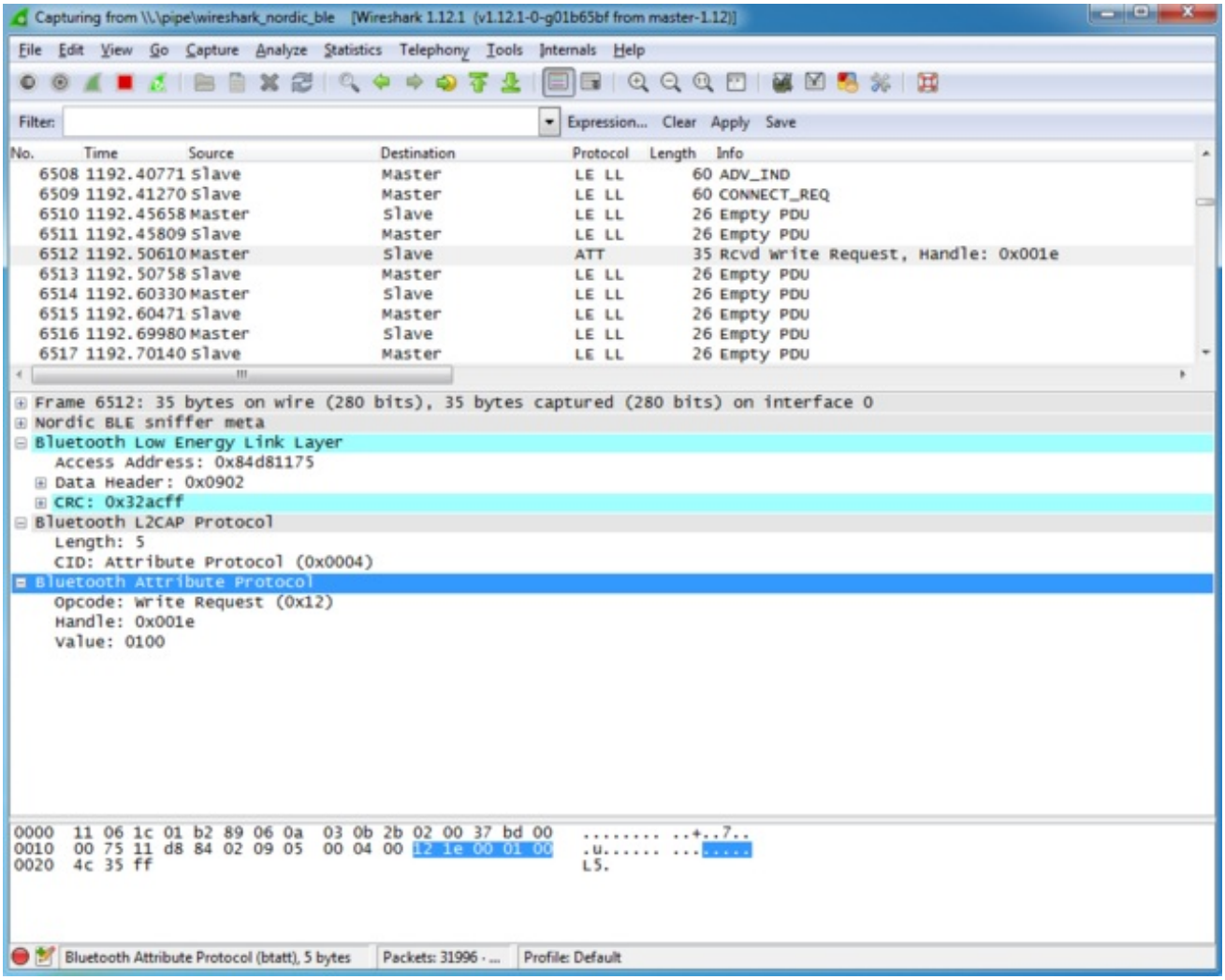
Once we click on the **UART** device in **nRF UART**, the two device will attempt to connect to each other by means of a **Connection Request**, which is initiated by the central device (the phone or tablet).

We can see this CONNECT_REQ in the timeline in the image below:



Write Request

Once the connection has been established, we can see that the **nRF UART** application tries to write data to the BLEFriend via a **Write Request** to handle '0x001E' (which is the location of an entry in the attribute table since everything in BLE is made up of attributes).



What this write request is trying to do is enable the 'notify' bit on the UART service's TX characteristic (http://adafru.it/k7b) (0x001E is the handle for the CCCD or 'Client Characteristic Configuration Descriptor (http://adafru.it/ecl)'). This bit enables an 'interrupt' of sorts to tell the BLEFriend that we want to be alerted every time there is new data available on the characteristic that transmits data from the BLEFriend to the phone or tablet.

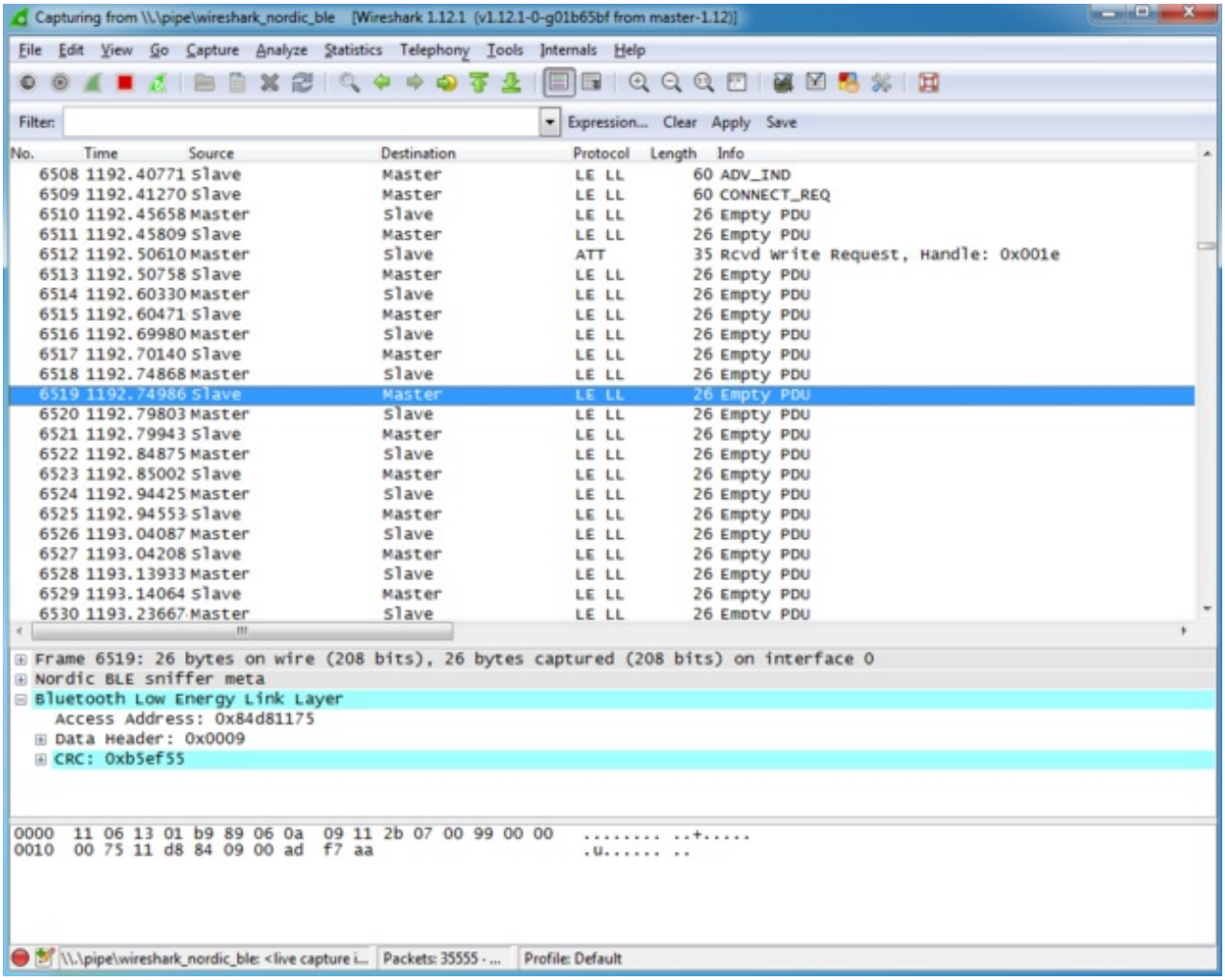
Regular Data Requests

At this point you will start to see a lot of regular **Empty PDU** requests. This is part of the way that Bluetooth Low Energy works.

Similar to USB, all BLE transaction are initiated by the bus 'Master', which is the central device (the tablet or phone).

In order to receive data from the bus slave (the peripheral device, or the BLEFriend in this particular case) the central device sends a 'ping' of sorts to the peripheral at a delay known as the 'connection interval' (not to be confused with the one-time connection highlighted earlier in this tutorial).

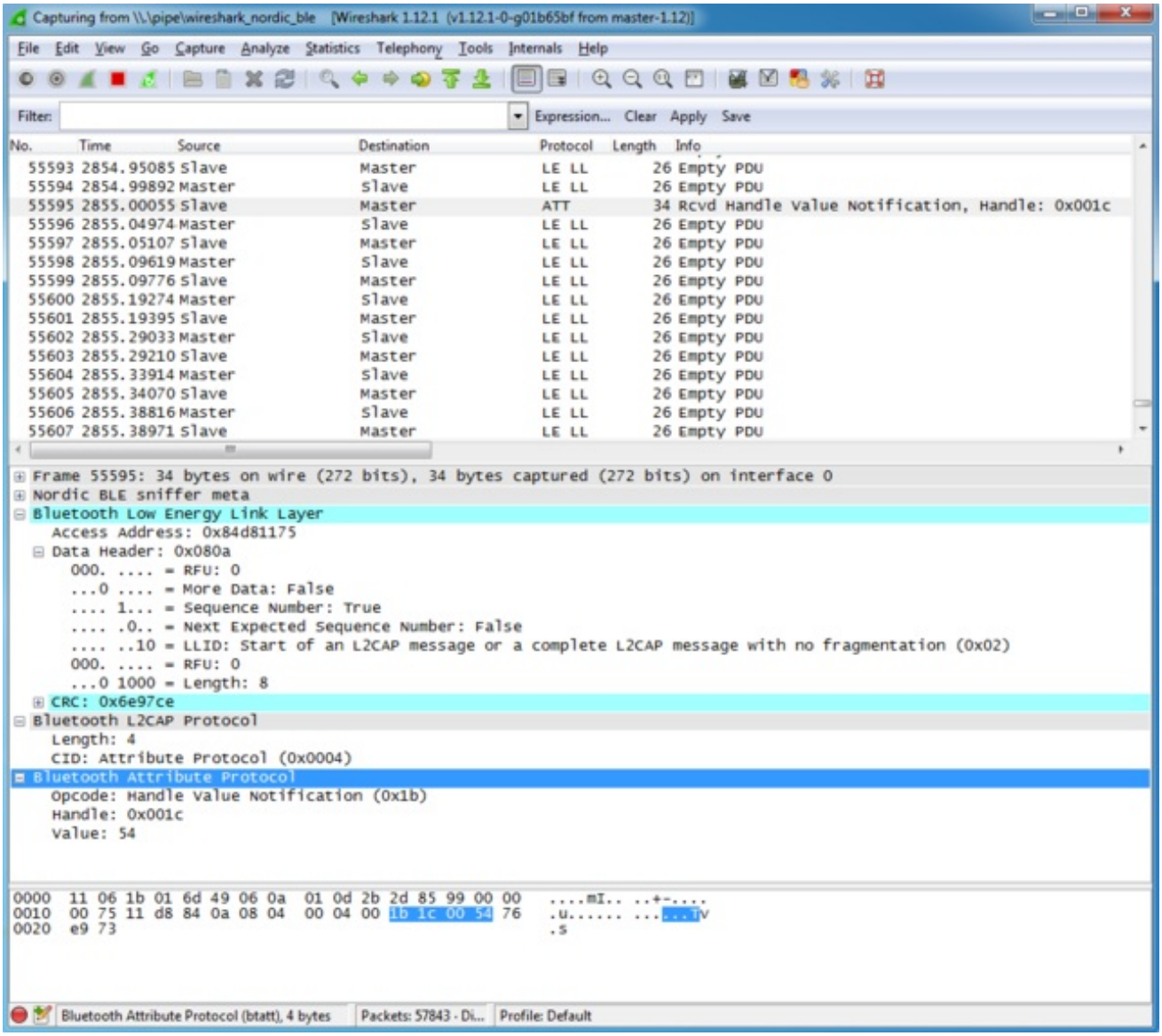
We can see pairs of transaction that happen at a reasonably consistent interval, but no data is exchanged since the BLEFriend (the peripheral) is saying 'sorry, I don't have any data for you':



Notify Event Data

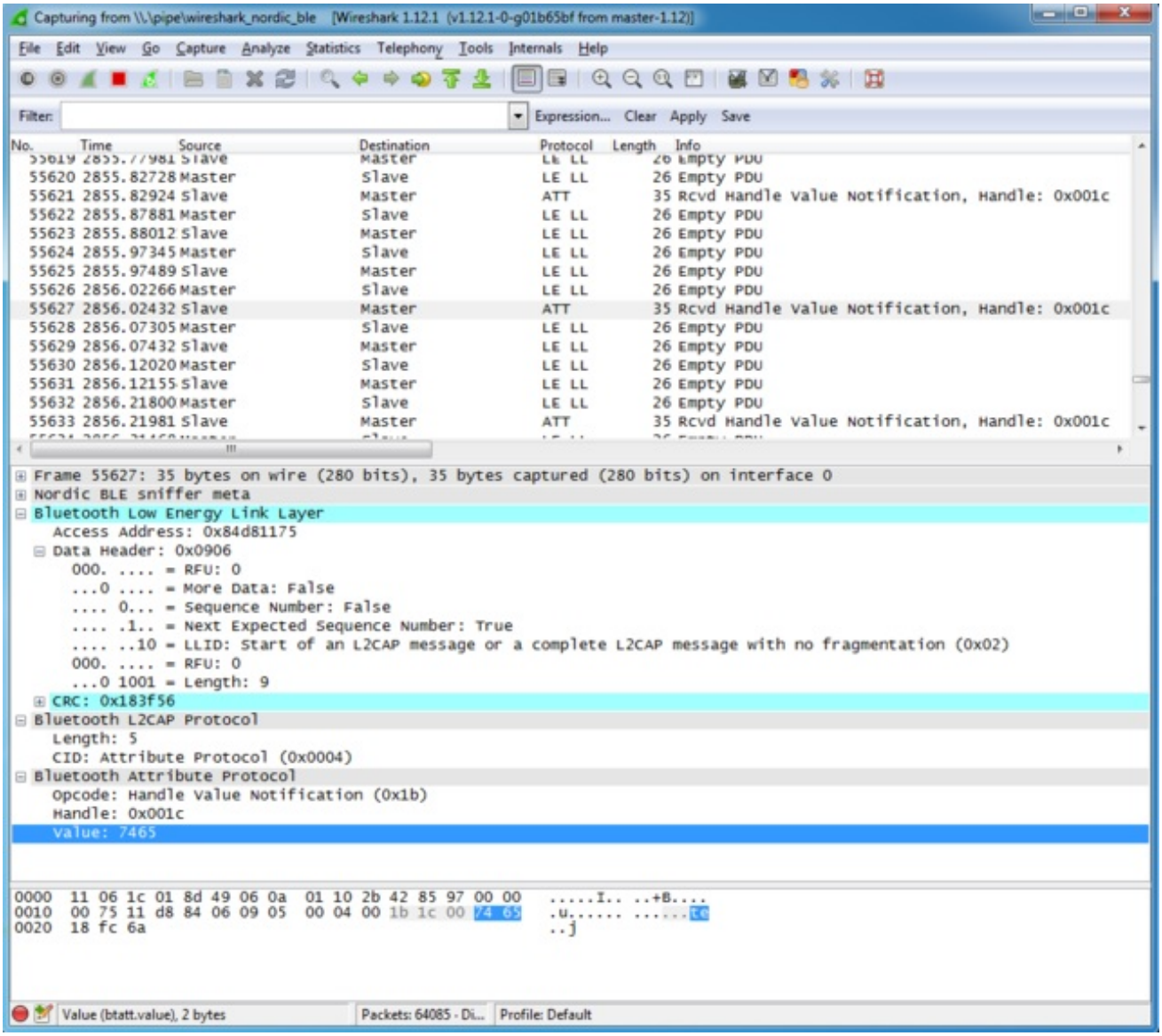
To see an actual data transaction, we simply need to enter some text in our terminal emulator SW which will cause the BLEFriend to send the data to **nRF UART** using the UART service.

Entering the string 'This is a test' in the terminal emulator, we can see the first packet being sent below (only the 'T' character is transmitted because the packets are sent out faster than we enter the characters into the terminal emulator):

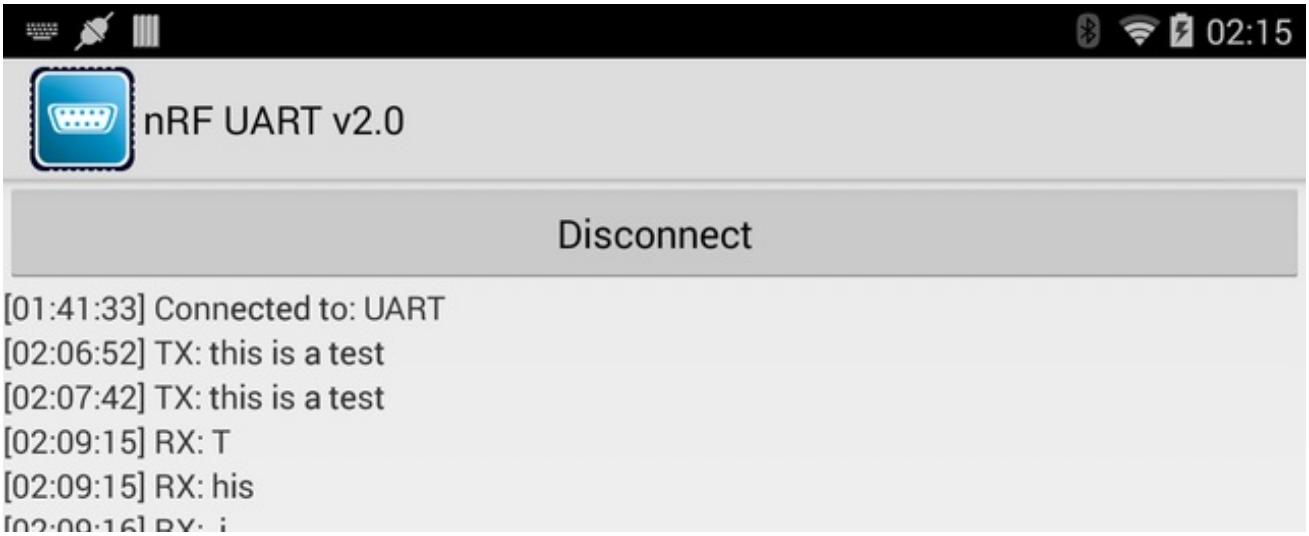


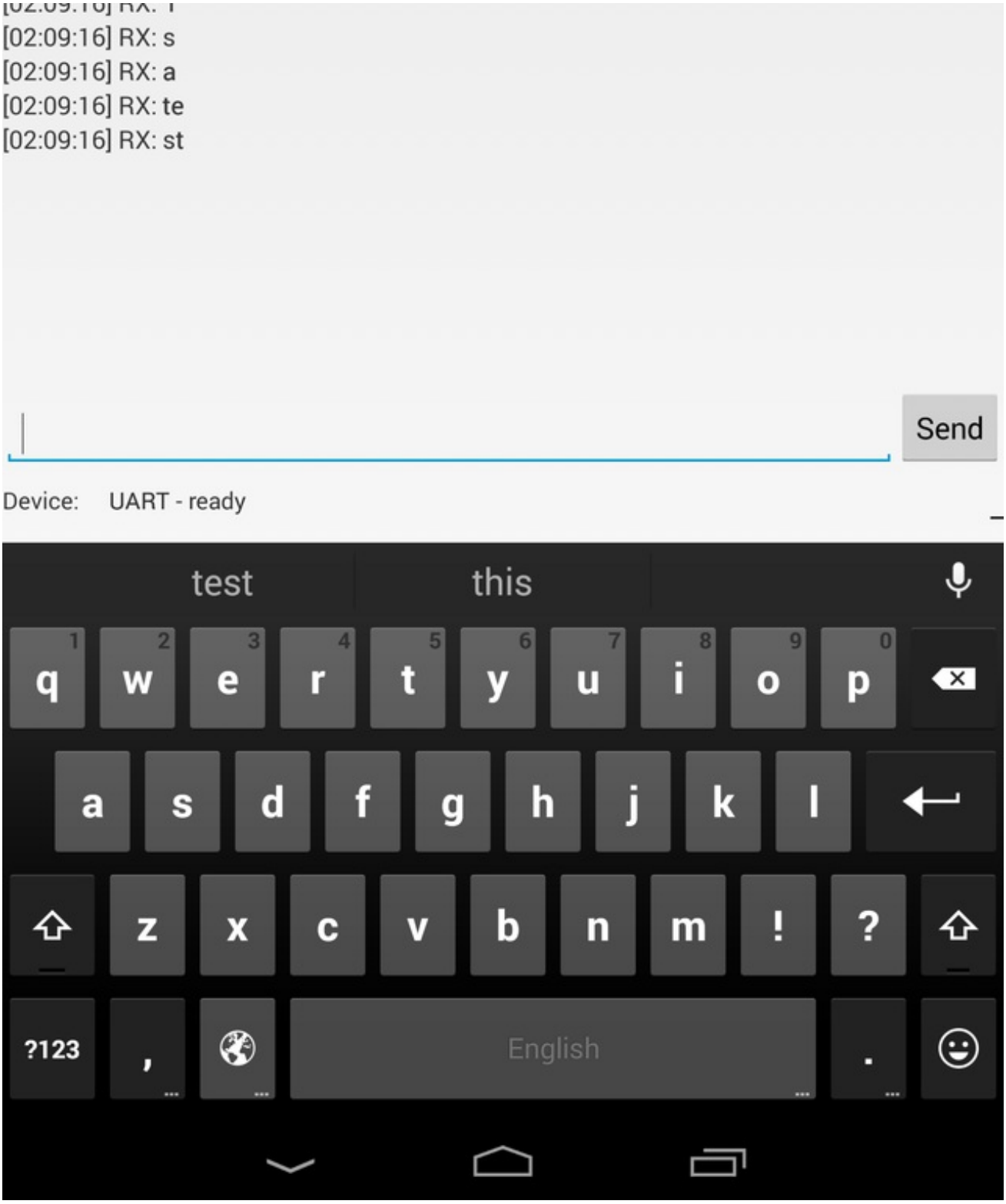
What this 4-byte 'Bluetooth Attribute Protocol' packet is actually saying is that attribute 0x001C (the location of the TX characteristic in the attribute table) has been updated, and the new value is '0x54', which corresponds to the letter 'T'.

Scrolling a bit further down we can see an example where more than one character was sent in a single transction ('te' in this case):



The results of this transaction in the nRF UART application can be seen below:





Closing Wireshark and nRF-Sniffer

When you're done debugging, you can save the session to a file for later analysis, or just close Wireshark right away and then close the nRF-Sniffer console window to end the debug session.

Moving Forward

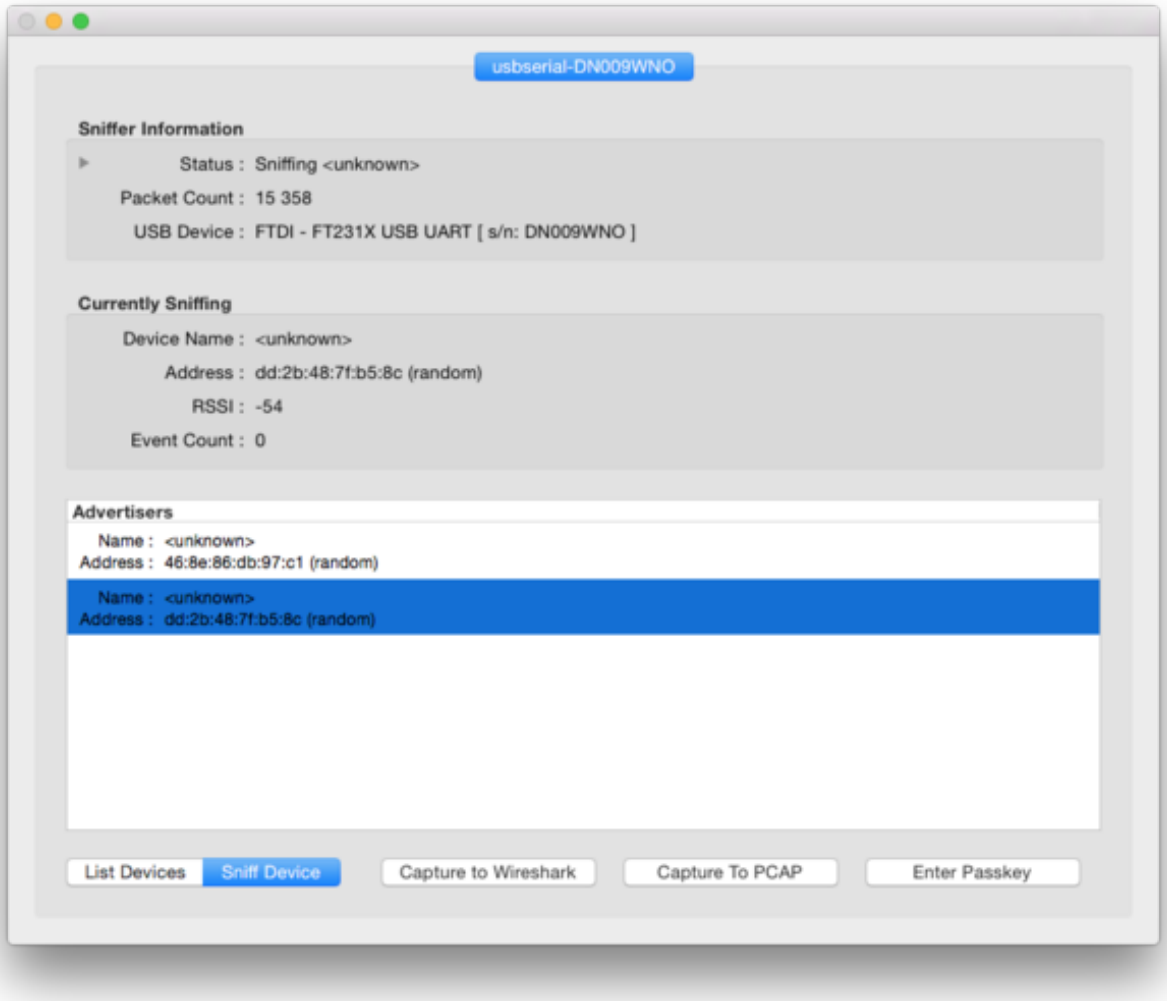
A sniffer is an incredibly powerful and valuable tool debugging your own hardware, reverse engineering existing BLE peripherals, or just to learn the ins and outs of how Bluetooth Low Energy actually works on the a packet by packet level.

You won't learn everything there is to know about BLE in a day, but a good book on BLE, a copy of the Bluetooth 4.1 Core Specification and a sniffer will go a long way to teaching you most of the important things there is to know about BLE in the real world.

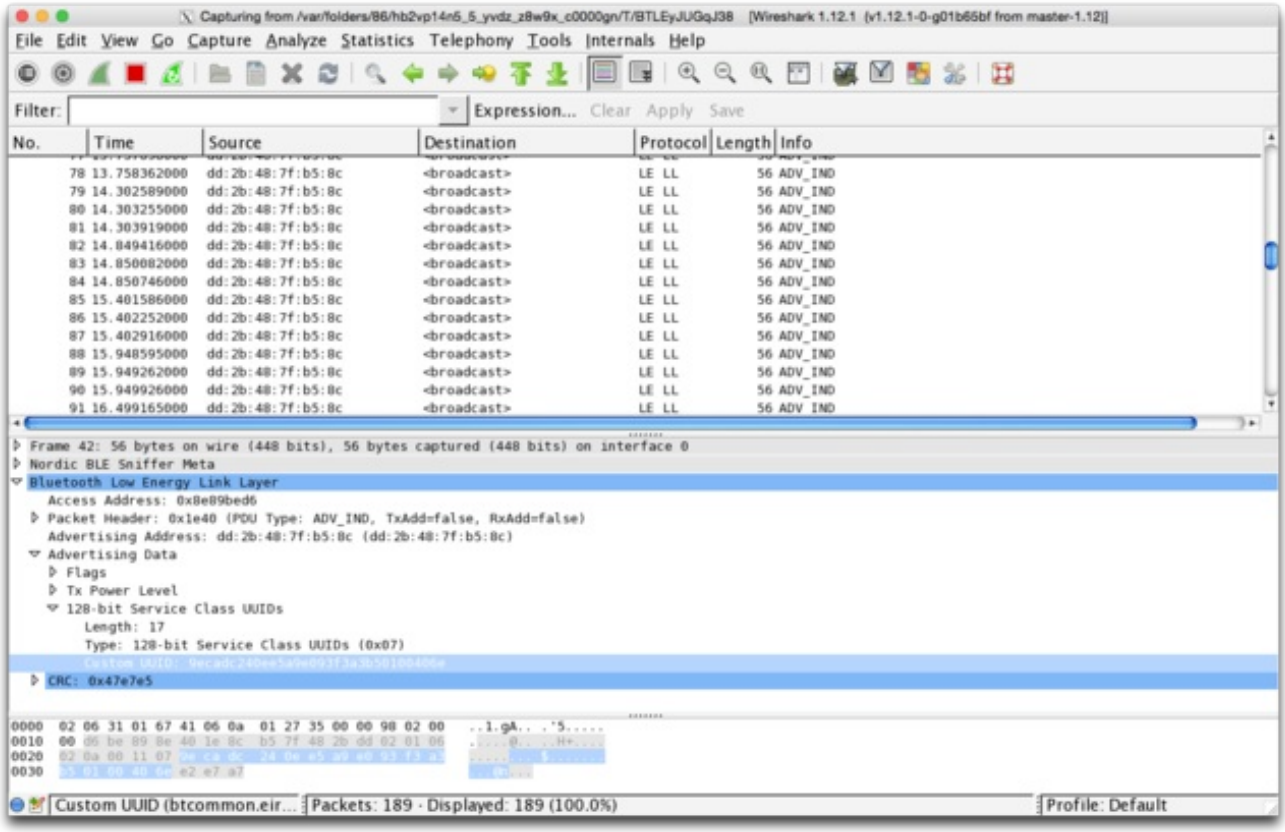
OS X Support

If you are running OS X 10.9 or higher, you can also use the sniffer on OS X using the nrf-ble-sniffer-osx (http://adafru.it/ft6) package from Roland King. (Make sure you have the latest version, as of 20 June 2015, which is now compatible with the FTDI chip used on the Adafruit board.)

Setup instructions are available on the wiki page (http://adafru.it/ft7) for the project.



Please note that there can be a long delay (30-60 seconds) before Wireshark shows up using the tool, due to the X11 startup time, etc.



If Wireshark doesn't show up and X11 has been installed correctly, try forcing X11 closed and trying a second time. The startup process can sometimes stall.

Python API

The Python interface requires a custom Wireshark library for Linux. We're currently working on adding support for this. Please use the Windows or OS X utility until the update is available.

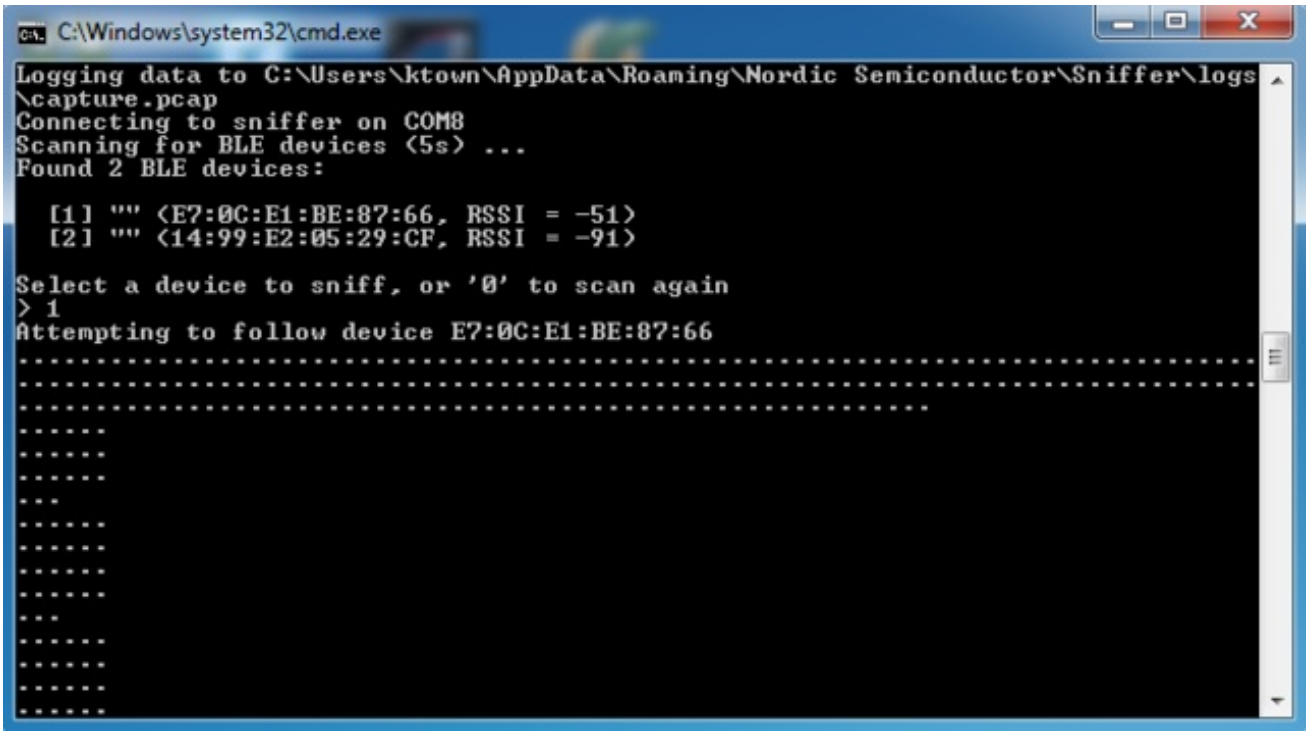
Nordic provides a Python API for their sniffer firmware that makes it possible for us to use the sniffer on any platform, and we've put together a basic wrapper for this API to help you get started.

We've tested this wrapper with Python 2.7 on the following platforms:

- OS X 10.10
- Windows 7 x64
- Ubuntu 14.04

To stream live data into Wireshark the way the official Windows app (http://adafru.it/k6F) from Nordic does you will need to compile a Wireshark utility that creates a name pipe that data gets pushed through.

To keep things simple, though, you can also just log sniffed traffic directly to a libpcap file, which can be opened directly in Wireshark when you are done, which is the easiest solution and what we'll be demonstrating here:



Requirements

To use the example we provide for the Python API, you will require the following utilities:

- Python 2.7.x (http://adafru.it/edH) (we tested with 2.7.6)
- pySerial (http://adafru.it/cLU)

If you're new to Python and pySerial, have a look at our Instaling Python and PySerial (http://adafru.it/k7c) guide by Simon Monk.

Download the API

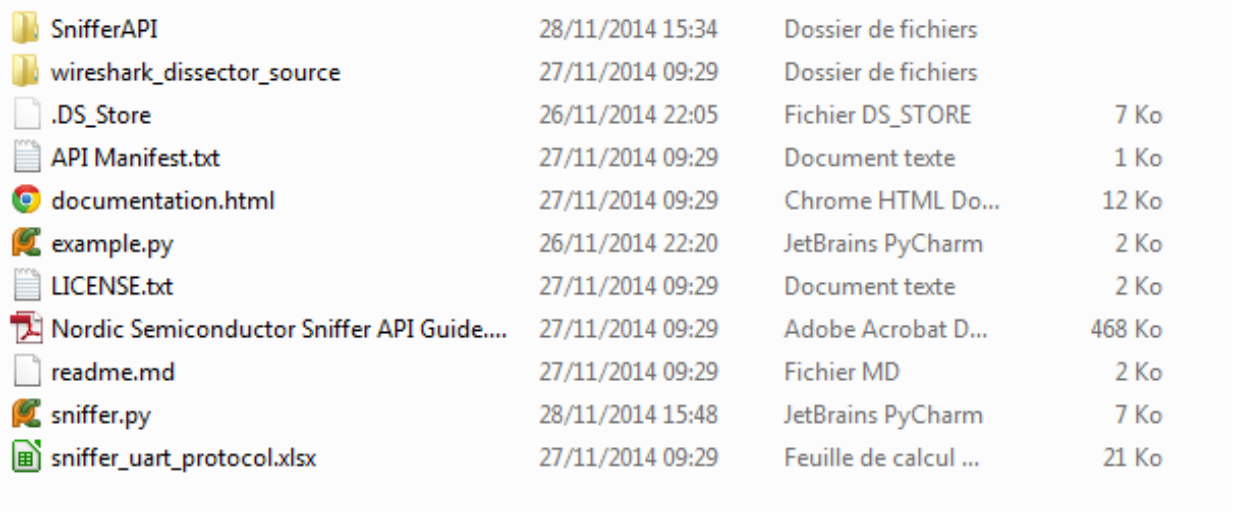
Once you have Python and pySerial installed on your system, you will need to download a copy of the Python API.

The latest version of the API is always available on Github (http://adafru.it/edJ), but you can also download a .zip file of the latest code directly using the button below:

Download the Python API from Github

http://adafru.it/edK

Unzipping the file should give you a file structure resembing the image below:



Using the sniffer.py Wrapper

To help you get started, we've made an easy to use wrapper called sniffer.py:

```
$ sudo python sniffer.py -h
usage: sniffer.py [-h] [-v] serialport

Interacts with the Bluefruit LE Friend Sniffer firmware

positional arguments:
  serialport     serial port location ('COM14', '/dev/tty.usbserial-DN009WNO',
                 etc.)

optional arguments:
  -h, --help     show this help message and exit
  -v, --verbose  verbose mode (all serial traffic is displayed)
```

It takes a single argument, the COM port location, which will be something like 'COM15' on Windows, '/dev/ttyACM*' on Linux, or '/dev/tty.usbserial*' on OS X.

Linux

To run the sniffer wrapper on Linux, enter the following command (changing the serial port as necessary):

```
$ sudo python sniffer.py /dev/ttyACM0
```

OS X

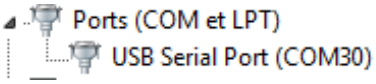
To run the sniffer wrapper on OS X, enter the following command (changing the serial port as necessary):

```
$ python sniffer.py /dev/tty.usbserial-DN009MP6
```

Windows

To run the sniffer wrapper on Windows, enter the following command (changing the serial port as necessary):

> You can find the serial port used by the Bluefruit LE Sniffer by opening the Device Manager on your system and looking in the 'Ports' category:




```
python sniffer.py COM30
```

Scanning for Devices

If the wrapper was able to connect to the Bluefruit LE Sniffer, it will perform a 5 second scan for Bluetooth Low Energy devices in range, and ask you which device you want to listen to:

```
$ sudo python sniffer.py /dev/ttyACM0
[sudo] password for ktown:
Logging data to logs/capture.pcap
Connecting to sniffer on /dev/ttyACM0
Scanning for BLE devices (5s) ...
Found 2 BLE devices:

  [1] "" (E7:0C:E1:BE:87:66, RSSI = -52)
  [2] "" (14:99:E2:05:29:CF, RSSI = -94)

Select a device to sniff, or '0' to scan again
>
```

Once you select a device, it will start scanning that specific device, and you will see an update every second of the number of packets 'sniffed' from the device (where each '.' represents a packet):

```
Select a device to sniff, or '0' to scan again
> 1
Attempting to follow device E7:0C:E1:BE:87:66
.....................................................................................................................
.....................
.....................
.....................
.....................
```

Locating the Log File

Once you've sniffed enough data, simply type **CTRL+C** to stop, and locate the libpcap log file at the path mentionned by the tool. This will normally be:

- **Windows**: 'C:\Users\ktown\AppData\Roaming\Nordic Semiconductor\Sniffer\logs

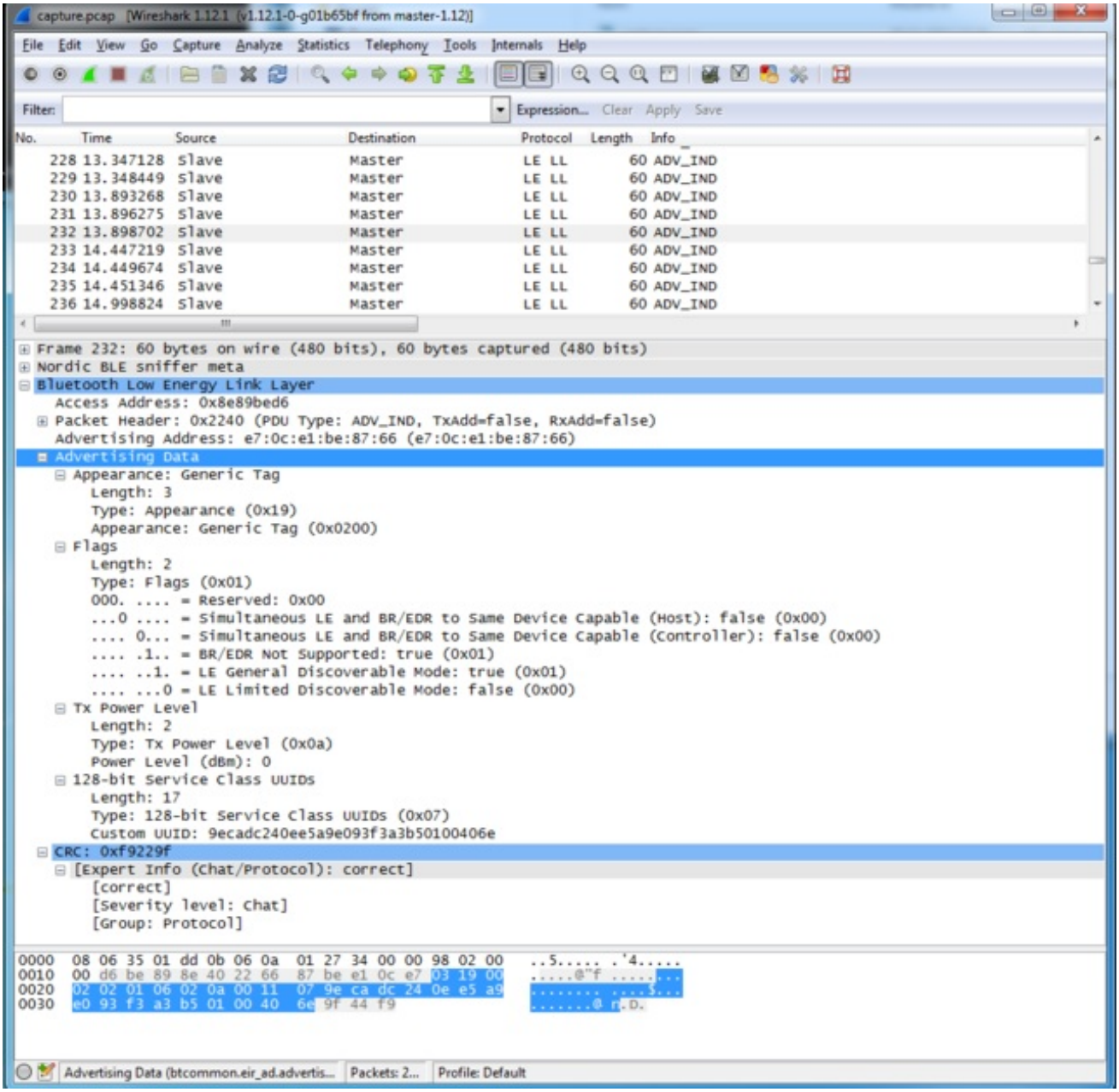
\capture.pcap' (this will of course change based on your username)

- **OS X/Linux**: 'logs/capture.pcap' (relative to the location of the Python API)

Analyze Data in Wireshark

At this point, you simply need to open the capture.pcap file in Wireshark, and you can analyze the sniffed data!

The image below shows an advertising packet from a factory default Bluefruit LE Friend (http://adafru.it/edl) board:



Note that the utility will start sniffing data as soon as you connect to the Bluefruit LE Sniffer, so early packets in the log file might contain advertising packets from other devices in range. It will only start filtering packets once you select a specific device via the selection dialogue.

For information on how to use Wireshark, have a look at the notes on the official nRF Sniffer utility (http://adafru.it/k6F), which describes some of the packet types you might encounter working with Bluetooth Low Energy.

FAQs

When I connect to a Central device, I don't see any connection data, but when I disconnect I see the advertising packets again. How do I capture data with a connected peripheral?

This is a limitation of the sniffer firmware from Nordic. Advertising in Bluetooth Low Energy happens on three dedicated channels, each running at it's own frequency. For the sniffer to 'follow' the connection it needs to be looking at the right channel when the connection happens, and there is a 2/3 chance that it is looking at another channel at any given moment.

To capture the connection and see data exchanges post connection, you may need to connect several times until the channels are aligned between the sniffer and the BLE peripheral+central devices.

How do I convert between Sniffer and Bluefruit LE firmware using SWD?

Reflashing Bluefruit LE modules over SWD (ex. switching to the sniffer firmware and back) is **at your own risk and can lead to a bricked device, and we can't offer any support for this operation!** You're on your own here, and there are unfortunately 1,000,000 things that can go wrong, which is why we offer two separate Bluefruit LE Friend boards -- the sniffer and the normal Bluefruit LE Friend board with the non-sniffer firmware, which provides a bootloader with fail safe features that prevents you from ever bricking boards via OTA updates.

AdaLink (SWD/JTAG Debugger Wrapper)

Transitioning between the two board types (sniffer and Bluefruit LE module) is unfortunately not a risk-free operation, and requires external hardware, software and know-how to get right, which is why it isn't covered by our support team.

That said ... if you're determined to go down that lonely road, and you have a Segger J-Link (http://adafru.it/fYU) (which is what we use internally for production and development), or have already erased your Bluefruit LE device, you should have a look at AdaLink (http://adafru.it/fPq), which is the tool we use internally to flash the four files required to restore a Bluefruit LE module. (Note: recent version of AdaLink also support the cheaper STLink/V2 (http://adafru.it/2548), though the J-Link is generally more robust if you are going to purchase a debugger for long term use.)

To go from the sniffer to Bluefruit LE firmware the mandatory Intel Hex files are available in the Bluefruit LE Firmware repo (http://adafru.it/edX). You will need to flash:

- An appropriate bootloader image
- An appropriate SoftDevice image
- The Bluefruit LE firmware image
- The matching signature file containing a CRC check so that the bootloader accepts the firmware image above (located in the same folder as the firmware image)

The appropriate files are generally listed in the version control .xml file (http://adafru.it/fPr) in the firmware repository.

If you are trying to flash the sniffer firmware (at your own risk!), you only need to flash a single .hex file, which you can find here (http://adafru.it/fYV). The sniffer doesn't require a SoftDevice image, and doesn't use the fail-safe bootloader -- which is why changing is a one way and risky operation if you don't have a supported SWD debugger.

Adafruit_nF51822_Flasher

We also have an internal python tool available that sits one level higher than AdaLink (referenced above), and makes it easier to flash specific versions of the official firmware to a Bluefruit LE module. For details, see the Adafruit_nRF51822_Flasher (http://adafru.it/fVL) repo.