adafruit learning system

Introducing the Adafruit Bluefruit LE Friend

Created by Kevin Townsend

Last updated on 2015-06-29 04:50:11 PM EDT

Guide Contents

Guide Contents 2
Introduction 7
Why Not Just Use a BLE USB Dongle? 7
So it's a Fancy Pants Wireless UART Adapter? 8
Why Use Adafruit's Module? 8
Getting Started 8
QuickStart Guide 10
HW Setup 11
Software Requirements 11
HW Layout 11
- Mode Selection Switch 11
- TXD/RXD Status LEDs 12
- DFU Mode Switch 12
- Mode Indicator LED 12
- Connection Status LED 12

Operating Modes 14
Data Mode 14
Command Mode 14
DFU Mode 15
Terminal Settings 16
TerraTerm (Windows) 16
CoolTerm (OS X) 17
Testing the Terminal Config 18
UART Test 20
BLEFriend Configuration 20
nRF UART Configuration 20
Sample Video 26
Factory Reset 28
Command Mode 29
Hayes/AT Commands 29
Test Command Mode '=?' 29

Write Command Mode '=xxx' 30
Execute Mode 30
Read Command Mode '?' 31
Standard AT 32
AT 32
ATI 32
ATZ 33
ATE 33
+++ 34
General Purpose 35
AT+FACTORYRESET 35
AT+DFU 35
AT+HELP 36
Hardware 37
AT+HWADC 37
AT+HWGETDIETEMP 37
AT+HWGPIO 37
AT+HWGPIOMODE 39
AT+HWI2CSCAN 40
AT+HWVBAT 40
AT+HWRANDOM 41
Beacon 42
AT+BLEBEACON 42
AT+BLEURIBEACON 44
BLE Generic 46
AT+BLEPOWERLEVEL 46
AT+BLEGETADDRTYPE 47
AT+BLEGETADDR 47
AT+BLEGETPEERADDR 48
AT+BLEGETRSSI 48
BLE Services 50
AT+BLEUARTTX 50

AT+BLEUARTRX 50
AT+BLEKEYBOARDEN 51
AT+BLEKEYBOARD 52
AT+BLEKEYBOARDCODE 52
Modifier Values 53
BLE GAP 54
AT+GAPGETCONN 54
AT+GAPDISCONNECT 54
AT+GAPDEVNAME 55
AT+GAPDELBONDS 55
AT+GAPINTERVALS 56
AT+GAPSTARTADV 57
AT+GAPSTOPADV 57
AT+GAPSETADVDATA 58
BLE GATT 61
AT+GATTCLEAR 61
AT+GATTADDSERVICE 61
AT+GATTADDCHAR 62
AT+GATTCHAR 64
AT+GATTLIST 65
Debug 67
AT+DBGMEMRD 67
AT+DBGNVMREAD 67
AT+DBGSTACKSIZE 68
AT+DBGSTACKDUMP 68
History 72
Version 0.6.5 72
Version 0.6.2 72
Version 0.5.0 72
Version 0.4.7 73
Version 0.3.0 73
Command Examples 74

Heart Rate Monitor Service 74
Python Script 75
Field Updates 79
Requirements 79
Entering DFU Mode 79
DFU Timeout 80
Firmware Images 80
DFU on iOS 81
Install nRF Toolbox 81
Using DFU in nRF Toolbox 83
Adding Custom Firmware 87
Transferring the Firmware via BLE 89
Put the BLEFriend in DFU Mode 99
Connecting to the DFU Service 99
DFU on Android (4.3+) 108
Verified Devices 108
Download nRF Toolbox 108
Load nRF Toolbox 108
Start the DFU Utility 110
Select the File Type and Hex File 111
Put the BLEFriend in DFU Mode 116
Select the Target DFU Device 117
Transfer the Firmware 118
BLE Sniffer 123
Select the Sniffer Target 123
Working with Wireshark 125
Capturing Exchanges Between Two Devices 127
Scan Response Packets 128
Connection Request 129
Write Request 129
Regular Data Requests 130
Notify Event Data 131
Closing Wireshark and nRF-Sniffer 134

Moving Forward 135
GATT Service Details 136
UART Service 136
UART Service 137
Characteristics 137
TX (0x0002) 137
RX (0x0003) 137

Introduction

The BLEFriend makes it easy to get any USB enabled device talking to your BLE enabled phone or tablet using a standard USB CDC connection.

In it's simplest form, it works on the same principle as a common USB/Serial adapter (the FTDI Friend (http://adafru.it/dQa), for example!). Any data that you enter via your favorite terminal emulator on your development machine will be transferred over the air to the connected phone or tablet, and vice versa.

Why Not Just Use a BLE USB Dongle?

Good question! You can get a $10.95 Bluetooth 4.0 USB dongle (http://adafru.it/ecb) from the store already, and it's a useful tool to have (particularly on the Raspberry Pi or BBB), but that won't solve some problems out of the box.

To start with, Bluez (Linux) has a decent learning curve (and doesn't run on OS X if you're a Mac user). Windows 7 doesn't even support Bluetooth Low Energy, and on OS X you'll have to sort through the native Bluetooth APIs and development tools that require an annual paid license and

specific license terms to access. There isn't a standard, open source, cross-platform way to talk BLE today.

With the BLEFriend, you can be up and running in under and hour on just about anything with a USB port, with easy migration across platforms and operating systems. It's not perfect (it's currently a peripheral mode only solution), but it's the easiest way you'll find to get any USB device talking to your iOS or Android device.

So it's a Fancy Pants Wireless UART Adapter?

The board is capable of much more than simulating a basic UART bridge (and this is still early days for the Bluefruit LE board family)! Thanks to an easy to learn AT command set (http://adafru.it/edo), you can also create you own basic GATT Services (http://adafru.it/ejP) and Characteristics, simulate Beacons (http://adafru.it/ei3), and change the way that the device advertises itself for other Bluetooth Low Energy devices to see.

To make sure that your device stays up to date and can benefit from the latest Bluefruit LE firmware from Adafruit, you can also update the firmware on your BLEFriend over the air (http://adafru.it/ejQ) using any supported iOS or Android device.

You can even pick up a sniffer edition of the board that comes pre-flashed with special firmware that turns your BLEFriend into a low cost Bluetooth Low Energy sniffer (http://adafru.it/ecn), capturing data and pushing it out to Wireshark. We currently offer this as a seperate product, though, since the firmware isn't compatible with the over the air bootloader used on the standard products, but we'll address this in the future with a tutorial for J-Link owners, allowing you to switch between modes using your SWD debugger.

Why Use Adafruit's Module?

There are plenty of BLE modules out there, with varying quality on the HW design as well as the firmware. We always try to keep the bar as high as possible at Adafruit, and one of the biggest advantages of the BLEFriend and the entire Bluefruit LE family is that **we wrote all of the firmware running on the devices ourelves from scratch.**

We control every line of code that runs on our modules ... and so we aren't at the mercy of any third party vendors who may or may not be interested in keeping their code up to date or catering to our customer's needs.

Because we control everything about the product, we add features that are important to *our* customers, benefit from being able to use the latest Bluetooth specifications, can solve any issues that do come up without having to persuade a half-hearted firmware engineer on the other side of the planet, and we can even change Bluetooth SoCs entirely if the need ever arises!

Getting Started

If you just want to get up and running quickly, this is the right guide for you, and the QuickStart section should have you up and running in no time.

If you're new to Bluetooth Low Energy, and want to get a high level overview of how data is organized and how devices communicate with each other, you might want to have a look at our Introduction to Bluetooth Low Energy (http://adafru.it/ech) learning guide, or buy Getting Started with Bluetooth Low Energy (http://adafru.it/1978) from the store.*

* Full disclosure: co-written by me :)

QuickStart Guide

The BLEFriend board is designed to be easy to use and get started with.

In most circumstance, the only thing you'll need is an FTDI driver, and a terminal emulator to start working with BLE from your development machine.

This guide will explain some of the different operating modes (http://adafru.it/ekA) that the BLEFriend board can be configure to operate in, how to setup your terminal emulator (http://adafru.it/edp) to start talking to the BLEFriend, and a basic example of sending bi-directional data (http://adafru.it/ekB)between the BLEFriend and a BLE-enabled phone or tablet.

The BLEFriend is (of course) capable of much more than basic UART data exchanges! You can use it to create custom GATT services and characteristics. You can emulate a Beacon (http://adafru.it/ei3) for indoor navigation purposes. You can update the firmware on your device (http://adafru.it/ejQ) over the air, or even develop your own firmware thanks to the on-board SWD connector if you possess a HW debugger like the Segger J-Link.

Depending on the version of the board you have or whether or not you have access to a J-Link, you can even use the BLEFriend as a powerful Bluetooth Low Energy sniffer (http://adafru.it/ekC) to debug or reverse engineer existing BLE devices, and write your own custom applications or drivers for existing HW!

Have a look through this quick start guide to familiarise yourself with the basics, though, and you should be able to quickly move on to more advanced topics in no time!

HW Setup

Software Requirements

Setting up the BLEFriend is super easy. All you need to start talking to the device is 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 BLEFriend 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

HW Layout

There are a few items on the BLEFriend you should be familiar with before you start working with it. To help you get started quickly, we've highlighted them in the image below:

Mode Selection Switch

This switch can be moved between 'CMD' (Command Mode) and 'UART' (Data Mode), which will change the way that the device behaves in your terminal emulator.

For more information on these two operating modes, see the Operating Modes (http://adafru.it/ekA) page in this learning guide.

TXD/RXD Status LEDs

These two LEDs are provided primarily for debug purposes to help you visualise the incoming and outgoing characters over the USB CDC interface.

DFU Mode Switch

Holding this switch down when you insert the device into the USB port will cause the device to enter a special 'DFU' mode, which allows you to update the firmware over the air.

For more information on DFU mode see the Field Updates (http://adafru.it/ejQ) page.

Mode Indicator LED

This LED is used to indicate the mode that the device is currently operating in (Data, Command or DFU).

Connection Status LED

This LED will be enabled when the BLEFriend has successfully established a connection with another BLE device, and it useful for debugging purposes.

ISSUE
ADAFRUIT INDUSTRIES
© 2014
DRAWN
KTOWN
TITLE
BLEFriend
REV
G
CHECKED
>CHECKED
DATE
11/10/2014 3:57:18 PM
DRG Nº
>DRGNO
DATE
>DATE
FILE: BLEFriend_MDBT40_REV-G
PAGE: 1/1

Operating Modes

BLEFriend modules can be operated in one of three modes:

Data Mode

Data mode makes use of the BLE UART Service, and turns the BLEFriend into a HW UART bridge between a BLE Central device (your phone or tablet) and your PC or USB-enabled device.

To use data mode, simply connect your BLEFriend module to the USB port on your PC, mode the mode selection switch to **UART** and start sending or receiving data at 9600 bps using your favorite terminal software.

If the MODE LED blinks twice followed by a three second delay you are in **Data Mode:**

Command Mode

Command mode is used to send configuration commands to the module or retrieve information about the module itself or the device connected on the other side of the BLE connection.

To use command mode, make sure that the mode selection switch is set to **CMD**, and enter a valid Hayes AT style command using your favorite terminal emulator at 9600 bps (for example 'ATI' to display some basic info about the module).

If the MODE LED blinks three times followed by a three second delay you are in **Command Mode**:

For more information on this operating mode see the dedicated Command Mode (http://adafru.it/edo) page in this learning guide.

DFU Mode

DFU Mode (which stands for 'device firmware upgrade' or sometimes 'device field update') is a special mode that allows you to update the firmware on the BLEFriend over the air using dedicated DFU applications available on iOS and Android.

This allows you to update your device with the latest BLEFriend firmware from Adafruit without having to purchase an external HW debugger like the Segger J-Link (http://adafru.it/e9G).

To enter DFU mode hold down the **DFU** button while inserting the BLEFriend into the USB port.

If the LED blinks at a constant rate, you know that you are in DFU mode:

For more information on DFU mode see the dedicate Device Field Update (DFU) page (http://adafru.it/ejQ).

Terminal Settings

Since the BLEFriend board uses UART for the data transport, you will need to configure your favorite terminal emulator software with the appropriate settings to send or receive data over UART.

The BLEFriend is configured to run with the following settings:

- **9600 bps**
- **HW flow control** (CTS+RTS)
- **8n1** (8-bit data, no parity, 1 stop bit)

TerraTerm (Windows)

If you are using Windows we recommend using TeraTerm (http://adafru.it/e9l), which should be configured as follows (via the 'Setup > Serial Port' menu):

CoolTerm (OS X)

If you are using OS X, we recommend using CoolTerm (http://adafru.it/e9J), a free and reasonably fully featured terminal emulator package.

You'll need to set the following configuration options in Coolterm to communicate with the Bluefruit LE Pro module:

Make sure you click the 'Connect' button to open the connection in CoolTerm, and 'Disconnect' to close the connection when you are done.

Testing the Terminal Config

To make sure the connection is properly configured, you can set the device in **Command Mode** (the LED should blink three times followed by a short delay in this mode), and enter the **ATI** command, as shown below (which will provide some basic information about the BLE module):

Make sure you are in CMD mode or you won't get any echo or AT command replies!

UART Test

To test out the BLE UART service that is used to transfer data between the BLEFriend and your Bluetooth Low Energy enabled mobile device, you can used the **nRF UART** application from Nordic Semiconductors for iOS (http://adafru.it/dd7) or Android (http://adafru.it/dd6).

This tutorial uses the Android version of nRF UART but the iOS UI is reasonably similar.

BLEFriend Configuration

Set the BLEFriend to **DATA MODE** by placing the mode selector switch (near the USB connector) to the **UART** position.

The 'MODE' indicator LED will blink two times and then pause for three seconds when you are in DATA mode:

Using your favorite terminal emulator (http://adafru.it/edp), connect to the BLEFriend at **9600 baud** with HW flow control enabled.

nRF UART Configuration

Open the nRF UART application on Android or iOS, and click the 'Connect' button

After clicking on 'Connect' you should see a list of BLE peripherals in range:

12:57
nRF UART v2.0
Connect
nRF UART v2.0
Select a device
UART
E4:C6:C7:31:95:11
Rssi = -73
Rssi = -90
14:99:E2:05:29:CF
Gear Fit (E52B)
68:48:98:B8:E5:2B
Rssi = -82
paired
Cancel
Send

Select the '**UART**' device by clicking on it from the popup selection dialogue, and a connection will be established between the BLEFriend and your phone or tablet.

NOTE: You know that you are connected to another device when the **CONN** LED is lit up on the BLEFriend:

Once you see the 'Connected To: UART' message in nRF UART, you can start sending and receiving data between the two devices using the mobile app or the terminal emulator.

Sending data from one device should cause the text to appear on the other end:

[12:57:05] Connected to: UART
testing
Send
Device: UART - ready
test ing
testing
te sting
a z e r t y u i o p
q s d f g h j k l m
w x c v b n '
?123
Français

On the BLEFriend side we can see the 'testing' string that was sent from nRF UART in the terminal emulator below:

Typing data into the terminal emulator will cause it to appear in the nRF UART application, as shown below:

The incoming data is broken up into small packets because it is transferred in chunks of up to 20 bytes per payload.

All data transfers in BLE occur at a rate controlled by the BLE central device (the phone or tablet). Every n milliseconds, the phone will ping the BLE peripheral (the BLEFriend) and check if there is any data available.

If anything is found in the UART FIFO, for example, it will be sent and data will start to accumulate again in the FIFO until the next connection event.

Sample Video

Download the screen capture below to see nRF UART in action, first receiving 'this is a test' from the BLEFriend (typed in on the terminal emulator), and then sending 'adafruit' out to the BLEFriend, where is will appear in the terminal emulator.

nRFUART_640.mp4

http://adafru.it/ea5

Depending on which terminal emulator SW you use, you would see something like this on your development machine running a similar demo yourself:

Factory Reset

As of version 0.5.0+ of the firmware, you can perform a factory reset by holding the DFU button down for 10s until the blue CONNECTED LED lights up, and then releasing the button.

If you have any problems with your BLEFriend module, you can perform a factory reset by entering command mode (set the mode selection switch to 'CMD'), and entering the following command in your terminal emulator:

```
AT+FACTORYRESET
```

This will erase the non volatile memory section where the config data is stored and reset everything to factory default values, and perform a system reset.

When you see the 'OK' response the device should be ready to use.

Command Mode

By placing the BLEFriend module in 'Command' mode (set the mode selection switch to **CMD**) you can enter a variety of Hayes AT style commands to configure the device or retrieve basic information about the module of BLE connection.

You can determine if you are in Command Mode by looking at the mode LED. It should blink three times followed by a three second pause, as shown below:

Hayes/AT Commands

When operating in command mode, the Bluefruit LE Pro modules use a Hayes AT-style command set (http://adafru.it/ebJ)to configure the device.

The advantage of an AT style command set is that it's easy to use in machine to machine communication, while still being somewhat user friendly for humans.

Test Command Mode '=?'

'Test' mode is used to check whether or not the specified command exists on the system or not.

Certain firmware versions or configurations may or may not include a specific command, and you can determine if the command is present by taking the command name and appending **'=?'** to it, as shown below

```
AT+BLESTARTADV=?
```

If the command is present, the device will reply with **'OK'**. If the command is not present, the device will reply with **'ERROR'**.

```
AT+BLESTARTADV=?
OK\r\n
AT+MISSINGCMD=?
ERROR\r\n
```

Write Command Mode '=xxx'

'Write' mode is used to assign specific value(s) to the command, such as changing the radio's transmit power level using the command we used above.

To write a value to the command, simple append an **'='** sign to the command followed by any paramater(s) you wish to write (other than a lone **'?'** character which will be interpretted as tet mode):

```
AT+BLEPOWERLEVEL=-8
```

If the write was successful, you will generally get an **'OK'** response on a new line, as shown below:

```
AT+BLEPOWERLEVEL=-8
OK\r\n
```

If there was a problem with the command (such as an invalid parameter) you will get an **'ERROR'** response on a new line, as shown below:

```
AT+BLEPOWERLEVEL=3
ERROR\r\n
```

Note: This particular error was generated because '3' is not a valid value for the AT+BLEPOWERLEVEL command. Entering '-4', '0' or '4' *would* succeed since these are all valid values for this command.

Execute Mode

'Execute' mode will cause the specific command to 'run', if possible, and will be used when the command name is entered with no additional parameters.

```
AT+FACTORYRESET
```

You might use execute mode to perform a factory reset, for example, by executing the AT+FACTORYRESET command as follows:

```
AT+FACTORYRESET
OK\r\n
```

NOTE: Many commands that are means to be read will perform the same action whether they are sent to the command parser in 'execute' or 'read' mode. For example, the following commands will produce identical results:

```
AT+BLEGETPOWERLEVEL
-4\r\n
OK\r\n
AT+BLEGETPOWERLEVEL?
-4\r\n
OK\r\n
```

If the command doesn't support execute mode, the response will normally be **'ERROR'** on a new line.

Read Command Mode '?'

'Read' mode is used to read the current value of a command.

Not every command supports read mode, but you generally use this to retrieve information like the current transmit power level for the radio by appending a **'?'** to the command, as shown below:

```
AT+BLEPOWERLEVEL?
```

If the command doesn't support read mode or if there was a problem with the request, you will normally get an **'ERROR'** response.

If the command read was successful, you will normally get the read results followed by **'OK'** on a new line, as shown below:

```
AT+BLEPOWERLEVEL?
-4\r\n
OK\r\n
```

Note: For simple commands, 'Read' mode and 'Execute' mode behave identically.

Standard AT

The following standard Hayes/AT commands are available on Bluefruit LE modules:

AT

Acts as a ping to check if we are in command mode. If we are in command mode, we should receive the 'OK' response.

Codebase Revision: 0.3.0

Parameters: None

Output: None

```
AT
OK
```

ATI

Displays basic information about the Bluefruit module.

Codebase Revision: 0.3.0

Parameters: None

Output: Displays the following values:

- Board Name
- Microcontroller/Radio SoC Name
- Unique Serial Number
- Core Bluefruit Codebase Revision
- Project Firmware Revision
- Firmware Build Date
- Softdevice, Softdevice Version, Bootloader Version (0.5.0+)

```
ATI
BLEFRIEND
nRF51822 QFAAG00
FB462DF92A2C8656
0.5.0
0.5.0
Feb 24 2015
S110 7.1.0, 0.0
OK
```

Updates:

- Version **0.4.7+** of the firmware adds the chip revision after the chip name if it can be detected (ex. 'nRF51822 QFAAG00').
- Version **0.5.0+** of the firmware adds a new 7th record containing the softdevice, softdevice version and bootloader version (ex. 'S110 7.1.0, 0.0').

ATZ

Performs a system reset.

Codebase Revision: 0.3.0

Parameters: None

Output: None

```
ATZ
OK
```

ATE

Enables or disables echo of input characters with the AT parser

Codebase Revision: 0.3.0

Parameters: '1' = enable echo, '0' = disable echo

Output: None

```
# Disable echo support
ATE=0
OK
#Enable echo support
ATE=1
OK
```

+++

Dynamically switches between DATA and COMMAND mode without changing the physical CMD/UART select switch.

When you are in COMMAND mode, entering '+++\n' or '+++\r\n' will cause the module to switch to DATA mode, and anything typed into the console will go direct to the BLUE UART service.

To switch from DATA mode back to COMMAND mode, simply enter '+++\n' or '+++\r\n' again (be sure to include the new line character!), and a new 'OK' response will be displayed letting you know that you are back in COMMAND mode (see the two 'OK' entries in the sample code below).

Codebase Revision: 0.4.7

Parameters: None

Output: None

Note that +++ can also be used on the mobile device to send and receive AT command on iOS or Android, though this should always be used with care.

```
ATI
BLEFRIEND
nRF51822 QFAAG00
B122AAC33F3D2296
0.4.6
0.4.6
Dec 22 2014
OK
+++
OK
OK
```

General Purpose

The following general purpose commands are available on all Bluefruit LE modules:

AT+FACTORYRESET

Clears any user config data from non-volatile memory and performs a factory reset before resetting the Bluefruit module.

Codebase Revision: 0.3.0

Parameters: None

Output: None

```
AT+FACTORYRESET
OK
```

As of version 0.5.0+ of the firmware, you can perform a factory reset by holding the DFU button down for 10s until the blue CONNECTED LED lights up, and then releasing the button.

AT+DFU

Forces the module into DFU mode, allowing over the air firmware updates using a dedicated DFU app on iOS or Android.

Codebase Revision: 0.3.0

Parameters: None

Output: None

The AT parser will no longer responsd after the AT+DFU command is entered, since normal program execution effectively halts and a full system reset is performed to start the bootloader code

```
AT+DFU
OK
```

AT+HELP

Displays a comma-separated list of all AT parser commands available on the system.

Codebase Version: 0.3.0

Parameters: None

Output: A comma-separated list of all AT parser commands available on the system.

The sample code below may not match future firmware releases and is provided for illustration purposes only

```
AT+HELP
AT+FACTORYRESET,AT+DFU,ATZ,ATI,ATE,AT+DBGMEMRD,AT+DBGNVMRD,AT+HWLEDPOLARITY,AT-
OK
```

Hardware

The following commands allow you to interact with the low level HW on the Bluefruit LE module, such as reading or toggling the GPIO pins, performing an ADC conversion ,etc.:

AT+HWADC

Performs an ADC conversion on the specified ADC pin

Codebase Revision: 0.3.0

Parameters: The ADC channel (0..7)

Output: The results of the ADC conversion

```
AT+HWADC=0
178
OK
```

AT+HWGETDIETEMP

Gets the temperature in degree celcius of the BLE module's die. This can be used for debug purposes (higher die temperature generally means higher current consumption), but does not corresponds to ambient temperature and can nto be used as a replacement for a normal temperature sensor.

Codebase Revision: 0.3.0

Parameters: None

Output: The die temperature in degrees celcius

```
AT+HWGETDIETEMP
32.25
OK
```

AT+HWGPIO

Gets or sets the value of the specified GPIO pin (depending on the pin's mode).

Codebase Revision: 0.3.0

Parameters: The parameters for this command change depending on the pin mode.

OUTPUT MODE: The following comma-separated parameters can be used when updating a pin that is set as an output:

- Pin numbers
- Pin state, where:
    - 0 = clear the pin (logic low/GND)
    - 1 = set the pin (logic high/VCC)

INPUT MODE: To read the current state of an input pin or a pin that has been configured as an output, enter the pin number as a single parameter.

Output: The pin state if you are reading an input or checking the state of an input pin (meaning only 1 parameter is supplied, the pin number), where:

- 0 means the pin is logic low/GND
- 1 means the pin is logic high/VCC

Trying to set the value of a pin that has not been configured as an output will produce an 'ERROR' response.

Some pins are reserved for specific functions on Bluefruit modules and can not be used as GPIO. If you try to make use of a reserved pin number an 'ERROR' response will be generated.

```
# Set pin 14 HIGH
AT+HWGPIO=14,1
OK

# Set pin 14 LOW
AT+HWGPIO=14,0
OK

# Read the current state of pin 14
AT+HWGPIO=14
0
OK

# Try to update a pin that is not set as an output
AT+HWGPIOMODE=14,0
OK
AT+HWGPIO=14,1
ERROR
```

AT+HWGPIOMODE

This will set the mode for the specified GPIO pin (input, output, etc.).

Codebase Revision: 0.3.0

Parameters: This command one or two values (comma-separated in the case of two parameters being used):

- The pin number
- The new GPIO mode, where:
  - 0 = Input
  - 1 = Output
  - 2 = Input with pullup enabled
  - 3 = Input with pulldown enabled

Output: If a single parameters is passed (the GPIO pin number) the current pin mode will be returned.

Some pins are reserved for specific functions on Bluefruit modules and can not be used as GPIO. If you try to make use of a reserved pin number an 'ERROR' response will be generated.

```
# Configure pin 14 as an output
AT+HWGPIOMODE=14,0
OK

# Get the current mode for pin 14
AT+HWPGIOMODE=14
0
OK
```

AT+HWI2CSCAN

Scans the I2C bus to try to detect any connected I2C devices, and returns the address of devices that were found during the scan process.

Codebase Revision: 0.3.0

Parameters: None

Output: A comma-separated list of any I2C address that were found while scanning the valid address range on the I2C bus, or nothing is no devices were found.

```
# I2C scan with two devices detected
AT+HWI2CSCAN
0x23,0x35
OK

# I2C scan with no devices detected
AT+HWI2CSCAN
OK
```

AT+HWVBAT

Returns the main power supply voltage level in millivolts

Codebase Revision: 0.3.0

Parameters: None

Output: The VBAT level in millivolts

```
AT+HWVBAT
3248
OK
```

AT+HWRANDOM

Generates a random 32-bit number using the HW random number generator on the nRF51822 (based on white noise).

Codebase Revision: 0.4.7

Parameters: None

Output: A random 32-bit hexadecimal value (ex. '0x12345678')

```
AT+HWRANDOM
0x769ED823
OK
```

Beacon

Bluefruit LE modules can be configured to act as 'Beacons' using the following commands:

AT+BLEBEACON

Codebase Revision: 0.3.0

Parameters: The following comma-separated parameters are required to enable beacon mode:

- Bluetooth Manufacturer ID (uint16_t)
- 128-bit UUID
- Major Value (uint16_t)
- Minor Value (uint16_t)
- RSSI @ 1m (int8_t)

Output: None

```
# Enable Apple iBeacon emulation
# Manufacturer ID = 0x004C
AT+BLEBEACON=0x004C,01-12-23-34-45-56-67-78-89-9A-AB-BC-CD-DE-EF-F0,0x0000,0x0000,-59
OK
# Reset to change the advertising data
ATZ
OK

# Enable Nordic Beacon emulation
# Manufacturer ID = 0x0059
AT+BLEBEACON=0x0059,01-12-23-34-45-56-67-78-89-9A-AB-BC-CD-DE-EF-F0,0x0000,0x0000,-59
OK
# Reset to change the advertising data
ATZ
OK
```

AT+BLEBEACON will cause the beacon data to be stored in non-volatile config memory on the Bluefruit LE module, and these values will be persisted across system resets and power cycles. To remove or clear the beacon data you need to enter the 'AT+FACTORYRESET' command in command mode.

Entering Nordic Beacon emulation using the sample code above, you can see the simulated beacon in Nordic's 'Beacon Config' tool below:

01:29
Beacon config
nRF Beacon
IDENTITY
UUID
01122334-4556-6778-899a-abbccddeeff0
MAJOR
0
MINOR
0
NOTIFY
EVENT
Near
ACTION
Show Mona Lisa
STATUS
ENABLED
OUI
BEACON CONFIG

AT+BLEURIBEACON

Converts the specified URI into a UriBeacon (http://adafru.it/edk) advertising packet, and configures the module to advertise as a UriBeacon (part of Google's Physical Web (http://adafru.it/ehZ) project).

To view the UriBeacon URIs you can use one of the following mobile applications:

- Android 4.3+: Physical Web (http://adafru.it/edi) on the Google Play Store
- iOS: Physical Web (http://adafru.it/edj) in Apple's App Store

Codebase Revision: 0.4.7

Parameters: The URI to encode (ex. http://www.adafruit.com/blog (http://adafru.it/ei0))

Output: None of a valid URI was entered (length is acceptable, etc.).

```
AT+BLEURIBEACON=http://www.adafruit.com/blog
OK

# Reset to change the advertising data
ATZ
OK
```

If the supplied URI is too long you will get the following output:

```
AT+BLEURIBEACON=http://www.adafruit.com/this/uri/is/too/long
URL is too long
ERROR
```

If the URI that you are trying to encode is too long, try using a shortening service like bit.ly, and encode the shortened URI.

BLE Generic

The following general purpose BLE commands are available on Bluefruit LE modules:

AT+BLEPOWERLEVEL

Gets or sets the current transmit power level for the module's radio (higher transmit power equals better range, lower transmit power equals better battery life).

Codebase Revision: 0.3.0

Parameters: The TX power level (in dBm), which can be one of the following values (from lowest to higher transmit power):

- -40
- -20
- -16
- -12
- -8
- -4
- 0
- 4

Output: The current transmit power level (in dBm)

> The updated power level will take affect as soon as the command is entered. If the device isn't connected to another device, advertising will stop momentarily and then restart once the new power level has taken affect.

```
# Get the current TX power level (in dBm)
AT+BLEPOWERLEVEL
0
OK

# Set the TX power level to 4dBm (maximum value)
AT+BLEPOWERLEVEL=4
OK

# Set the TX power level to -12dBm (better battery life)
AT+BLEPOWERLEVEL=-12
OK

# Set the TX power level to an invalid value
AT+BLEPOWERLEVEL=-3
ERROR
```

AT+BLEGETADDRTYPE

Gets the address type (for the 48-bit BLE device address).

Normally this will be '1' (random), which means that the module uses a 48-bit address that was randomly generated during the manufacturing process and written to the die by the manufacturer.

Random does not mean that the device address is randomly generated every time, only that a one-time random number is used.

Codebase Revision: 0.3.0

Parameters: None

Output: The address type, which can be one of the following values:

- 0 = public
- 1 = random

```
AT+BLEGETADDRTYPE
1
OK
```

AT+BLEGETADDR

Gets the 48-bit BLE device address.

Codebase Revision: 0.3.0

Parameters: None

Output: The 48-bit BLE device address in the following format: 'AA:BB:CC:DD:EE:FF'

```
AT+BLEGETADDR
E4:C6:C7:31:95:11
OK
```

AT+BLEGETPEERADDR

Gets the 48-bit address of the peer (central) device we are connected to.

Codebase Revision: 0.6.5

Parameters: None

Output: The 48-bit address of the connected central device in hex format. The command will return **ERROR** if we are not connected to a central device.

> Please note that the address returned by the central device is almost always a random value that will change over time, and this value should generally not be trusted. This command is provided for certain edge cases, but is not useful in most day to day scenarios.

```
AT+BLEGETPEERADDR
48:B2:26:E6:C1:1D
OK
```

AT+BLEGETRSSI

Gets the RSSI value (Received Signal Strength Indicator), which can be used to estimate the reliability of data transmission between two devices (the lower the number the better).

Codebase Revision: 0.3.0

Parameters: None

Output: The RSSI level (in dBm) if we are connected to a device, otherwise '0'

```
# Connected to an external device
AT+BLEGETRSSI
-46
OK

# Not connected to an external device
AT+BLEGETRSSI
0
OK
```

BLE Services

AT+BLEUARTTX

This command will transmit the specified text message out via the UART Service (http://adafru.it/ekD) while you are running in Command Mode.

Codebase Revision: 0.3.0

Parameters: The message payload to transmit (20 characters max)

Output: None

> You must be connected to another device for this command to execute ('ERROR' will be displayed if no connection has been established).

```
# Send a string when connected to another device
AT+BLEUARTTX=THIS IS A TEST
OK

# Send a string when not connected
AT+BLEUARTTX=THIS IS A TEST
ERROR
```

As of firmware release **0.6.2** and higher, AT+BLEUARTTX can accept a limited set of escape code sequences:

- \r = carriage return
- \n = new line
- \t = tab
- \b = backspace
- \\ = backward slash

AT+BLEUARTRX

This command will dump the UART service (http://adafru.it/ekD)'s RX buffer to the display if any data has been received from from the UART service while running in Command Mode. The data will be removed from the buffer once it is displayed using this command.

Any characters left in the buffer when switching back to Data Mode will cause the buffered characters to be displayed as soon as the mode switch is complete (within the limits of available buffer space, typically 256 characters).

Codebase Revision: 0.3.0

Parameters: None

Output: The RX buffer's content if any data is available, otherwise nothing.

```
# Command results when data is available
AT+BLEUARTRX
Sent from Android
OK

# Command results when no data is available
AT+BLEUARTRX
OK
```

The following commands allow you to interact with mandatory GATT services present on Bluefruit LE Pro modules like the BLEFRiend when running in Command Mode.

AT+BLEKEYBOARDEN

This command will enable GATT over HID (GoH) keyboard support, which allows you to emulate a keyboard on supported iOS and Android devices. By default HID keyboard support is disabled, so you need to set BLEKEYBOARDEN to 1 and then perform a system reset before the keyboard will be enumerated and appear in the Bluetooth preferences on your phone, where if can be bonded as a BLE keyboard.

Codebase Revision: 0.5.0

Parameters: 1 or 0 (1 = enable, 0 = disable)

Output: None

You must perform a system reset (ATZ) before the changes take effect!

Before you can use your HID over GATT keyboard, you will need to bond your mobile device with the Bluefruit LE module in the Bluetooth preferences panel.

```
# Enable BLE keyboard support then reset
AT+BLEKEYBOARDEN=1
OK
ATZ
OK

# Diable BLE keyboard support then reset
AT+BLEKEYBOARDEN=0
OK
ATZ
OK
```

AT+BLEKEYBOARD

Sends text data over the BLE keyboard interface (if it has previously been enabled via AT+BLEKEYBOARDEN).

Any valid alpha-numeric character can be sent, and the following escape sequences are also supported:

- \r - Carriage Return
- \n - Line Feed
- \b - Backspace
- \t - Tab
- \\ - Backslash

Codebase Revision: 0.5.0

Parameters: The text string (optionally including escape characters) to transmit

Output: None

```
# Send a URI with a new line ending to execute in Chrome, etc.
AT+BLEKEYBOARD=http://www.adafruit.com\r\n
OK
```

AT+BLEKEYBOARDCODE

Sends a raw hex sequence of characters to the BLE keyboard interface including key modifiers and up to six alpha-numeric characters.

This command excepts the following ascii-encoded HEX payload, matching the way HID over GATT sends keyboard data:

- **Byte 0**: Modifier
- **Byte 1**: Reserved (should always be 00)
- **Bytes 2..7**: Hexadecimal values for ASCII-encoded characters (if no character is used you can enter '00' or leave trailing characters empty)

After a keycode sequence is sent with the AT+BLEKEYBOARDCODE command, **you must send a second AT+BLEKEYBOARDCODE command with at least two 00 characters to indicate the keys were released!**

Modifier Values

The modifier byte can have one or more of the following bits set:

- **Bit 0 (0x01)**: Left Control
- **Bit 1 (0x02)**: Left Shift
- **Bit 2 (0x04)**: Left Alt
- **Bit 3 (0x08)**: Left Window
- **Bit 4 (0x10)**: Right Control
- **Bit 5 (0x20)**: Right Shift
- **Bit 6 (0x40)**: Right Alt
- **Bit 7 (0x80)**: Right Window

Codebase Revision: 0.5.0

Parameters: A set of ASCII-encoded hexadecimal characters values separated by a hyphen ('-')

Output: None

```
# send ABC with shift key --> "ABC"
AT+BLEKEYBOARDCODE=01-00-04-05-06-00-00
OK
# Indicate that the keys were released (mandatory!)
AT+BLEKEYBOARDCODE=00-00
OK
```

BLE GAP

GAP (http://adafru.it/eci), which stands for the *Generic Access Profile*, governs advertising and connections with Bluetooth Low Energy devices.

The following commands can be used to configure the GAP settings on the BLE module.

You can use these commands to modify the advertising data (for ex. the device name that appears during the advertising process), to retrieve information about the connection that has been established between two devices, or the disconnect if you no longer wish to maintain a connection.

AT+GAPGETCONN

Diplays the current connection status (if we are connected to another BLE device or not).

Codebase Revision: 0.3.0

Parameters: None

Output: 1 if we are connected, otherwise 0

```
# Connected
AT+GAPGETCONN
1
OK

# Not connected
AT+GAPGETCONN
0
OK
```

AT+GAPDISCONNECT

Disconnects to the external device if we are currently connected.

Codebase Revision: 0.3.0

Parameters: None

Output: None

```
AT+GAPDISCONNECT
OK
```

AT+GAPDEVNAME

Gets or sets the device name, which is included in the advertising payload for the Bluefruit LE module

Codebase Revision: 0.3.0

Parameters:

- None to read the current device name
- The new device name if you want to change the value

Output: The device name if the command is executed in read mode

> Updating the device name will persist the new value to non-volatile memory, and the updated name will be used when the device is reset. To reset the device to factory settings and clean the config data from memory run the AT+FACTORYRESET command.

```
# Read the current device name
AT+GAPDEVNAME
UART
OK

# Update the device name to 'BLEFriend'
AT+GAPDEVNAME=BLEFriend
OK
# Reset to take effect
ATZ
OK
```

AT+GAPDELBONDS

Deletes and bonding information stored on the Bluefruit LE module.

Codebase Revision: 0.3.0

Parameters: None

Output: None

```
AT+GAPDELBONDS
OK
```

AT+GAPINTERVALS

Gets or sets the various advertising and connection intervals for the Bluefruit LE module.

Be extremely careful with this command since it can be easy to cause problems changing the intervals, and depending on the values selected some mobile devices may no longer recognize the module or refuse to connect to it.

Codebase Revision: 0.3.0

Parameters: If updating the GAP intervals, the following comma-separated values can be entered:

- Minimum connection interval (in milliseconds)
- Maximum connection interval (in milliseconds)
- Advertising interval (in milliseconds)
- Advertising timeout (in milliseconds)

If you only wish to update one interval value, leave the other comma-separated values empty (ex. ',,110,' will only update the third value, advertising interval).

Output: If reading the current GAP interval settings, the following comma-separated information will be displayed:

- Minimum connection interval (in milliseconds)
- Maximum connection interval (in milliseconds)
- Advertising interval (in milliseconds)
- Advertising timeout (in milliseconds)

Updating the GAP intervals will persist the new values to non-volatile memory, and the updated values will be used when the device is reset. To reset the device to factory settings and clean the config data from memory run the AT+FACTORYRESET command.

```
# Read the current GAP intervals
AT+GAPINTERVALS
20,100,100,30

# Update all values
AT+GAPINTERVALS=20,200,200,30
OK

# Update only the advertising interval
AT+GAPINTERVALS=,,150,
OK
```

AT+GAPSTARTADV

Causes the Bluefruit LE module to start transmitting advertising packets if this isn't already the case (assuming we aren't already connected to an external device).

Codebase Revision: 0.3.0

Parameters: None

Output: None

```
# Command results when advertising data is not being sent
AT+GAPSTARTADV
OK

# Command results when we are already advertising
AT+GAPSTARTADV
ERROR

# Command results when we are connected to another device
AT+GAPSTARTADV
ERROR
```

AT+GAPSTOPADV

Stops advertising packets from being transmitted by the Bluefruit LE module.

Codebase Revision: 0.3.0

Parameters: None

Output: None

```
AT+GAPSTOPADV
OK
```

AT+GAPSETADVDATA

Sets the raw advertising data payload to the specified byte array (overriding the normal advertising data), following the guidelines in the Bluetooth 4.0 or 4.1 Core Specification (http://adafru.it/ddd).

In particular, **Core Specification Supplement (CSS) v4** contains the details on common advertising data fields like 'Flags' (Part A, Section 1.3) and the various Service UUID lists (Part A, Section 1.1). A list of all possible GAP Data Types is available on the Bluetooth SIG's Generic Access Profile (http://adafru.it/cYs) page.

The Advertising Data payload consists of Generic Access Profile (http://adafru.it/cYs) data that is inserted into the advertising packet in the following format: [U8:LEN] [U8:Data Type Value] [n:Value]

WARNING: This command requires a degree of knowledge about the low level details of the Bluetooth 4.0 or 4.1 Core Specification, and should only be used by expert users. Misuse of this command can easily cause your device to be undetectable by central devices in radio range.

WARNING: This command will override the normal advertising payload and may prevent some services from acting as expected.

To restore the advertising data to the normal default values use the AT+FACTORYRESET command.

For example, to insert the 'Flags' Data Type (Data Type Value 0x01), and set the value to 0x06/0b00000110 (BR/EDR Not Supported and LE General Discoverable Mode) we would use the following byte array:

```
02-01-06
```

- 0x02 indicates the number of bytes in the entry

- 0x01 is the 'Data Type Value (http://adafru.it/cYs)' and indicates that this is a '**Flag**'
- 0x06 (0b00000110) is the Flag value, and asserts the following fields (see Core Specification 4.0, Volume 3, Part C, 18.1):
    - **LE General Discoverable Mode** (i.e. anyone can discover this device)
    - **BR/EDR Not Supported** (i.e. this is a Bluetooth Low Energy only device)

If we also want to include two 16-bit service UUIDs in the advertising data (so that listening devices know that we support these services) we could append the following data to the byte array:

```
05-02-0D-18-0A-18
```

- 0x05 indicates that the number of bytes in the entry (5)
- 0x02 is the 'Data Type Value (http://adafru.it/cYs)' and indicates that this is an '**Incomplete List of 16-bit Service Class UUIDs**'
- 0x0D 0x18 is the first 16-bit UUID (which translates to **0x180D**, corresponding to the Heart Rate Service (http://adafru.it/ddB)).
- 0x0A 0x18 is another 16-bit UUID (which translates to **0x180A**, corresponding to the Device Information Service (http://adafru.it/ecj)).

Including the service UUIDs is important since some mobile applications will only work with devices that advertise a specific service UUID in the advertising packet. This is true for most apps from Nordic Semiconductors, for example.

Codebase Revision: 0.3.0

Parameters: The raw byte array that should be inserted into the advertising data section of the advertising packet, being careful to stay within the space limits defined by the Bluetooth Core Specification.

Response: None

```
# Advertise as Discoverable and BLE only with 16-bit UUIDs 0x180D and 0x180A
AT+GAPSETADVDATA=02-01-06-05-02-0d-18-0a-18
OK
```

The results of this command can be seen in the screenshot below, taken from a sniffer analyzing the advertising packets in Wireshark. The advertising data payload is higlighted in blue in the raw

byte array at the bottom of the image, and the packet analysis is in the upper section:

```
▽ Bluetooth Low Energy Link Layer
    Access Address: 0x8e89bed6
  ▷ Packet Header: 0x0f40 (PDU Type: ADV_IND, TxAdd=false, RxAdd=false)
    Advertising Address: e4:c6:c7:31:95:11 (e4:c6:c7:31:95:11)
  ▽ Advertising Data
    ▽ Flags
        Length: 2
        Type: Flags (0x01)
        000. .... = Reserved: 0x00
        ...0 .... = Simultaneous LE and BR/EDR to Same Device Capable (Host): false (0x00)
        .... 0... = Simultaneous LE and BR/EDR to Same Device Capable (Controller): false (0x00)
        .... .1.. = BR/EDR Not Supported: true (0x01)
        .... ..1. = LE General Discoverable Mode: true (0x01)
        .... ...0 = LE Limited Discoverable Mode: false (0x00)
    ▽ 16-bit Service Class UUIDs (incomplete)
        Length: 5
        Type: 16-bit Service Class UUIDs (incomplete) (0x02)
        UUID 16: Heart Rate (0x180d)
        UUID 16: Device Information (0x180a)
  ▷ CRC: 0x93b900

0000  00 06 22 01 8b 17 06 0a  01 26 2b 00 00 97 02 00   ..".....  .&+.....
0010  00 d6 be 89 8e 40 0f 11  95 31 c7 c6 e4 02 01 06   .....@..  .1......
0020  05 02 0d 18 0a 18 c9 9d  00                        ........  .
```

BLE GATT

GATT (http://adafru.it/ebg), which standards for the *Generic ATTribute Profile*, governs data organization and data exchanges between connected devices. One device (the peripheral) acts as a GATT Server, which stores data in *Attribute* records, and the second device in the connection (the central) acts as a GATT Client, requesting data from the server whenever necessary.

The following commands can be used to create custom GATT services and characteristics on the BLEFriend, which are used to store and exchange data.

Please note that any characteristics that you define here will automatically be saved to non-volatile FLASH config memory on the device and re-initialised the next time the device starts. **You need to perform a system reset before most of the commands below will take effect!**

If you want to clear any previous config value, enter the '**AT+FACTORYRESET**' command before working on a new peripheral configuration.

AT+GATTCLEAR

Clears any custom GATT services and characteristics that have been defined on the device.

Codebase Revision: 0.3.0

Parameters: None

Response: None

```
AT+GATTCLEAR
OK
```

AT+GATTADDSERVICE

Adds a new custom service definition to the device.

Codebase Revision: 0.3.0

Parameters: This command accepts a set of comma-separated key-value pairs that are used to define the service properties. The following key-value pairs can be used:

- **UUID**: The 16-bit UUID to use for this service. 16-bit values should be in hexadecimal format (0x1234).
- **UUID128**: The 128-bit UUID to use for this service. 128-bit values should be in the following format: 00-11-22-33-44-55-66-77-88-99-AA-BB-CC-DD-EE-FF

Response: The index value of the service in the custom GATT service lookup table. (It's important to keep track of these index values to work with the service later.)

Note: Key values are not case-sensitive

Only one UUID type can be entered for the service (either UUID or UUID128)

```
# Clear any previous custom services/characteristics
AT+GATTCLEAR
OK

# Add a battery service (UUID = 0x180F) to the peripheral
AT+GATTADDSERVICE=UUID=0x180F
1
OK

# Add a battery measurement characteristic (UUID = 0x2A19), notify enabled
AT+GATTADDCHAR=UUID=0x2A19,PROPERTIES=0x10,MIN_LEN=1,VALUE=100
1
OK
```

```
# Clear any previous custom services/characteristics
AT+GATTCLEAR
OK

# Add a custom service to the peripheral
AT+GATTADDSERVICE=UUID128=00-11-00-11-44-55-66-77-88-99-AA-BB-CC-DD-EE-FF
1
OK

# Add a custom characteristic to the above service (making sure that there
# is no conflict between the 16-bit UUID and bytes 3+4 of the 128-bit service UUID)
AT+GATTADDCHAR=UUID=0x0002,PROPERTIES=0x02,MIN_LEN=1,VALUE=100
1
OK
```

AT+GATTADDCHAR

Adds a custom characteristic to the last service that was added to the peripheral (via AT+GATTADDSERVICE).

AT+GATTADDCHAR must be run AFTER AT+GATTADDSERVICE, and will add the new characteristic to the last service definition that was added.

Codebase Revision: 0.3.0

Parameters: This command accepts a set of comma-separated key-value pairs that are used to define the characteristic properties. The following key-value pais can be used:

- **UUID**: The 16-bit UUID to use for the characteristic (which will be insert in the 3rd and 4th bytes of the parent services 128-bit UUID). This value should be entered in hexadecimal format (ex. 'UUID=0x1234'). This value must be unique, and should not conflict with bytes 3+4 of the parent service's 128-bit UUID.
- **PROPERTIES**: The 8-bit characteristic properties field, as defined by the Bluetooth SIG. The following values can be used:
  - 0x02 - Read
  - 0x04 - Write Without Response
  - 0x08 - Write
  - 0x10 - Notify
  - 0x20 - Indicate
- **MIN_LEN**: The minimum size of the values for this characteristic (in bytes, min = 1, max = 20, default = 1)
- **MAX_LEN**: The maximum size of the values for the characteristic (in bytes, min = 1, max = 20, default = 1)
- **VALUE**: The initial value to assign to this characteristic (within the limits of 'MIN_LEN' and 'MAX_LEN'). Value can be an integer ("-100", "27"), a hexadecimal value ("0xABCD"), a byte array ("aa-bb-cc-dd") or a string ("GATT!").

Response: The index value of the characteristic in the custom GATT characteristic lookup table. (It's important to keep track of these characteristic index values to work with the characteristic later.)

Note: Key values are not case-sensitive

Make sure that the 16-bit UUID is unique and does not conflict with bytes 3+4 of the 128-bit service UUID

```
# Clear any previous custom services/characteristics
AT+GATTCLEAR
OK

# Add a battery service (UUID = 0x180F) to the peripheral
AT+GATTADDSERVICE=UUID=0x180F
1
OK

# Add a battery measurement characteristic (UUID = 0x2A19), notify enabled
AT+GATTADDCHAR=UUID=0x2A19,PROPERTIES=0x10,MIN_LEN=1,VALUE=100
1
OK
```

```
# Clear any previous custom services/characteristics
AT+GATTCLEAR
OK

# Add a custom service to the peripheral
AT+GATTADDSERVICE=UUID128=00-11-00-11-44-55-66-77-88-99-AA-BB-CC-DD-EE-FF
1
OK

# Add a custom characteristic to the above service (making sure that there
# is no conflict between the 16-bit UUID and bytes 3+4 of the 128-bit service UUID)
AT+GATTADDCHAR=UUID=0x0002,PROPERTIES=0x02,MIN_LEN=1,VALUE=100
1
OK
```

AT+GATTCHAR

Gets or sets the value of the specified custom GATT characteristic (based on the index ID returned when the characteristic was added to the system via AT+GATTADDCHAR).

Codebase Revision: 0.3.0

Parameters: This function takes one or two comma-separated functions (one parameter = read, two parameters = write).

- The first parameter is the characteristic index value, as returned from the AT+GATTADDCHAR function. This parameter is always required, and if no second parameter is entered the current value of this characteristic will be returned.
- The second (optional) parameter is the new value to assign to this characteristic (within the

MIN_SIZE and MAX_SIZE limits defined when creating it).

Response: If the command is used in read mode (only the characteristic index is provided as a value), the response will display the current value of the characteristics. If the command is used in write mode (two comma-separated values are provided), the characteristics will be updated to use the provided value.

```
# Clear any previous custom services/characteristics
AT+GATTCLEAR
OK

# Add a battery service (UUID = 0x180F) to the peripheral
AT+GATTADDSERVICE=UUID=0x180F
1
OK

# Add a battery measurement characteristic (UUID = 0x2A19), notify enabled
AT+GATTADDCHAR=UUID=0x2A19,PROPERTIES=0x10,MIN_LEN=1,VALUE=100
1
OK

# Read the battery measurement characteristic (index ID = 1)
AT+GATTCHAR=1
0x64
OK

# Update the battery measurement characteristic to 32 (hex 0x20)
AT+GATTCHAR=1,32
OK

# Verify the previous write attempt
AT+GATTCHAR=1
0x20
OK
```

AT+GATTLIST

Lists all custom GATT services and characteristics that have been defined on the device.

Codebase Revision: 0.3.0

Parameters: None

Response: A list of all custom services and characteristics defined on the device.

```
# Clear any previous custom services/characteristics
AT+GATTCLEAR
OK

# Add a battery service (UUID = 0x180F) to the peripheral
AT+GATTADDSERVICE=UUID=0x180F
1
OK

# Add a battery measurement characteristic (UUID = 0x2A19), notify enabled
AT+GATTADDCHAR=UUID=0x2A19,PROPERTIES=0x10,MIN_LEN=1,VALUE=100
1
OK

# Add a custom service to the peripheral
AT+GATTADDSERVICE=UUID128=00-11-00-11-44-55-66-77-88-99-AA-BB-CC-DD-EE-FF
2
OK

# Add a custom characteristic to the above service (making sure that there
# is no conflict between the 16-bit UUID and bytes 3+4 of the 128-bit service UUID)
AT+GATTADDCHAR=UUID=0x0002,PROPERTIES=0x02,MIN_LEN=1,VALUE=100
2
OK

# Get a list of all custom GATT services and characteristics on the device
AT+GATTLIST
ID=01,UUID=0x180F
  ID=01,UUID=0x2A19,PROPERTIES=0x10,MIN_LEN=1,MAX_LEN=1,VALUE=0x64
ID=02,UUID=0x11, UUID128=00-11-00-11-44-55-66-77-88-99-AA-BB-CC-DD-EE-FF
  ID=02,UUID=0x02,PROPERTIES=0x02,MIN_LEN=1,MAX_LEN=1,VALUE=0x64
OK
```

Debug

The following debug commands are available on Bluefruit LE modules:

> Use these commands with care since they can easily lead to a HardFault error on the ARM core, which will cause the device to stop responding.

AT+DBGMEMRD

Displays the raw memory contents at the specified address.

Codebase Revision: 0.3.0

Parameters: The following comma-separated parameters can be used with this command:

- The starting address to read memory from (in hexadecimal form, with or without the leading '0x')
- The word size (can be 1, 2, 4 or 8)
- The number of words to read

Output: The raw memory contents in hexadecimal format using the specified length and word size (see examples below for details)

```
# Read 12 1-byte values starting at 0x10000009
AT+DBGMEMRD=0x10000009,1,12
FF FF FF FF FF FF FF 00 04 00 00 00
OK

# Try to read 2 4-byte values starting at 0x10000000
AT+DBGMEMRD=0x10000000,4,2
55AA55AA 55AA55AA
OK

# Try to read 2 4-byte values starting at 0x10000009
# This will fail because the Cortex M0 can't perform misaligned
# reads, and any non 8-bit values must start on an even address
AT+DBGMEMRD=0x10000009,4,2
MISALIGNED ACCESS
ERROR
```

AT+DBGNVMREAD

Displays the raw contents of the config data section of non-volatile memory

Codebase Revision: 0.3.0

Properties: None

Output: The raw config data from non-volatile memory

```
AT+DBGNVMRD
FE CA 38 05 00 03 00 00 01 12 01 00 55 41 52 54 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 0
OK
```

AT+DBGSTACKSIZE

Returns the current stack size, to help detect stack overflow or detect stack memory usage when optimising memory usage on the system.

Codebase Revision: 0.4.7

Parameters: None

Output: The current size of stack memory in bytes

```
AT+DBGSTACKSIZE
1032
OK
```

AT+DBGSTACKDUMP

Dumps the current stack contents. Unused sections of stack memory are filled with '0xCAFEFOOD' to help determine where stack usage stops.

This command is purely for debug and development purposes.

Codebase Revision: 0.4.7

Parameters: None

Output: The memory contents of the entire stack region

```
AT+DBGSTACKDUMP
```

```
AT+DBGSTACKDUMP
0x20003800: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003810: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003820: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003830: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003840: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003850: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003860: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003870: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003880: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003890: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x200038A0: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x200038B0: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x200038C0: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x200038D0: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x200038E0: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x200038F0: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003900: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003910: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003920: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003930: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003940: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003950: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003960: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003970: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003980: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003990: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x200039A0: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x200039B0: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x200039C0: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x200039D0: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x200039E0: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x200039F0: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003A00: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003A10: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003A20: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003A30: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003A40: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003A50: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003A60: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003A70: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003A80: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003A90: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003AA0: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003AB0: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003AC0: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003AD0: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
```

```
0x20003AE0: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003AF0: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003B00: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003B10: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003B20: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003B30: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003B40: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003B50: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003B60: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003B70: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003B80: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003B90: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003BA0: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003BB0: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003BC0: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003BD0: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003BE0: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003BF0: CAFEF00D CAFEF00D 00000000 CAFEF00D
0x20003C00: 00000004 20001D04 CAFEF00D FFFFEF68
0x20003C10: CAFEF00D 00001098 CAFEF00D CAFEF00D
0x20003C20: CAFEF00D CAFEF00D 00001006 200018D8
0x20003C30: 00000001 200018D8 20001C50 00000004
0x20003C40: 20001BB0 000134A5 0000100D 20001D28
0x20003C50: 00000006 00000006 20001C38 20001D44
0x20003C60: 20001C6C 20001D44 00000006 00000005
0x20003C70: 20001D38 00000005 20001D38 00000000
0x20003C80: 00000001 00012083 200018C8 000013D3
0x20003C90: 00550000 00000001 80E80000 4FC40000
0x20003CA0: 000080E8 00000009 60900000 000080E8
0x20003CB0: 60140000 20002764 0009608F 000080E8
0x20003CC0: 80000000 000080E8 00000000 00129F5F
0x20003CD0: 00000000 0001E4D9 80E80000 200018C8
0x20003CE0: 200018D4 00000000 80E80000 000000FF
0x20003CF0: 0000011C 0001BCE1 0000203A 0001BC1D
0x20003D00: 00000000 0001BC1D 80E80000 0001BCE1
0x20003D10: 0000011C 0001BDA9 80E80000 0001BDA9
0x20003D20: 0000011C FFFFFFF9 008B8000 0001BC1D
0x20003D30: 00000048 00000010 0000A000 00000009
0x20003D40: 0001E326 00000001 80E80000 51538000
0x20003D50: 000080E8 0001E9CF 00000000 00000009
0x20003D60: 61C78000 000080E8 00000048 00000504
0x20003D70: 0000A1FC 0002125C 00000000 000080E8
0x20003D80: 00000000 0012A236 00000000 0001E4D9
0x20003D90: 000080E8 00000009 00004998 000080E8
0x20003DA0: 622C8000 0012A29B 00000042 0001E479
0x20003DB0: 40011000 000185EF 00006E10 00000000
0x20003DC0: 00000000 00000004 0000000C 00000000
```

```
0x20003DD0: 62780000 00018579 2000311B 0001ACDF
0x20003DE0: 00000000 20003054 20002050 00000001
0x20003DF0: 20003DF8 0002085D 00000001 200030D4
0x20003E00: 00000200 0001F663 00000001 200030D4
0x20003E10: 00000001 2000311B 0001F631 00020A6D
0x20003E20: 00000001 00000000 0000000C 200030D4
0x20003E30: 2000311B 00000042 200030D4 00020AD7
0x20003E40: 20002050 200030D4 20002050 00020833
0x20003E50: 20002050 20003F1B 20002050 0001FF89
0x20003E60: 20002050 0001FFA3 00000005 20003ED8
0x20003E70: 20002050 0001FF8B 00000010 00020491
0x20003E80: 00000001 0012A54E 00000020 00022409
0x20003E90: 00000000 20002050 200030D4 0001FF8B
0x20003EA0: 00021263 00000005 0000000C 20003F74
0x20003EB0: 20003ED8 20002050 200030D4 00020187
0x20003EC0: 20003ED4 20003054 00000000 20003F75
0x20003ED0: 00000008 20003F64 00000084 FFFFFFFF
0x20003EE0: FFFFFFFF 00000008 00000001 00000008
0x20003EF0: 20302058 2000311B 0001F631 00020A6D
0x20003F00: 20002050 00000000 0000000C 200030D4
0x20003F10: 32002050 32303032 00323330 000258D7
0x20003F20: 20002050 200030D4 20002050 00020833
0x20003F30: 00000000 20002050 00000020 000001CE
0x20003F40: 20003F40 200030D4 00000004 0001ED83
0x20003F50: 200030D4 20003F60 000001D6 000001D7
0x20003F60: 000001D8 00016559 0000000C 00000000
0x20003F70: 6C383025 00000058 200030D4 FFFFFFFF
0x20003F80: 1FFF4000 00000028 00000028 000217F8
0x20003F90: 200020C7 000166C5 000166AD 00017ED9
0x20003FA0: FFFFFFFF 200020B8 2000306C 200030D4
0x20003FB0: 200020B4 000180AD 1FFF4000 200020B0
0x20003FC0: 200020B0 200020B0 1FFF4000 0001A63D
0x20003FD0: CAFEF00D CAFEF00D 200020B4 00000002
0x20003FE0: FFFFFFFF FFFFFFFF 1FFF4000 00000000
0x20003FF0: 00000000 00000000 00000000 00016113
OK
```

History

This page tracks additions or changes to the AT command set based on the firmware version number (which you can obtain via the 'ATI' command):

Version 0.6.5

The following AT commands were added in the 0.6.5 release:

- AT+BLEGETPEERADDR (http://adafru.it/fuU)

The following commands were changed in the 0.6.5 release:

- Increased the UART buffer size (on the nRF51) from 128 to 256 bytes
- +++ now responds with the current operating mode
- Fixed a bug with AT+GATTCHAR values sometimes not being saved to NVM
- Fixed a bug with AT+GATTCHAR max_len value not being taken into account after a reset (min_len was always used when repopulating the value)

Version 0.6.2

This is the first release targetting **32KB SRAM parts (QFAC)**. 16KB SRAM parts can't be used with this firmware due to memory management issues, and should use the earlier 0.5.0 firmware.

The following AT commands were changed in the 0.6.2 release:

- AT+BLEUARTTX (http://adafru.it/eBK)
  Basic escape codes were added for new lines, tabs and backspace
- AT+BLEKEYBOARD (http://adafru.it/eBK)
  Also works with OS X now, and *may* function with other operating systems that support BLE HID keyboards

Version 0.5.0

The following AT commands were added in the 0.5.0 release:

- AT+BLEKEYBOARDEN (http://adafru.it/eBK)
- AT+BLEKEYBOARD (http://adafru.it/eBK)
- AT+BLEKEYBOARDCODE (http://adafru.it/eBK)

The following AT commands were changed in the 0.5.0 release:

- ATI (http://adafru.it/ei1)
  The SoftDevice, SoftDevice version and bootloader version were added as a new (7th)

record. For Ex: "S110 7.1.0, 0.0" indicates version 7.1.0 of the S110 softdevice is used with the 0.0 bootloader (future boards will use a newer 0.1 bootloader).

Other notes concerning 0.5.0:

Starting with version 0.5.0, you can execute the **AT+FACTORYRESET** command at any point (and without a terminal emulator) by holding the DFU button down for 10 seconds until the blue CONNECTED LED starts flashing, then releasing it.

Version 0.4.7

The following AT commands were added in the 0.4.7 release:

- +++ (http://adafru.it/ei1)
- AT+HWRANDOM (http://adafru.it/ei2)
- AT+BLEURIBEACON (http://adafru.it/ei3)
- AT+DBGSTACKSIZE (http://adafru.it/ei4)
- AT+DBGSTACKDUMP (http://adafru.it/ei4)

The following commands were changed in the 0.4.7 release:

- ATI (http://adafru.it/ei1)The chip revision was added after the chip name. Whereas ATI would previously report 'nRF51822', it will now add the specific HW revision if it can be detected (ex 'nRF51822 QFAAG00')

Version 0.3.0

- First public release

Command Examples

The following code snippets can be used when operating in Command Mode to perform specific tasks.

Heart Rate Monitor Service

The command list below will add a Heart Rate (http://adafru.it/ddB) service to the BLEFriend's attribute table, with two characteristics:

- Heart Rate Measurement (http://adafru.it/ddD)
- Body Sensor Location (http://adafru.it/eck)

```
# Perform a factory reset to make sure we get a clean start
AT+FACTORYRESET
OK

# Add the Heart Rate service entry
AT+GATTADDSERVICE=UUID=0x180D
1
OK

# Add the Heart Rate Measurement characteristic
AT+GATTADDCHAR=UUID=0x2A37, PROPERTIES=0x10, MIN_LEN=2, MAX_LEN=3, VALUE=00-40
1
OK

# Add the Body Sensor Location characteristic
AT+GATTADDCHAR=UUID=0x2A38, PROPERTIES=0x02, MIN_LEN=1, VALUE=3
2
OK

# Create a custom advertising packet that includes the Heart Rate service UUID
AT+GAPSETADVDATA=02-01-06-05-02-0d-18-0a-18
OK

# Reset the device to start advertising with the custom payload
ATZ
OK

# Update the value of the heart rate measurement (set it to 0x004A)
AT+GATTCHAR=1,00-4A
OK
```

Python Script

The following code performs the same function, but has been placed inside a Python wrapper using PySerial (http://adafru.it/cLU) to show how you can script actions for the AT parser.

```
import io
import sys
import serial
import random
from time import sleep

filename = "hrm.py"
ser = None
serio = None
verbose = True  # Set this to True to see all of the incoming serial data


def usage():
    """Displays information on the command-line parameters for this script"""
    print "Usage: " + filename + " <serialPort>\n"
    print "For example:\n"
    print "  Windows : " + filename + " COM14"
    print "  OS X    : " + filename + " /dev/tty.usbserial-DN009WNO"
    print "  Linux   : " + filename + " /dev/ttyACM0"
    return


def checkargs():
    """Validates the command-line arguments for this script"""
    if len(sys.argv) < 2:
        print "ERROR: Missing serialPort"
        usage()
        sys.exit(-1)
    if len(sys.argv) > 2:
        print "ERROR: Too many arguments (expected 1)."
        usage()
        sys.exit(-2)


def errorhandler(err, exitonerror=True):
    """Display an error message and exit gracefully on "ERROR\r\n" responses"""
    print "ERROR: " + err.message
    if exitonerror:
        ser.close()
        sys.exit(-3)
```

```
def atcommand(command, delayms=0):
    """Executes the supplied AT command and waits for a valid response"""
    serio.write(unicode(command + "\n"))
    if delayms:
        sleep(delayms/1000)
    rx = None
    while rx != "OK\r\n" and rx != "ERROR\r\n":
        rx = serio.readline(2000)
        if verbose:
            print unicode(rx.rstrip("\r\n"))
    # Check the return value
    if rx == "ERROR\r\n":
        raise ValueError("AT Parser reported an error on '" + command.rstrip() + "")


if __name__ == '__main__':
    # Make sure we received a single argument (comPort)
    checkargs()

    # This will automatically open the serial port (no need for ser.open)
    ser = serial.Serial(port=sys.argv[1], baudrate=9600, rtscts=True)
    serio = io.TextIOWrapper(io.BufferedRWPair(ser, ser, 1),
                             newline='\r\n',
                             line_buffering=True)

    # Add the HRM service and characteristic definitions
    try:
        atcommand("AT+FACTORYRESET", 1000)  # Wait 1s for this to complete
        atcommand("AT+GATTCLEAR")
        atcommand("AT+GATTADDSERVICE=UUID=0x180D")
        atcommand("AT+GATTADDCHAR=UUID=0x2A37, PROPERTIES=0x10, MIN_LEN=2, MAX_LEN=3, VAI
        atcommand("AT+GATTADDCHAR=UUID=0x2A38, PROPERTIES=0x02, MIN_LEN=1, VALUE=3")
        atcommand("AT+GAPSETADVDATA=02-01-06-05-02-0d-18-0a-18")
        # Perform a system reset and wait 1s to come back online
        atcommand("ATZ", 1000)
        # Update the value every second
        while True:
            atcommand("AT+GATTCHAR=1,00-%02X" % random.randint(50, 100), 1000)
    except ValueError as err:
        # One of the commands above returned "ERROR\n"
        errorhandler(err)
    except KeyboardInterrupt:
        # Close gracefully on CTRL+C
        ser.close()
        sys.exit()
```

The results of this script can be seen below in the 'HRM' app of Nordic's nRF Toolbox application:

Please note that nRF Toolbox will only display HRM data if the value changes, so you will need to update the Heart Rate Measurement characteristic at least once after opening the HRM app and connecting to the BLEFriend

DISCONNECT
Wireless by Nordic

Field Updates

The BLEFriend module includes a special 'DFU' bootloader that allows you to update the firmware over the air using a compatible device (such as a recent iPhone or iPod or a BLE-enabled Android device running Android 4.3 or later).

The benefit of this bootloader is that you can reprogram the flash contents of the MCU without having to purchase a HW debugger, using nothing but your tablet or phone to update the flash memory contents of the BLEFriend.

You can update the Firmware over the air but not over the USB/UART connection!

Requirements

In order to perform over the air field updates of the firmware on the Bluefruit LE Pro modules, you will need access to the following resources:

- A Bluetooth Low Energy enabled iOS device (iPad 3rd generation or later, iPhone 5 or later, or a recent iPod Touch) or a BLE enabled Android device (running Android 4.3 or higher, for example the Nexus 4, Nexus 5, and the Nexus 7 2013)
- The nRF Toolbox application from Nordic Semiconductors for your platform of choice (available in the Apple (http://adafru.it/dd7) or Android (http://adafru.it/dd6) app stores).
- The latest Bluefruit LE Firmware image (http://adafru.it/edX) for you device (always available via Github).

While the DFU (Device Firmware Update) code running on the modules has several safety feature (CRC packet checks, caching and testing the entire firmware image before writing it, etc.) this procedure isn't without risk, however small. There is a small possibility you can brick your device in DFU mode, requiring access to a HW debugger like the J-Link to recover it.

Entering DFU Mode

The enter DFU mode, hold down the **DFU** button when inserting the BLEFriend into your USB port.

You will know that you are in DFU mode because the MODE LED will blink at a constant rate:

DFU Timeout

By default, the DFU code has a **5-minute timeout**. After 5 minutes in DFU mode, the device will revert back to the normal user code if any user code is present on the device. To return back to DFU mode, you will need to hold down the DFU button and reset the device via a power cycle.

Once you're in DFU mode, continue on to the iOS or Android pages ahead.

Firmware Images

The latest Bluefruit LE firmware images are always available in our Github repo at https://github.com/adafruit/Adafruit_BluefruitLE_Firmware (http://adafru.it/edX)

DFU on iOS

You can perform over the air field updates to the BLEFriend using most recent iOS devices since Bluetooth Low Energy support has been available from Apple for a couple years.

We used an **iPad Mini running iOS 7.1.2** for this tutorial, and the screenshots may vary slightly if you are using iOS7 or another version of Apple's mobile operating system.

Install nRF Toolbox

The first step to enable DFU support on your iOS device is to install an application named nRF Toolbox (http://adafru.it/e9L).

Note: When searching for 'nRF Toolbox' in the App Store you may need to specify that you are searching for an iPhone app since the default will be iPad apps.

iPad
12:48 am
64%
Annuler
Juste l'iPhone
Prix
Catégories
Pertinence
Tous les âges
nrf tool
nRF Toolbox
Nordic Semiconductor ASA
Aucune note
nRF Toolbox
Nordic Semiconductor ASA
Aucune note
Détails
Avis
Associés
DfuTarg
DEFAULT DFU
Firmware
Name: firmware.zip
Size: 115740 bytes
Type: softdevice
DEVICE FIRMWARE UPDATE
SELECT DEVICE
Description
The nRF Toolbox works with a wide range of the most popular Bluetooth Low Energy accessories.
Sélection
Classements
À proximité
Achats
Mises à jour

Using DFU in nRF Toolbox

Once you've installed the nRF Toolbox application and have it open you should see the following main menu:

iPad
12:48 am
64%
Bluetooth
SMART
BGM
BPM
CSC
DFU
DFU
HRM
HTM
PROXIMITY
RSC
Wireless by Nordic

Tap the **DFU** icon to load the DFU app, which is what we will be using to transfer new firmware to the BLEFriend over the air:

iPad
12:48 am
64%
Back
Bluetooth
SMART
DEFAULT DFU
DEVICE FIRMWARE UPDATE
Firmware
Name:
Size:
Type: Required
Select File
Select File Type
DEVICE FIRMWARE UPDATE
Upload
SELECT DEVICE

Adding Custom Firmware

Before we can transfer our custom firmware to the BLEFriend, we need to copy the .hex file over to iPad, which requires us to connect the iPad to our computer and load up iTunes.

With the iPad connected to our computer and iTunes open, click the 'iPad' icon in the top-right-hand corner:

Once the iPad has been selected, select the **Apps** tab at the top and scroll down a bit, and you should see a section called **File Sharing**, as well as the **nRF Toolbox** application, as shown below:

Select the **nRF Toolbox** icon on the left-hand side, and then click the **"Add..."** button which will give you a file selection dialogue box, as shown below:

Select the appropriate .**hex** file that contains your firmware (in this case **blemodule_s110_xxaa.hex**), and click the '**Add**' button, and the file should be added to the nRF Toolbox Documents list, and shown below:

Now sync the iPad with your computer using the **Sync** button in the bottom right-hand corder of iTunes, and the .hex file should be transferred to the nRF Toolbox app on your iOS device.

Transferring the Firmware via BLE

Now that you have the custom firmware image(s) on your iOS device, we can transfer one of them over the air.

Going back into the nRF Toolbox application on the iPad, open the DFU home page and click the **Select File** label shown in the image below:

iPad
12:48 am
64%
Back
Bluetooth
SMART
DEFAULT DFU
DEVICE FIRMWARE UPDATE
Firmware
Name:
Size:
Type: Required
Select File
Select File Type
DEVICE FIRMWARE UPDATE
Upload
SELECT DEVICE

This will bring up the following file selection dialogue box by default, listing some pre-installed firmware images from Nordic Semiconductors:

iPad
12:49 am
63%
Cancel
Bluetooth
SMART
ble_app_hrs_s110_v5_0_0...
ble_app_hrs_s110_v6_0_0...
ble_app_hrs_s110_v7_0_0...
ble_app_rscs_s110_v5_0_...
ble_app_rscs_s110_v6_0_...
ble_app_rscs_s110_v7_0_...
Wireless by Nordic
App Files
User Files

Click the **User Files** icon in the image above, which should show us any custom firmware files we have transferred via iTunes, as decribed earlier in this page:

> The filename used here may be different than the production firmware images or future updates provided by Adafruit.

iPad
12:24 pm
Cancel
Bluetooth
SMART
Edit
blemodule_s110_xxaa....
Wireless by Nordic
App Files
User Files

Select the appropriate firmware image by clicking on it, which will bring you back to the DFU homepage, but some basic information about the .hex file will be displayed for us:

iPad
12:25 pm
Back
Bluetooth
SMART
DEFAULT DFU
DEVICE FIRMWARE UPDATE
Firmware
Name: blemodule_s110_xxaa...
Size: 140338 bytes
Type: Required
Select File
Select File Type
DEVICE FIRMWARE UPDATE
Upload
SELECT DEVICE

Take note of the field that says '**Type: Required**'. This means that we haven't specified what type of code is encoded in the .hex file (user code, an update to the SoftDevice or an update to the DFU bootloader, all of which can be updated over the air).

Next click the **Select File Type** label, and indicate that this is **application** code, as shown below:

Be very careful to only select APPLICATION in this dialogue or you can corrupt the BLEFriends flash memory contents!

iPad
12:50 am
63%
Back
Bluetooth
SMART
softdevice
bootloader
application
softdevice and bootloader
Wireless by Nordic

This will send us back to the DFU home page where the **Type** field will have been updated to **application**, and we're almost ready to transfer our firmware image.

Put the BLEFriend in DFU Mode

Next we need to put the BLEFriend in DFU mode, which we can do by pressing and holding down the DFU button while inserting the device, being sure to keep the DFU button pressed for 2-3 seconds after inserting the device.

If everything worked properly you should see an LED blinking quickly at a constant rate with no irregular pauses between pulses, indicating that we are in DFU mode:

Connecting to the DFU Service

With the BLEFriend in DFU mode, we can now connect to it.

Click the **Select Device** button at the bottom of the DFU application on the iPad, which should bring up the following dialogue box:

> NOTE: Because of the way iOS caches device data, the device may be advertised as UART or whatever custom name you set up. If that is the case you need to clear the cache by enabling and then disabling Bluetooth on your iOS device. This is unfortunately something outside the application level, and we can't reset Bluetooth inside an app ourselves.

iPad
12:49 am
63%
Select device:
No name
DfuTarg
Cancel

If the BLEFriend module is in DFU mode, it will advertise itself as **DfuTarg**. Select this target by clicking on the DfuTarg entry in the listview.

This should enable the **Upload** label, as shown below:

iPad
1:07 pm
51%
Back
Bluetooth
SMART
DfuTarg
DEVICE FIRMWARE UPDATE
Firmware
Name: blemodule_s110_xxaa...
Size: 140338 bytes
Type: application
Select File
Select File Type
DEVICE FIRMWARE UPDATE
Upload
SELECT DEVICE

Click the **Upload** label to start transferring the file over the air, and you will be updated on the file progress via a simple progress bar:

NOTE: If you encounter any problems with DfuTarg disconnecting or the Upload label not being enabled, trying enabling and then disabling Bluetooth on your device, which will reset the Bluetooth stack.

iPad
1:08 pm
51%
Back
Bluetooth
SMART
DfuTarg
DEVICE FIRMWARE UPDATE
Firmware
Name: blemodule_s110_xxaa...
Size: 140338 bytes
Type: application
Select File
Select File Type
DEVICE FIRMWARE UPDATE
Cancel
uploading application ...
19 %
SELECT DEVICE

Once the upload process is complete, you should see the following dialogue box, saying that the transfer is finished, and you can reset your device to start executing the new firmware (the blinky rate should change to show you are no longer in DFU mode after the reset):

The filesize shown in this dialogue is the actually size of the firmware, whereas the .hex file size is much larger due to the file encoding and extra information contained in the Intel .hex file.

iPad
1:08 pm
51%
Back
Bluetooth
SMART
DEFAULT DFU
DEVICE FIRMWARE UPDATE
Firmware
Name: blemodule_s110_xxaa...
DFU
45772 bytes transfered in 26 seconds
OK
SELECT DEVICE

Power cycle the BLEfriend by unplugging/replugging just to be sure, you're now updated!

DFU on Android (4.3+)

You can perform a field update of the BLEFriend's firmware over the air on Android devices, though you will need a recent version of Android to do this since Bluetooth Low Energy support was only added to version 4.3 (with some important stability updates in 4.4).

Verified Devices

We've tested DFU updates with the following devices:

- Nexus 4 (Android 4.4.4)
- Nexus 7 - 2013 (Android 4.4.4)

Note: The 2012 edition of the Nexus 7 won't work since it doesn't support BLE.

These screenshots all come from a **Nexus 7 running Android 4.4.4**, and the actual screen content may vary slightly depending on the device you are using and the version of Android you are running.

Download nRF Toolbox

The first thing you'll need to do is download the nRF Toolbox app for Android 4.3+ (http://adafru.it/e9M) from Nordic Semiconductors. It's available in Google's Play Store, making it relatively easy to install and keep up to date.

Load nRF Toolbox

Next, load the application on your Bluetooth Low Energy enabled Android 4.3 or higher device, where you should see a home page resembling the screenshot below:

BGM
BPM
CSC
DFU
HRM
HTM
PROXIMITY
RSC
Wireless by Nordic

Start the DFU Utility

Once the app has been opened, click the **DFU** icon to start the field update utility, which should display a page like this:

Select the File Type and Hex File

Select the .hex file that you want to program over the air by clicking the **SELECT FILE** button.

This will bring up a selection dialogue box asking what kind of code is contained inside the .hex file. In most cases, this will be an **Application** file:

> Make sure you select Application type, if you try to overwrite the bootloader you can corrupt the DFU system requiring a JTAG reprogramming!

Once the file type has been specified, you will be presented with a file selection dialogue, allowing you to navigate to your .hex file.

Récents
Drive
microbuilder.eu@gmail.com
Téléchargements
FSNavigator
Dropbox
ucun élément

Navigate to your **.hex file**, and select it using the appropriate popup dialogue box.

HINT: If you use Dropbox and have the Android Dropbox app (http://adafru.it/e9N) installed on your Android tablet, you can conveniently sync the latest .hex files by placing code inside a Dropbox folder.

Once the file is selected, you should be redirected back to the DFU home page, where the file details will have been updates, as shown below:

Put the BLEFriend in DFU Mode

Next we need to put the BLEFriend in DFU mode, which we can do by pressing and holding down the DFU button while inserting the device, being sure to keep the DFU button pressed for 2-3 seconds after inserting the device.

If everything worked properly you should see an LED blinking quickly at a constant rate with no irregular pauses between pulses, indicating that we are in DFU mode:

Select the Target DFU Device

Now that you've selected the application .hex file that you want to transfer over the air and put hte BLEFriend in DFU mode, you'll need to tell the DFU app which DFU-enabled device you want to transfer the image to.

You can do this by clicking the **SELECT DEVICE** button, making sure that your BLE device is in DFU mode (the LED should be blinky at a very high rate).

The Select Device dialogue will show you a list of all BLE devices in range, and you need to find a device labelled **DfuTarg**, which is our device advertising itself in DFU mode:

Select the **DfuTarg** device, and the dialogue box will close.

> If you see UART in the device list and you are sure that you are running in DFU mode (constant MODE LED blinking), you may need to enable and then disable Bluetooth on the Android device and try again). This will clear the device cache on Android and reset the BLE stack.

Transfer the Firmware

The final step is to start transferring the firmware over the air. Simply click the **UPLOAD** button, and you should see a screen update saying 'Uploading...', as shown below:

You'll see a progress indicator showing the over the air transfer (it can take up to 60 seconds to transfer a large firmware image):

01:17
DFU
DFUTARG
APPLICATION
File Name: blemodule_s110_xxaa.hex
File Type: Application
File Size: 117425 bytes
Status: OK
SELECT FILE
?
DEVICE FIRMWARE UPDATE
CANCEL
Uploading...
70%
DEVICE FIRMWARE UPDATE

When the transfer is complete you will get a small alert at the bottom of the application, the BLE module will reset and the new firmware should start executing automatically:

DEVIC
Application has been transferred successfully.
SELECT DEVICE
Wireless by Nordic

BLE Sniffer

Using a special firmware image provided by Nordic Semiconductors and the open source network analysis tool Wireshark, the BLEFriend can be converted into a low cost Bluetooth Low Energy sniffer.

> 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.

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:

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 periperhals 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/eci).

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/ekD) (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.

GATT Service Details

Data in Bluetooth Low Energy is organized around units called 'GATT Services (http://adafru.it/ebg)' and 'GATT Characteristics'. To expose data to another device, you must instantiate at least one service on your device.

Adafruit's Bluefruit LE Pro modules support some 'standard' services, described below (more may be added in the future).

UART Service

The UART Service is the standard means of sending and receiving data between connected devices, and simulates a familiar two-line UART interface (one line to transmit data, another to receive it).

The service is described in detail on the dedicated UART Service (http://adafru.it/ekD) page.

UART Service

Base UUID: 6E40**0001**-B5A3-F393- E0A9- E50E24DCCA9E

This service simulates a basic UART connection over two lines, TXD and RXD.

It is based on a proprietary UART service specification by Nordic Semiconductors. Data sent to and from this service can be viewed using the nRFUART apps from Nordic Semiconductors for Android and iOS.

This service is available on every Bluefruit LE Pro module and is automatically started during the power-up sequence.

Characteristics

Nordic's UART Service includes the following characteristics:

Name	Mandatory	UUID	Type	R	W	N	I
TX	Yes	0x0002	U8[20]	X		X	
RX	Yes	0x0003	U8[20]		X		

R = Read; W = Write; N = Notify; I = Indicate

TX (0x0002)

This characteristic is used to send data out to the connected Central device. Notify can be enabled by the connected device so that an alert is raised every time the TX channel is updated.

RX (0x0003)

This characteristic is used to send data back to the sensor node, and can be written to by the connected Central device (the mobile phone, tablet, etc.).