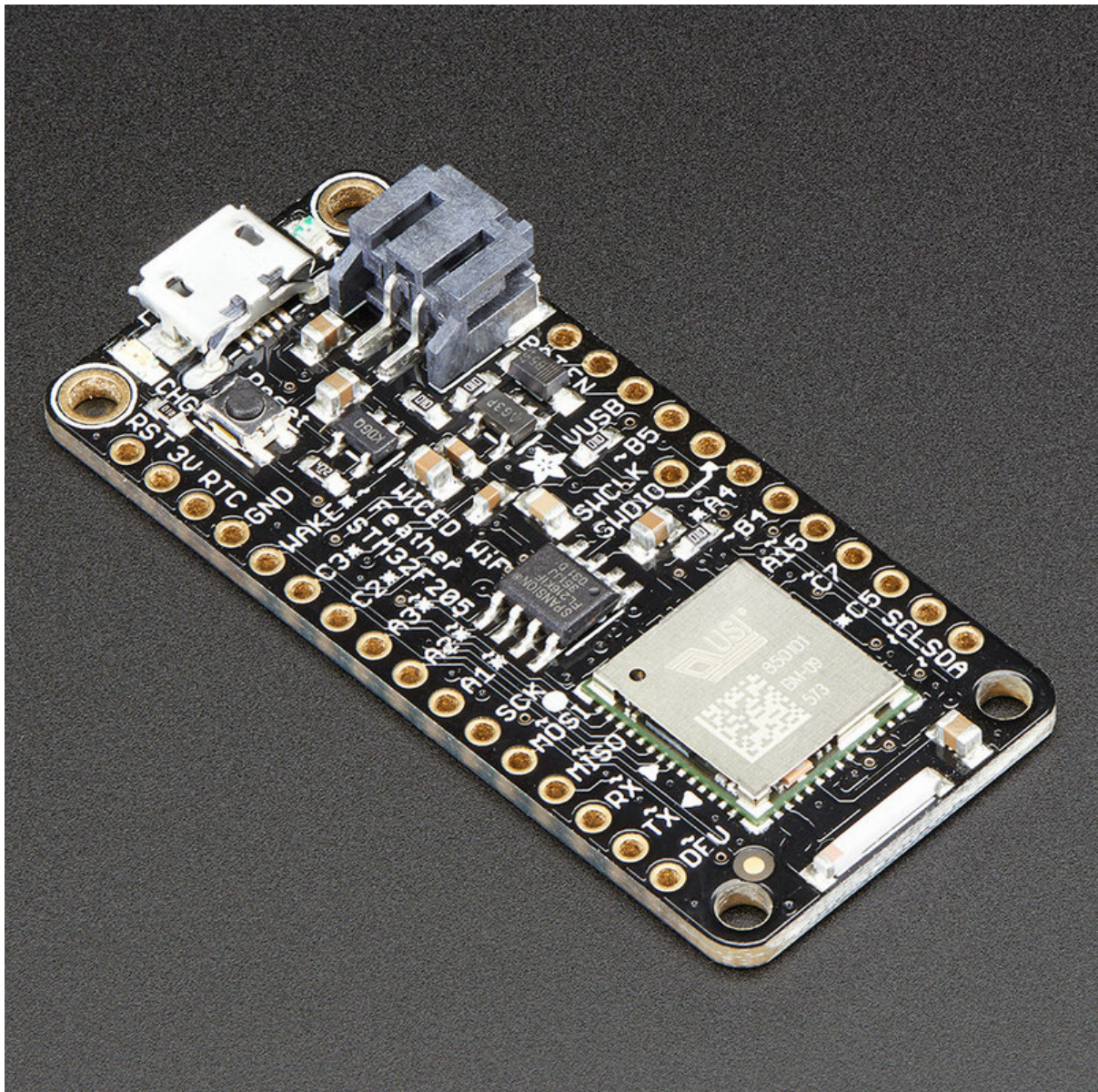




Introducing the Adafruit WICED Feather WiFi

Created by lady ada



<https://learn.adafruit.com/introducing-the-adafruit-wiced-feather-wifi>

Last updated on 2025-09-23 05:01:27 PM EDT

Table of Contents

Overview	11
Board Layout	14
• Pin Multiplexing • Accessing Pins in Software • Power Config • 16 Mbit (2MByte) SPI Flash • PWM Outputs	
Assembly	20
• Header Options! • Soldering in Plain Headers • Prepare the header strip: • Add the breakout board: • And Solder! • Soldering on Female Header • Tape In Place • Flip & Tack Solder • And Solder!	
Get the WICED BSP	30
• Adding Adafruit Board Support • Add the Adafruit BSP List • Add the Adafruit WICED BSP • Upgrading From Earlier WICED BSP Releases (<0.6.0)	
Windows Setup	33
• Install Adafruit Windows Drivers • Install libusb 0.1 Runtime • Install Python 2.7 • Testing the Python Installation • Install Python Tools • Optional: Install AdaLink • Setup Problems	
OS X Setup	39
• Install dfu-util • Install Python Tools • Optional: Install AdaLink	
Linux Setup	42
• UDEV Setup • Install dfu-util • Install Python Tools (BSP <= 0.6.2) • Optional: Install AdaLink • External Resources	
Arduino IDE Setup	46
• Board Selection • Setting the 'Section' • Selecting the Serial Port • Optional: Updating the Bootloader	

- [Compiling your Sketch](#)

System Architecture

53

- [WICED WiFi + RTOS + SDEP = FeatherLib](#)
- [Arduino User Code](#)
- [Inter Process Communication \(SDEP\)](#)
- [Flash Memory Layout](#)
- [User Code \(256KB + 20KB SRAM\)](#)
- [Feather Lib \(704 KB + 108KB SRAM\)](#)
- [Config Data \(32KB\)](#)
- [USB DFU Bootloader \(32KB\)](#)
- [USB Setup](#)
- [DFU Mode \(Fast Blinky\)](#)
- [Normal Operating Mode \(User Code\)](#)
- [Flash Updates](#)

WICED Feather API

58

- [AdafruitFeather](#)
- [AdafruitTCP](#)
- [AdafruitUDP](#)
- [AdafruitHTTP](#)
- [AdafruitMQTT](#)
- [AdafruitAIO](#)
- [AdafruitSDEP](#)
- [Client API](#)

AdafruitFeather

60

- [AdafruitFeather API](#)
- [Firmware Version Management](#)
- [char const* bootloaderVersion \(void \)](#)
- [char const* sdkVersion \(void \)](#)
- [char const* firmwareVersion \(void \)](#)
- [char const* arduinoVersion \(void \)](#)
- [Scanning](#)
- [int scanNetworks \(wl_ap_info_t ap_list\[\], uint8_t max_ap\)](#)
- [Connecting](#)
- [bool connect \(void \)](#)
- [bool connect \(const char *ssid \)](#)
- [bool connect \(const char *ssid, const char *key, int enc_type = ENC_TYPE_AUTO \)](#)
- [bool begin \(void \)](#)
- [bool begin \(const char *ssid \)](#)
- [bool begin \(const char *ssid, const char *key, int enc_type = ENC_TYPE_AUTO \)](#)
- [void disconnect \(void\)](#)
- [Network and Connection Details](#)
- [bool connected \(void \);](#)
- [uint8_t* macAddress \(uint8_t *mac \);](#)
- [uint32_t localIP \(void \);](#)
- [uint32_t subnetMask \(void \);](#)
- [uint32_t gatewayIP \(void \);](#)
- [char* SSID \(void \);](#)
- [int32_t RSSI \(void \);](#)
- [int32_t encryptionType \(void \);](#)
- [uint8_t* BSSID \(uint8_t* bssid \);](#)
- [DNS Lookup](#)
- [IPAddress hostByName \(const char* hostname \)](#)
- [bool hostByName \(const char* hostname, IPAddress& result \)](#)

- `bool hostByName ( const String &hostname, IPAddress& result )`
- Ping
- `uint32_t ping (char const* host)`
- `uint32_t ping (IPAddress ip)`
- Factory Reset
- `void factoryReset ( void )`
- `void nvmReset ( void )`
- Hardware Random Number Generator
- `bool randomNumber ( uint32_t* random32bit )`
- Real Time Clock
- `bool getISO8601Time ( iso8601_time_t* iso8601_time )`
- `uint32_t getUtcTime ( void )`
- TLS Root Certificate Management
- `bool useDefaultRootCA ( bool enabled )`
- `bool initRootCA (void)`
- `bool addRootCA (uint8_t const* root_ca, uint16_t len)`
- `bool clearRootCA ( void )`
- Print Helpers
- `void printVersions (Print& p = Serial)`
- `void printNetwork (Print& p = Serial)`
- `void printEncryption (int32_t enc, Print& p = Serial)`

AdafruitFeather: Profiles

73

- Connecting via Profiles
- Profiles API
- `bool saveConnectedProfile (void)`
- `bool addProfile (char* ssid)`
- `bool addProfile (char* ssid, char* key, wl_enc_type_t enc_type)`
- `bool removeProfile (char* ssid)`
- `void clearProfiles (void)`
- `char* profileSSID (uint8_t pos);`
- `int32_t profileEncryptionType (uint8_t pos);`

AdafruitTCP

77

- TCP Socket API
- Packet Buffering
- `void usePacketBuffering (bool enable)`
- TLS/SSL Certificate Verification
- Verifying Certificates with the WICED Feather (Safer)
- Ignoring Certificate Verification (Easier)
- `void tlsRequireVerification (bool required)`
- Socket Handler Functions
- `void getHandle (void)`
- Client API
- `int connect (IPAddress ip, uint16_t port)`
- `int connect (const char * host, uint16_t port)`
- `int connectSSL (IPAddress ip, uint16_t port)`
- `int connectSSL (const char* host, uint16_t port)`
- `uint8_t connected (void)`
- `void stop (void)`
- Stream API
- `int read (void)`
- `int read (uint8_t * buf, size_t size)`
- `size_t write (uint8_t data)`
- `size_t write (const uint8_t *content, size_t len)`
- `int available (void)`

- int peek (void)
- void flush (void)
- Callback API
- void setReceivedCallback (tcpcallback_t fp)
- void setDisconnectCallback (tcpcallback_t fp)
- Callback Function Signatures
- Example: Callback Based HTTP Request

AdafruitTCPServer

88

- Constructor
- Functions
- bool begin (void)
- AdafruitTCP accept (void)
- AdafruitTCP available (void)
- void stop (void)
- void setConnectCallback (tcpserver_callback_t fp)
- Example

AdafruitUDP

92

- UDP Socket API
- UDP API
- uint8_t begin (uint16_t port)
- void stop (void)
- int beginPacket (IPAddress ip, uint16_t port)
- int beginPacket (const char *host, uint16_t port)
- int endPacket (void)
- int parsePacket (void)
- IPAddress remoteIP (void)
- uint16_t remotePort (void)
- Stream API
- int read (void)
- int read (unsigned char* buffer, size_t len)int read (char* buffer, size_t len)
- int peek (void)
- int available (void)
- void flush (void)
- size_t write (uint8_t byte)
- size_t write (const uint8_t *buffer, size_t size)
- Callback Handlers
- void setReceivedCallback (udpcallback_t fp)
- Examples
- UDP Echo Server

AdafruitHTTP

101

- AdafruitHTTP API
- HTTP Headers
- bool addHeader (const char* name, const char* value)
- bool clearHeaders (void)
- HTTP GET Requests
- bool get (char const* url)
- bool get (char const* host, char const* url)
- HTTP POST Requests
- bool post (char const* url, char const* encoded_data)
- bool post (char const* host, char const* url, char const* encoded_data)
- HTTP GET Example

AdafruitHTTPServer

106

- AdafruitHTTPServer API
- Constructor
- Adding Pages
- Starting/Stopping the HTTP Server
- Complete Example

AdafruitMQTT

114

- Constructors
- Functions
- Connection Management
- `bool connected(void)`
- `bool connect ( IPAddress ip, uint16_t port = 1883, bool cleanSession = true, uint16_t keepalive_sec = MQTT_KEEPALIVE_DEFAULT);`
- `bool connect ( const char* host, uint16_t port = 1883, bool cleanSession = true, uint16_t keepalive_sec = MQTT_KEEPALIVE_DEFAULT);`
- `bool connectSSL ( IPAddress ip, uint16_t port = 8883, bool cleanSession = true, uint16_t keepalive_sec = MQTT_KEEPALIVE_DEFAULT)`
- `bool connectSSL ( const char* host, uint16_t port = 8883, bool cleanSession = true, uint16_t keepalive_sec = MQTT_KEEPALIVE_DEFAULT)`
- `bool disconnect (void)`
- Messaging
- `bool publish ( UTF8String topic, UTF8String message, uint8_t qos = MQTT_QOS_AT_MOST_ONCE, bool retained = false );`
- `bool subscribe ( const char* topicFilter, uint8_t qos, messageHandler mh);`
- `bool unsubscribe( const char* topicFilter );`
- Last Will
- `void will ( const char* topic, UTF8String message, uint8_t qos = MQTT_QOS_AT_MOST_ONCE, uint8_t retained = 0);`
- Client ID
- `void clientID(const char* client)`
- Disconnect Callback
- AdafruitMQTT Example

AdafruitMQTTTopic

128

- Constructor
- Functions
- `void retain (bool on)`
- Subscribe Callbacks
- `bool subscribe (messageHandler_t mh)`
- `bool unsubscribe (void)`
- `bool subscribed (void)`
- Publishing Data via 'Print'
- Example

AdafruitAIO

135

- Constructor
- Functions
- Connecting
- `bool connect (bool cleanSession = true, uint16_t keepalive_sec = MQTT_KEEPALIVE_DEFAULT)`
- `bool connectSSL (bool cleanSession = true, uint16_t keepalive_sec = MQTT_KEEPALIVE_DEFAULT)`
- Feed Management
- `bool updateFeed ( const char* feed, UTF8String message, uint8_t qos=MQTT_QOS_AT_MOST_ONCE, bool retain=true )`
- `bool followFeed ( const char* feed, uint8_t qos, messageHandler_t mh )`
- `bool unfollowFeed ( const char* feed )`

- [Example](#)

[AdafruitAIOFeed](#) 142

- [Constructor](#)
- [Functions](#)
- [bool follow \(feedHandler_t fp\)](#)
- [bool unfollow \(void\)](#)
- [bool followed \(void\)](#)
- [Example](#)

[AdafruitTwitter](#) 147

- [1. Creating a WICED Twitter Application](#)
- [Enter the Application Details](#)
- [Set the Application Permissions](#)
- [Manage the Access Keys](#)
- [Copy the Appropriate Key Data](#)
- [Create your Access Token](#)
- [2. Using the AdafruitTwitter Class](#)

[AdafruitSDEP](#) 152

- [AdafruitSDEP API](#)
- [Constructor](#)
- [Functions](#)
- [sdep](#)
- [sdep_n](#)
- [Error Handling Functions](#)
- [err_t errno \(void\)](#)
- [char const* errstr\(void\)](#)
- [char const* cmdstr \(uint16_t cmd_id\)](#)
- [void err_actions \(bool print, bool halt\)](#)
- [Error Handling Example](#)

[Client](#) 159

- [Adapting Client Examples](#)
- [1. Update Header Includes](#)
- [2. Change 'WiFi.*' References to 'Feather.*'](#)
- [3. Change WiFiUDP and WiFiTCP Class Types](#)

[Constants](#) 161

- [wl_enc_type_t](#)
- [err_t](#)
- [wl_ap_info_t](#)

[Python Tools](#) 164

- [pyresource.py \(Convert static files to C headers\)](#)
- [pycert.py \(Python TLS Certificate Converter\)](#)
- [feather_dfu.py \(Python USB DFU Utility\)](#)

[pyresource.py](#) 166

- [Usage](#)
- [HTTPResource Records](#)
- [HTTPResource Collection: resources.h](#)

[pycert.py](#) 168

- [Downloading the Root Certificate for a Domain](#)
- [Parameters](#)

- Usage
- Converting PEM Files
- Parameters
- Usage

feather_dfu.py 170

- Commands
- arduino_upgrade
- featherlib_upgrade
- enter_dfu
- info
- factory_reset
- nvm_reset
- reboot

SDEP Commands 173

Generic 174

- Reset (0x0001)
- Factory Reset (0x0002)
- Enter DFU Mode (0x0003)
- System Information (0x0004)
- NVM Reset (0x0005)
- Error String (0x0006)
- Generate Random Number (0x0101)

Examples 178

- Accessing the Examples (Arduino 1.6.5)
- Accessing the Examples (Arduino >= 1.6.8)
- Example Folders
- Making Modifications to the Examples

ScanNetworks 180

- Setup
- Compile and Flash
- Testing the Sketch

Ping 181

- Setup
- Compile and Flash
- Testing the Sketch

GetHostByName 183

- Setup
- Compile and Flash
- Testing the Sketch

HttpGetPolling 184

- Setup
- Compile and Flash
- Testing the Sketch

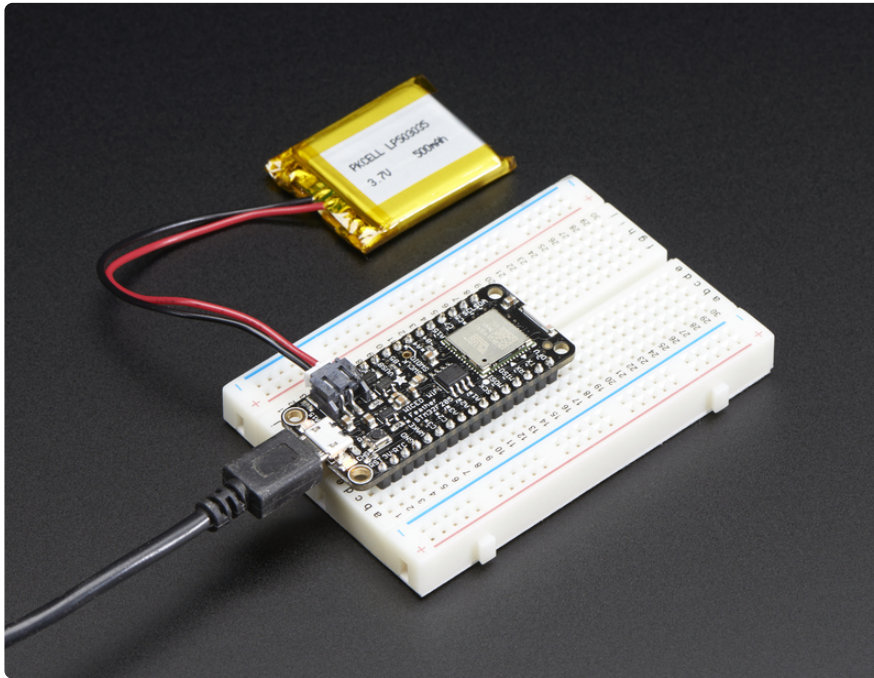
HttpGetCallback 186

- Setup
- Compile and Flash
- Testing the Sketch

HTTPSLargeData	188
• Setup • Compile and Flash • Testing the Sketch	
Throughput	190
• Setup • Running Netcat • Compile and Flash • Testing the Sketch	
FeatherOLED	193
• Setup • Compile and Flash • Testing the Sketch	
FAQs	196
Downloads	203
• Related Documents • Schematic • Fabrication Print	

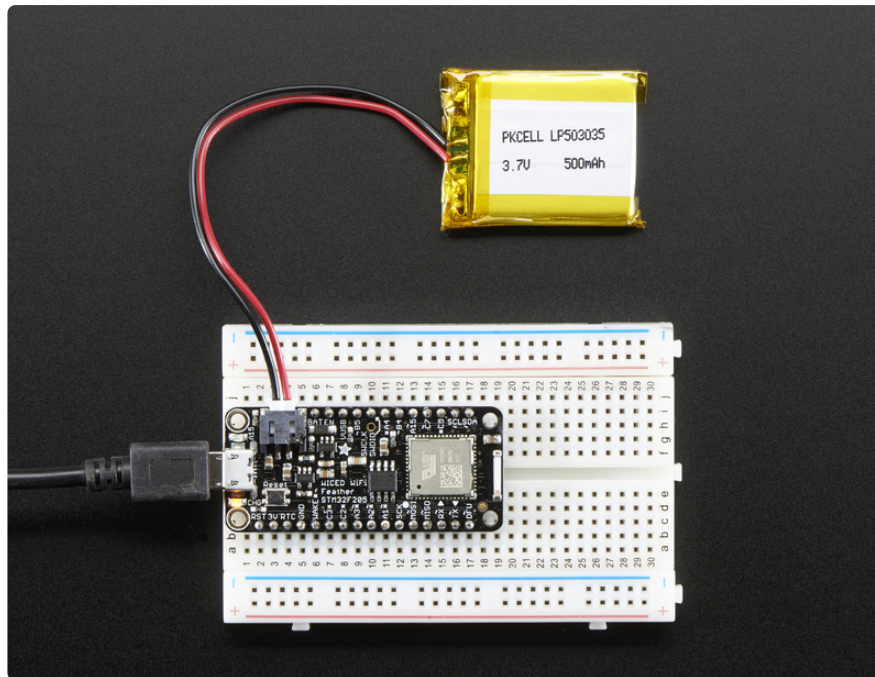
Overview

This guide is for an Adafruit product that has been discontinued. It is left as a reference only. It may contain code or libraries that no longer work.



[Feather \(https://adafru.it/I7B\)](https://adafru.it/I7B) is the new development board from Adafruit, and like its namesake it is thin, light, and lets you fly! We designed Feather to be a new standard for portable microcontroller cores. This is the **Adafruit WICED Feather** - it's our most powerful Feather yet! [We have other boards in the Feather family, check'em out here. \(https://adafru.it/I7B\)](https://adafru.it/I7B)

Say "Hi!" the WICED Feather! Perfect for your next Internet connected project, with a processor and WiFi core that can take anything you throw at it!

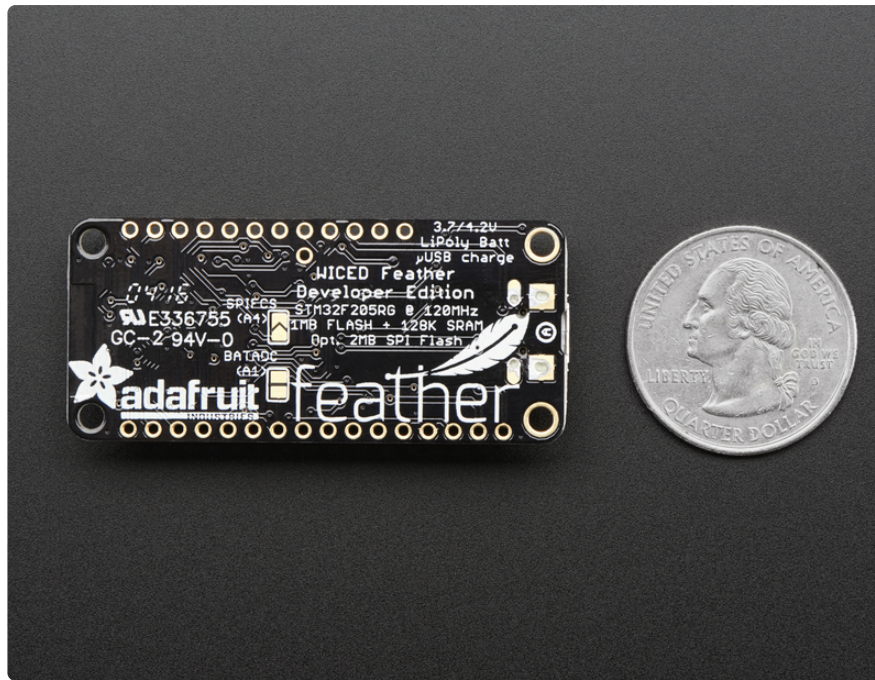


The WICED Feather is based on Broadcom's WICED (Wireless Internet Connectivity for Embedded Devices) platform, and is paired up with a powerful STM32F205 ARM Cortex M3 processor running at 120MHz, with support for TLS 1.2 to access sites and web services safely and securely.

We spent a lot of time adding support for this processor and WiFi chipset to the Arduino IDE you know and love. Programming doesn't rely on any online or third party tools to build, flash or run your code. You write your code in the Arduino IDE using many of the same standard libraries you've always used (Wire, SPI, etc.), compile locally, and the device is flashed directly from the IDE over USB. **Note that this chipset is not identical to the Arduino standard-supported Atmega series and many libraries that are written for AVR will not compile or work with WICED!**

Since the WICED Feather is based on the standard [Adafruit Feather \(https://adafru.it/mf2\)](https://adafru.it/mf2) layout, you also have instant access to a variety of FeatherWings, as well as all the usual standard breakouts available from Adafruit or other vendors.

After more than a year of full time effort in the making, we think it's the best and most flexible WiFi development board out there, and the easiest way to get your TCP/IP-based project off the ground without sacrificing flexibility or security. We even cooked in some built-in libraries in the WiFi core, such as TCP client and Server, HTTP client and server, and MQTT client (with easy Adafruit IO interfacing).



The WICED Feather has the following key features:

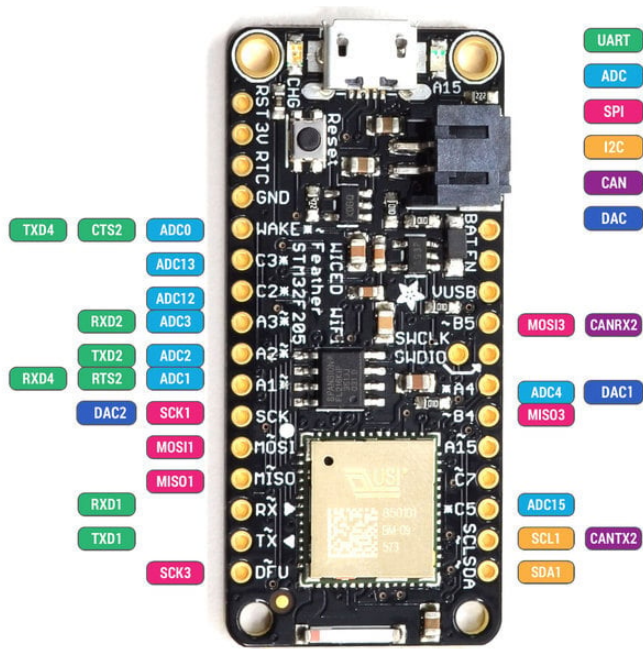
- Measures 2.0" x 0.9" x 0.28" (51mm x 23mm x 8mm) without headers soldered in
- Light as a (large?) feather - 5.7 grams
- [STM32F205RG](https://adafru.it/m9A) (https://adafru.it/m9A) 120MHz ARM Cortex M3 MCU
- [BCM43362](https://adafru.it/meC) (https://adafru.it/meC) 802.11b/G/N radio
- 128KB SRAM and 1024KB flash memory (total)
- 16KB SRAM and 128KB flash available for user code
- 16MBit (2MB) SPI flash for additional data storage
- Built in Real Time Clock (RTC) with optional external battery supply
- Hardware SPI and I2C (including clock-stretching)
- 12 standard GPIO pins, with additional GPIOs available via SPI, UART and I2C pins
- 7 standard PWM outputs, with additional outputs available via SPI, UART and I2C pins
- Up to 8 12-bit ADC inputs
- Two 12-bit DAC outputs (Pins A4 and SCK/A5)
- Up to 3 UARTs (including one with full HW flow control)
- TLS 1.2 support to access secure HTTPS and TCP servers
- On board single-cell LIPO charging and battery monitoring
- Fast and easy firmware updates to keep your module up to date
- Based on the excellent community-supported [Maple](https://adafru.it/mpE) (https://adafru.it/mpE) project

The WICED Feather uses the same standard pinout as the rest of the [Feather family](https://adafruit.it/m0b) (<https://adafruit.it/m0b>), allowing you to use the same FeatherWings across all your compatible devices.

It has the standard Feather on board LIPO battery charger (simply connect a LIPO battery and USB power at the same time), and 3.3V voltage regulation from either USB or VBAT (the LIPO cell) with automatic switching between power supplies.

Pin Multiplexing

The pins on the WICED Feather can be configured for several different purposes, with the main config options shown in the illustration below:



Accessing Pins in Software

For most pin names, you **must append 'P'** to the pin name shown on the silk screen. The table below lists the pin names on the silkscreen and their corresponding macro in your Arduino code:

Slikscreen	Arduino Code	Note(s)
WAKE	WAKE or PA0	-
C3	PC3	-
C2	PC2	-
A3	PA3	-
A2	PA2	-
A1	PA1	-
SCK	SCK or PA5	-
MOSI	MOSI or PA7	-
MISO	MISO or PA6	-
RX	PA10	-
TX	PA9	-
DFU	PB3	-
B5	PB5	-
SWCLK	PA14	-
SWDIO	PA13	-
A4	P14	-
B4	PB4	-
A15	PA15	-
C7	PC7	-
C5	PC5	-
SCL	PB6	-
SDA	PB7	-

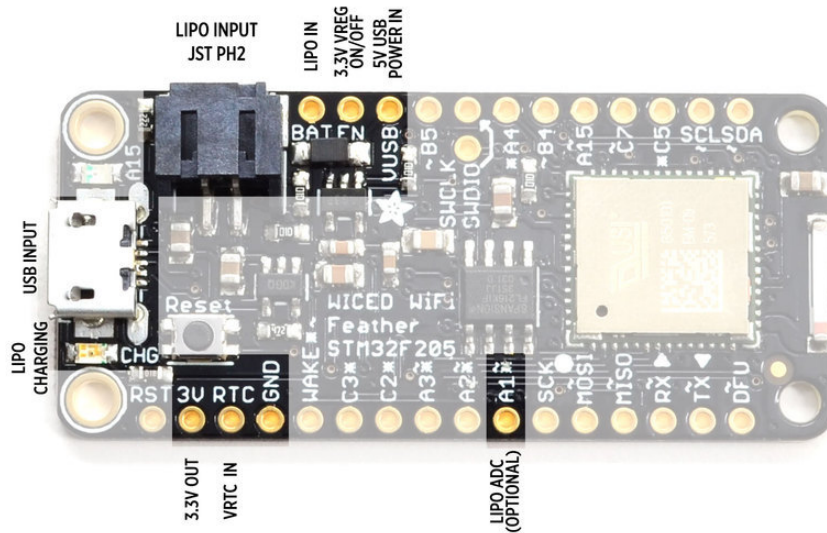
Other notable pins defined in [feather.h \(https://adafru.it/CaM\)](https://adafru.it/CaM) include:

Main Macro Name	Direct Arduino Pin Name
BOARD_LED_PIN	PA15

For further details on the board layout, see the [schematic here \(https://adafru.it/olE\)](https://adafru.it/olE).

Power Config

The WICED Feather can be run from either 5V USB power or a standard ~3.7V LIPO cell, and includes the ability to charge LIPO cells from USB power when both are connected at the same time.



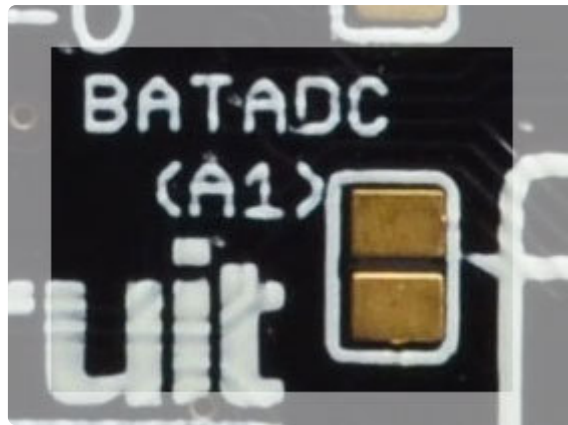
The following pins are included as part of the WICED Feather's power system:

- **3V**: The output of the on-board 3.3V 600mA voltage regulator
- **RTC**: The input for the real-time clock (RTC) on the STM32F205 (optional)
- **GND**: The common/GND pin which should be connect to GND on any other boards you use
- **BAT**: The input for the 3.7V LIPO cell
- **EN**: The 'EN' switch for the 3.3V voltage regulator. Set this to GND to disable power.
- **VUSB**: The 5V USB power input (USB VBUS)
- **A1**: This pin is optionally connected to a 10K+10K voltage divider that allows you to safely measure the output of the LIPO cell using the internal ADC (analog to digital converter).

LIPO Cell Power Monitoring (A1)

The LIPO battery level can optionally be monitored via a voltage divider configured on ADC pin **A1**.

To enable the 10K + 10K voltage divider (which will divide the LIPO voltage levels in half so that the ADC pin can safely read them), you need to solder shut the **BATADC** solder jumper on the bottom of the PCB:



This will allow you to read the voltage level of the LIPO cell using pin **A1** where each value on the ADC is equal to **0.80566mV** since:

- $3300\text{mV} / 4096 \text{ (12-bit ADC)} = 0.80566406\text{mV per LSB}$

You need to **double** the calculated voltage to compensate for the 10K+10K voltage divider, so in reality every value from the ADC is equal to **1.61133mV** on the LIPO cell, although it appears on the ADC at half that level.

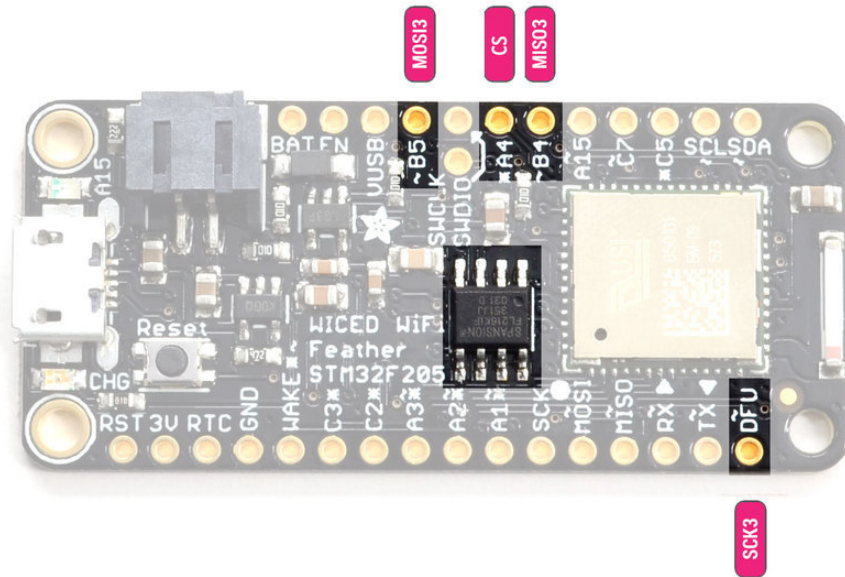
16 Mbit (2MByte) SPI Flash

The WICED Feather contains an optional (default = off) 16MBit SPI flash chip that is controlled by FeatherLib.

In order to keep the maximum number of pins available to customers, the SPI flash is disabled by default, but can be enabled with USB Mass Storage support so that you can access the contents on the flash memory from your PC to easily exchange data and files. Simply solder the **SPIFCS** solder jumper on the bottom of the device closed, and make sure you are running FeatherLib version 0.6.0 or higher to enabled flash and USB mass storage support.

The 16MBit SPI Flash is enabled starting with FeatherLib 0.6.0. Please make sure you are running a recent version of FeatherLib when working with the external flash memory.

The SPI3 bus used for SPI flash is controlled by FeatherLib, and the four pins shown below should be avoided in your own sketches when SPI Flash is enabled in a future FeatherLib release.



SPI flash is disabled by default. It can be enabled by soldering the **SPIFCS (A4)** solder jumper on the back of the PCB closed before powering the board up, which will connect the CS/SSEL of the SPI flash to pin **A4**:



PWM Outputs

Pins that can be used as PWM outputs are marked with a tilde character ("~") on the silk screen.

The timers associated with specific PWM outputs are listed below. These timers are important since all PWM outputs on the same HW timer will use the same period or pulse width. This means that if you set the pulse width for PA1, which uses HW Timer 5, this will also set the pulse width for PA2 and PA3 which use the same timer peripheral block.

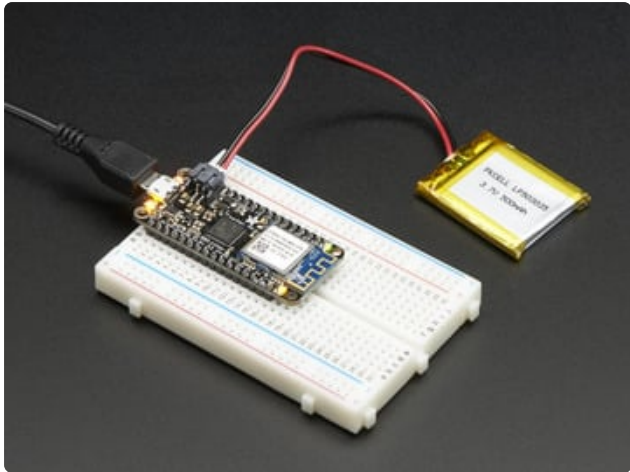
Pin Name	HW Timer	Notes
PA1	Timer 5	Status LED
PA2	Timer 5	
PA3	Timer 5	
PA15	Timer 2	
PB4	Timer 3	
PB5	Timer 3	
PC7	Timer 8	

Assembly

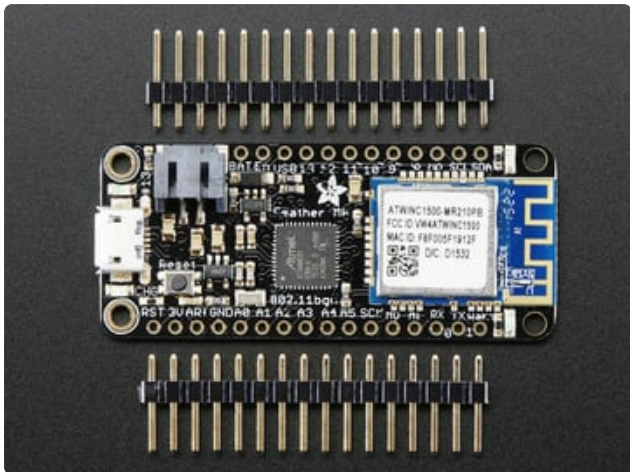
We ship Feathers fully tested but without headers attached - this gives you the most flexibility on choosing how to use and configure your Feather

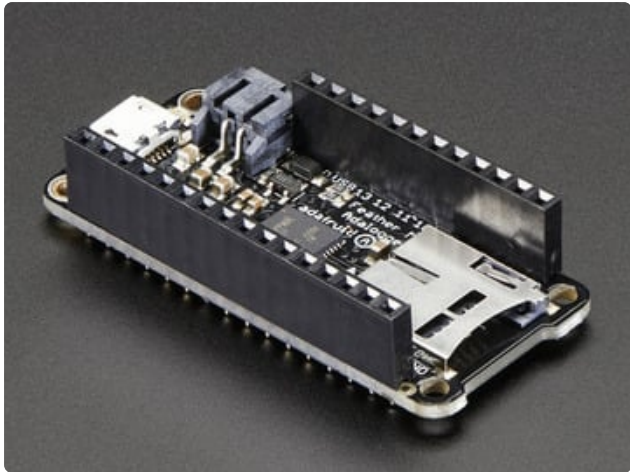
Header Options!

Before you go gung-ho on soldering, there's a few options to consider!

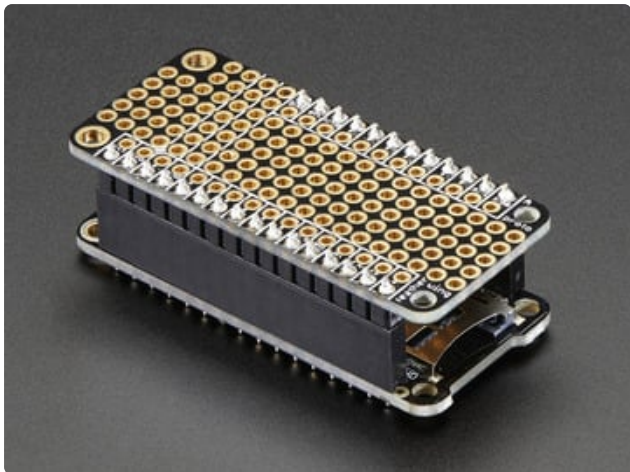


The first option is soldering in plain male headers, this lets you plug in the Feather into a solderless breadboard

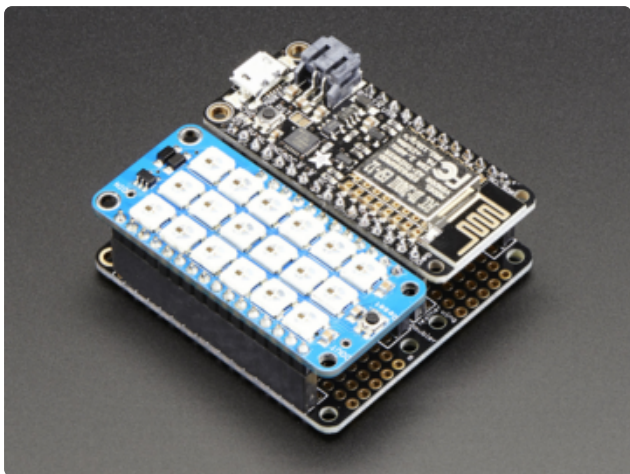


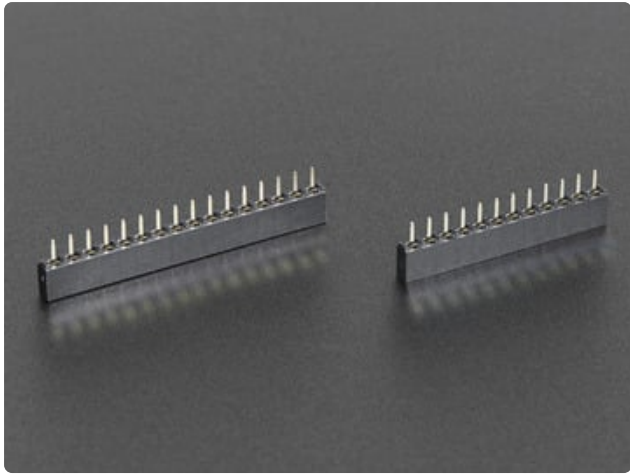


Another option is to go with socket female headers. This won't let you plug the Feather into a breadboard but it will let you attach featherwings very easily

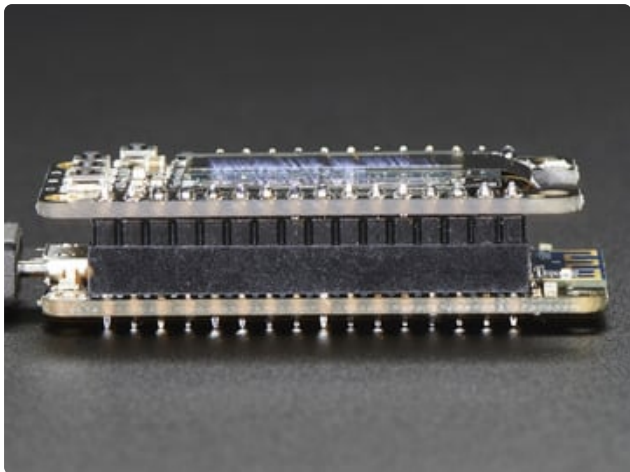


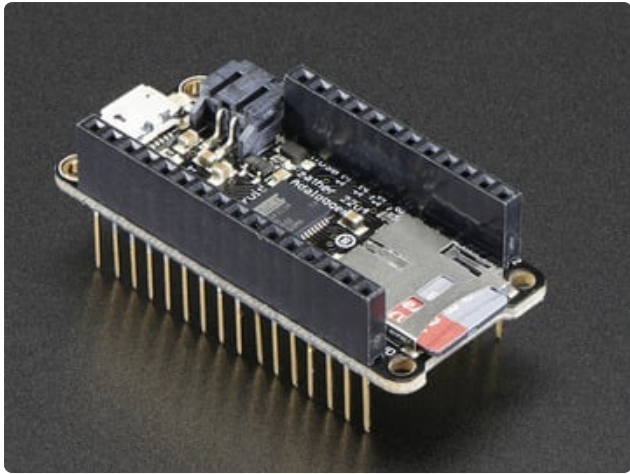
A few Feather boards require access to top-side components like buttons or connectors, making stacking impractical. Sometimes you can stack in the opposite order—FeatherWing underneath—or, if both Feather and Wing require top-side access, place the boards side-by-side with a [FeatherWing Doubler](http://adafru.it/2890) (<http://adafru.it/2890>) or [Tripler](http://adafru.it/3417) (<http://adafru.it/3417>).



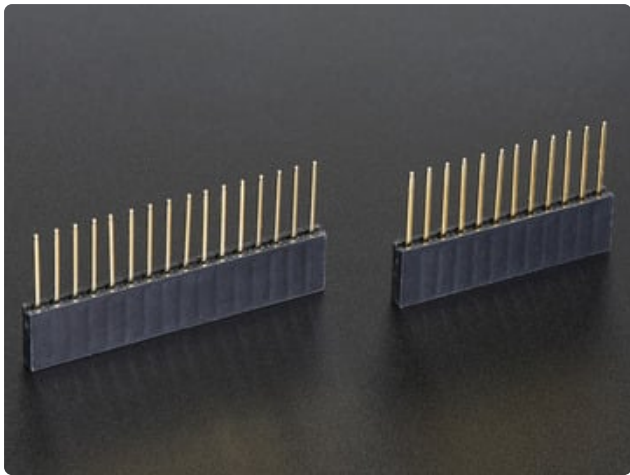


We also have 'slim' versions of the female headers, that are a little shorter and give a more compact shape

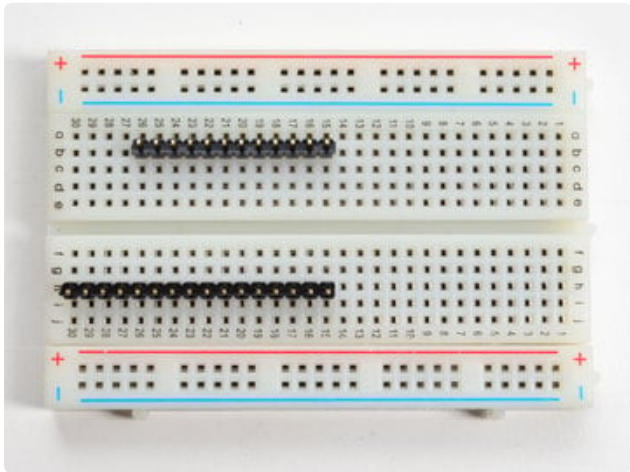




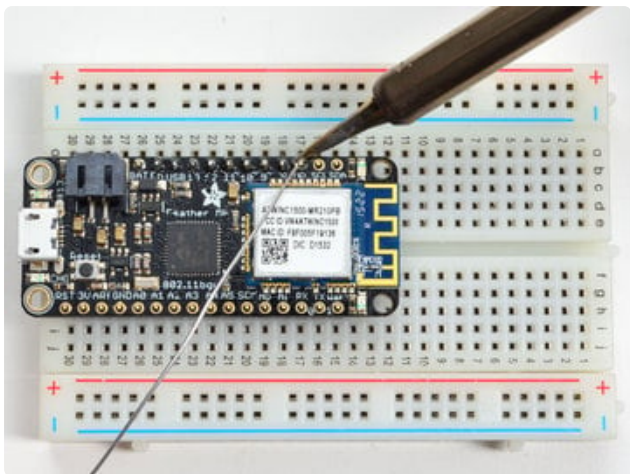
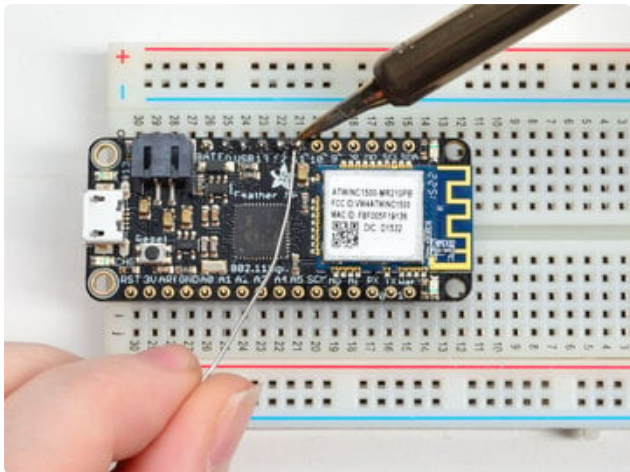
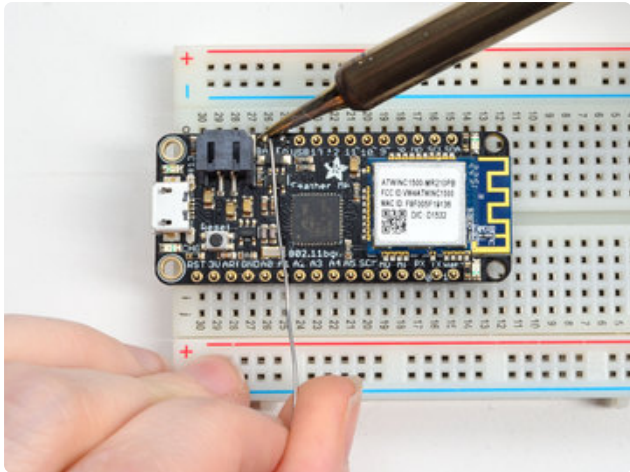
Finally, there's the "Stacking Header" option. This one is sort of the best-of-both-worlds. You get the ability to plug into a solderless breadboard and plug a featherwing on top. But its a little bulky



Soldering in Plain Headers



Prepare the header strip:
Cut the strip to length if necessary. It will be easier to solder if you insert it into a breadboard - **long pins down**



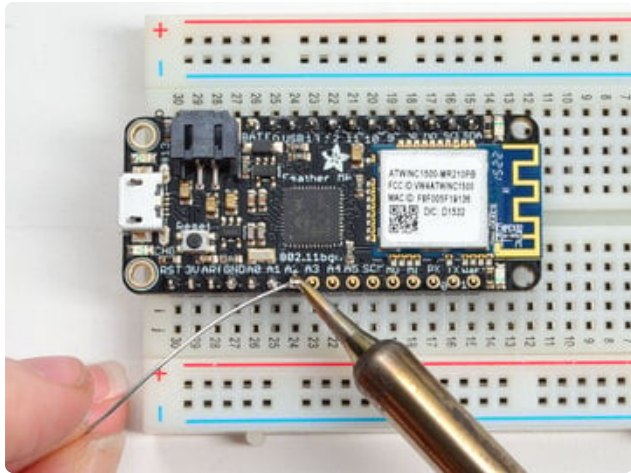
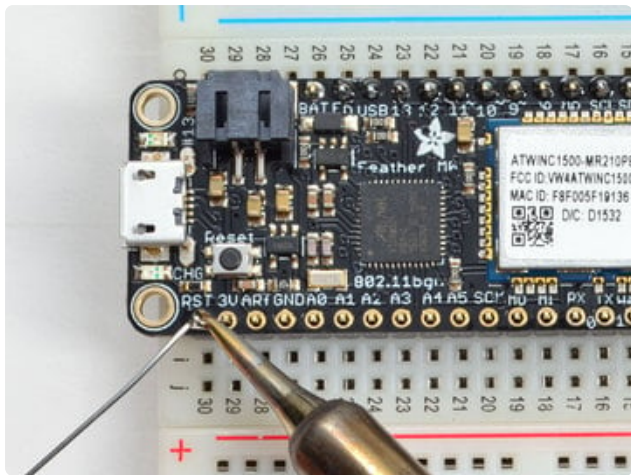
Add the breakout board:

Place the breakout board over the pins so that the short pins poke through the breakout pads

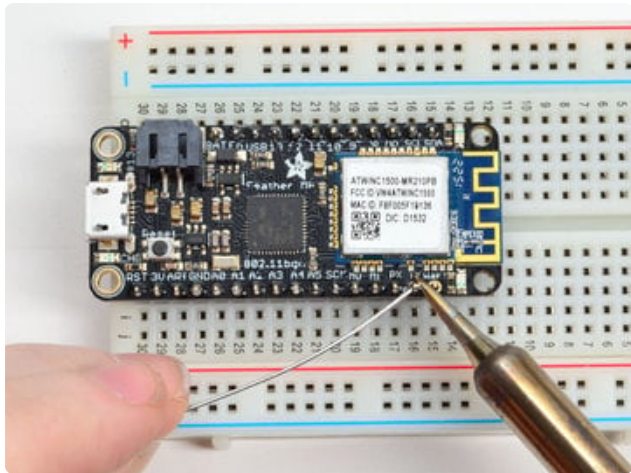
And Solder!

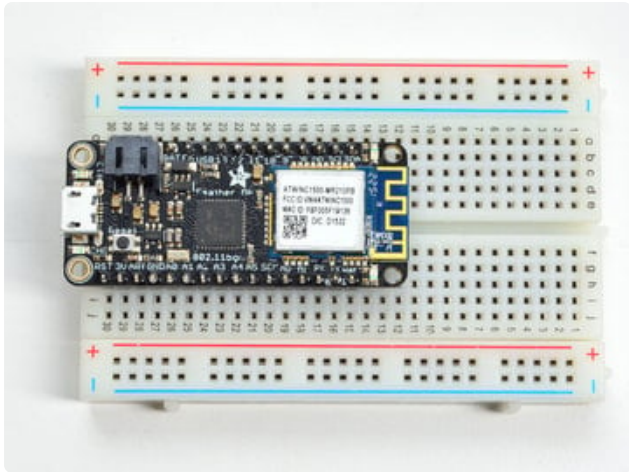
Be sure to solder all pins for reliable electrical contact.

(For tips on soldering, be sure to check out our [Guide to Excellent Soldering](https://adafruit.it/aTk) (<https://adafruit.it/aTk>)).



Solder the other strip as well.





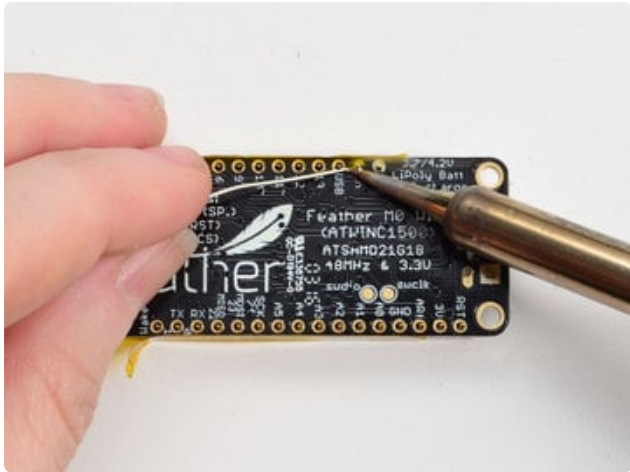
You're done! Check your solder joints visually and continue onto the next steps

Soldering on Female Header



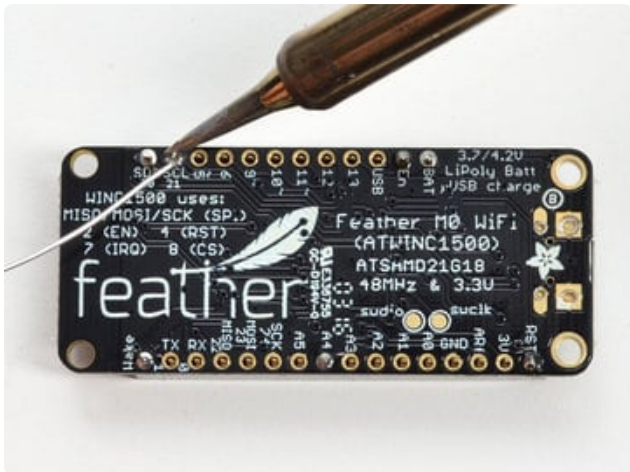
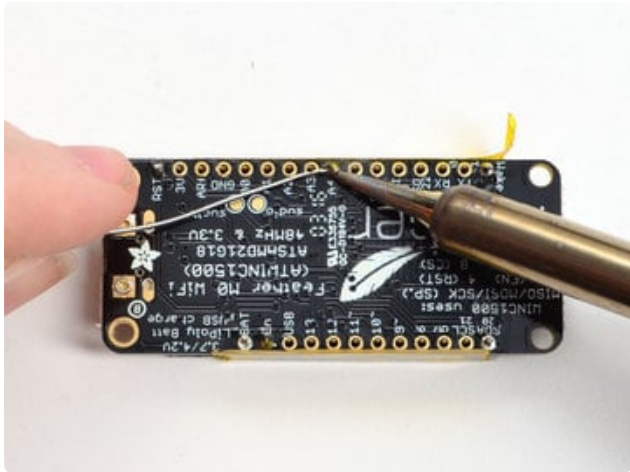
Tape In Place

For sockets you'll want to tape them in place so when you flip over the board they don't fall out



Flip & Tack Solder

After flipping over, solder one or two points on each strip, to 'tack' the header in place



And Solder!

Be sure to solder all pins for reliable electrical contact.

(For tips on soldering, be sure to check out our [Guide to Excellent Soldering \(https://adafruit.it/aTk\)](https://adafruit.it/aTk)).



You're done! Check your solder joints visually and continue onto the next steps

Get the WICED BSP

The WICED BSP installation procedure for 0.6.0 and higher is completely different than the manual installation procedure from earlier versions. See the notes at the bottom of this page if you are upgrading.

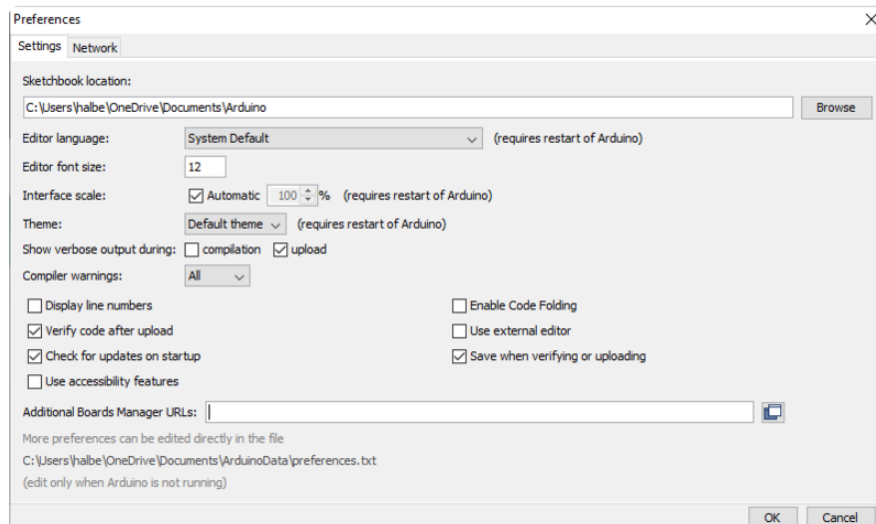
To use the WICED Feather, you first need to install a board support package (BSP) that includes all the classes, drivers and example code that make it possible to create projects that can talk to the STM32F205 MCU and Broadcom radio. This guide will walk you through the process of getting the BSP setup on your development machine.

This guide is based on Arduino 1.6.5 or higher. You will need a similar version of the Arduino IDE to follow this guide, which was tested with 1.6.11.

Adding Adafruit Board Support

The first thing you will need to do is start the IDE and navigate to the **Preferences** menu. You can access it from the **File** menu in Windows or Linux, or the **Arduino** menu on OS X.

A dialog like this will pop up:



We will be adding a URL to the new **Additional Boards Manager URLs** option. The list of URLs is comma separated, and you will only have to add each URL once. New Adafruit boards and updates to existing boards will automatically be picked up by the Board Manager each time it is opened. The URLs point to index files that the Board Manager uses to build the list of available & installed boards.

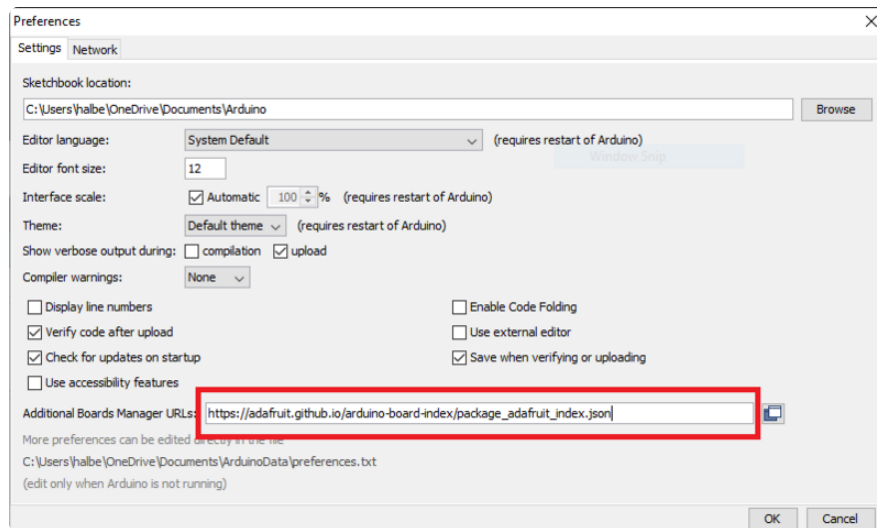
If you don't see the Additional Boards Manager URLs box, make sure you downloaded the Arduino IDE from arduino.cc! Older versions and derivatives of the IDE may not have it

Add the Adafruit BSP List

We will only need to add one URL to the IDE in this example, but you can add multiple URLs by separating them with commas. Copy and paste the link below into the **Additional Boards Manager URLs** option in the Arduino IDE preferences.

```
https://adafruit.github.io/arduino-board-index/package_adafruit_index.json
```

You should see something like this:



Click **OK** to save the new preference settings. Next we will look at installing boards with the Board Manager.

Add the Adafruit WICED BSP

Adding the link to the Adafruit board support package does not actually install anything, it only tells the Arduino IDE where to find the software.

Now that you have added the appropriate URLs to the Arduino IDE preferences, you can open the **Boards Manager** by navigating to the **Tools->Board** menu item.

Once the Board Manager opens, click on the **category** drop down menu on the top left hand side of the window and select **Contributed**. You will then be able to select and install the boards supplied by the URLs added to the preferences.

Find the example named **Adafruit WICED** from the list and click the **Install** button:



Next, **quit and reopen the Arduino IDE** to ensure that all of the boards are properly installed. You should now be able to see the new boards listed in the **Tools->Board** menu.

Finally follow the OS specific steps in this guide for your platform to finish the installation - basically installing drivers and permissions management.

Upgrading From Earlier WICED BSP Releases (<0.6.0)

If you are using an earlier version of the WICED SDK (< 0.6.0), you will need to remove the old files from the `/hardware/Adafruit_WICED_Arduino` folder before starting this guide. You may also need to delete the '`arduino15/staging`' dir in the Arduino installation folder before the BSP appears.

Windows Setup

To setup the WICED Feather on Windows, the following steps are necessary:

This page assumes you have already installed the WICED Feather BSP, as detailed earlier in this guide.

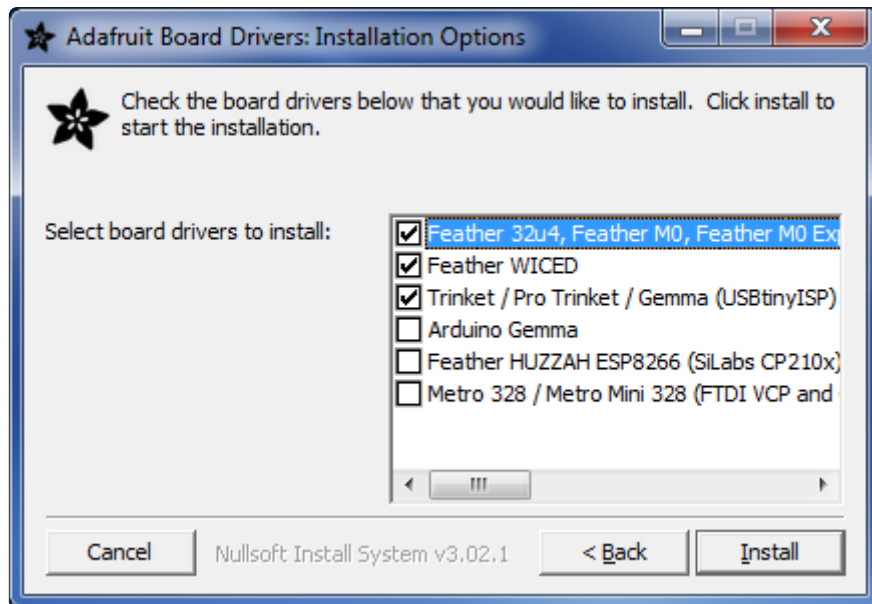
Install Adafruit Windows Drivers

If you are using a Windows based system, you will need to install a set of drivers for the USB DFU, USB CDC and other USB interfaces used by the WICED Feather to perform firmware updates and communicate with the device.

Adafruit provides a convenient [Adafruit Windows Drivers \(https://adafru.it/mb8\)](https://adafru.it/mb8) installer that takes care of the details for you. Simply download and install the package below:

**Visit the Adafruit Windows Drivers
download page**

<https://adafru.it/mb8>



Once the installation process is complete, you should be able to plug your WICED Feather into your system and it will be recognized thanks to the signed drivers you just installed.

Install libusb 0.1 Runtime

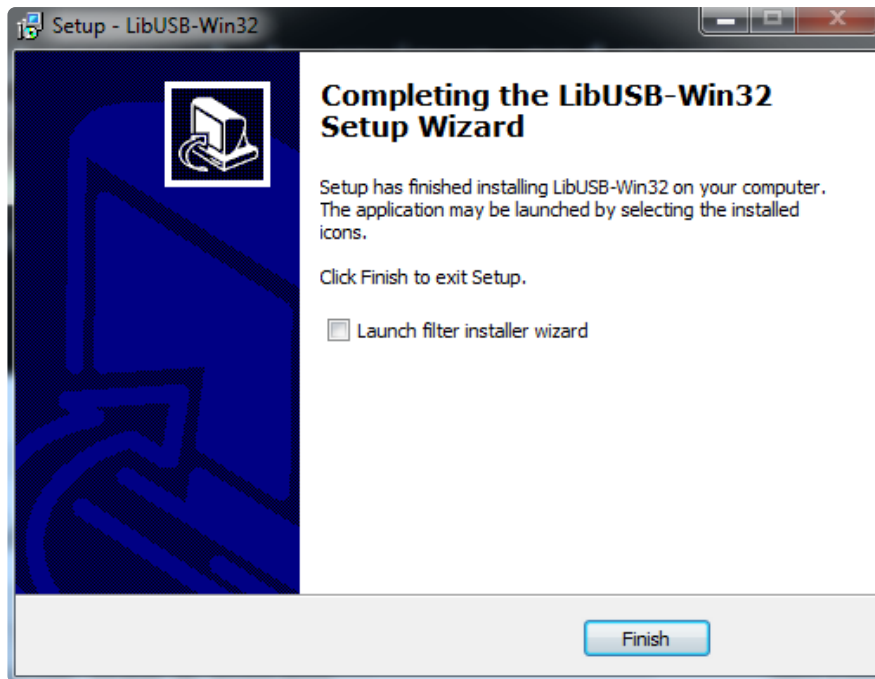
To use libusb (which is required to communicate with the WICED Feather), you will first need to install a pre-compiled libusb runtime.

You can install this by downloading and running [libusb-win32 driver \(https://adafru.it/mb9\)](https://adafru.it/mb9), taking care to select the file named **libusb-win32-devel-filter-1.2.6.0.exe**.

Download libusb-win32-devel-filter-1.2.6.0.exe from SourceForge

<https://adafru.it/mba>

Make sure to **DISABLE** the 'Launch filter installer wizard' option at the end of the installation process!



Install Python 2.7

Python is used by the WICED Feather for a number of cross-platform tools and scripts, meaning that you will need to install [Python 2.7](https://adafru.it/mbb) (ideally 2.7.9 or higher) on your system in order to communicate with the board.

Depending on whether you are running a 32-bit (x86) or a 64-bit (AMD x64) version of Windows, download the installer linked below and start the installation process:

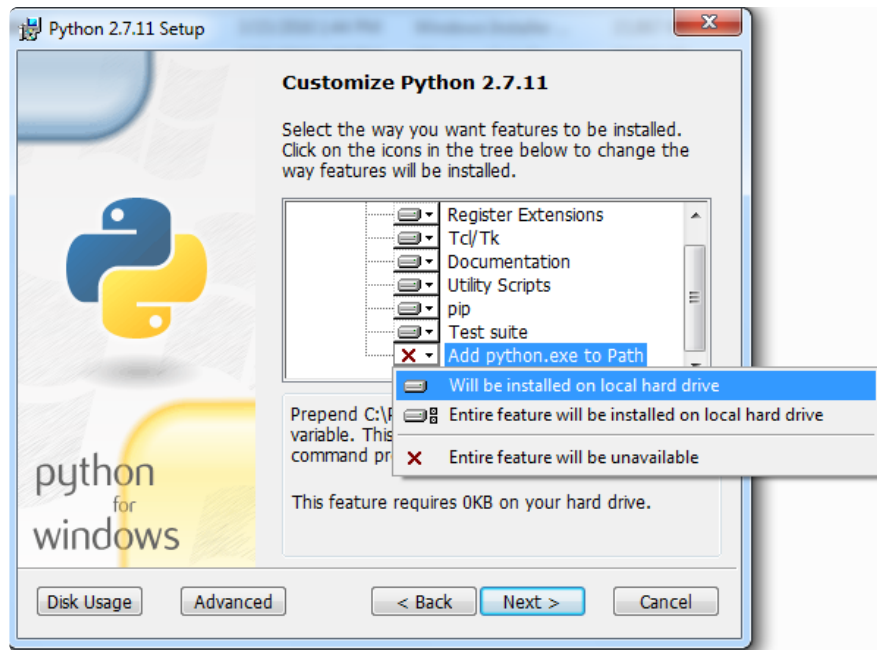
Click here to download the Python 2.7.11 Windows 32-bit (x86) Installer

<https://adafru.it/mbc>

Click here to download the Python 2.7.11 Windows 64-bit (AMD x64) Installer

<https://adafru.it/mbd>

During the installation process make sure that you enable the option to add Python to the system path (the option is disabled by default). This is required for the Arduino IDE to be able to access the python scripts it needs to communicate with the WICED Feather:



Testing the Python Installation

Once the installer is finished you can open the command line and enter the following command to test the availability of Python on your system:

```
python --version
```

You should see something like this:

```
Python 2.7.11
```

Install Python Tools

Update: Recent versions of the BSP now include a pre-compiled version of the feather_dfu tool in the '/tools/win32-x86/feather_dfu' folder, which should run on most systems once the libusb dependencies above are installed. You will still need python for the pycert tool though.

The WICED Feather BSP uses a few Python based tools to allow the Arduino IDE to talk to the hardware in a platform-independent manner (specifically **tools/source/feather_dfu/feather_dfu.py**).

To use these Python tools, you will need a few additional libraries to make the python scripts work.

Running the following command from the command line will install these dependencies:

```
pip install --pre pyusb
pip install click
```

This will display some basic progress data on the installation process, and you should end up with something resembling the following output:

```
C:\Users\me>pip install --pre pyusb
Collecting pyusb
  Downloading pyusb-1.0.0rc1.tar.gz (53kB)
    100% |#####| 57kB 1.3MB/s
Installing collected packages: pyusb
Running setup.py install for pyusb
Successfully installed pyusb-1.0.0rc1
```

Testing the Installation

You can test if Python is setup correctly by going to the '/tools/source/feather_dfu' folder in the WICED Feather BSP and running the following command with the WICED Feather connected:

This step assumes you have already installed the Arduino IDE and the WICED Feather BSP, detailed earlier in this learning guide.

```
cd \tools\source\feather_dfu
python feather_dfu.py info
```

This should display something resembling the following output:

```
Feather
ST32F205RGY
353231313533470E00420037
FF:FF:FF:FF:FF:FF
1.0.0
3.5.2
0.5.0
0.5.0
Mar  8 2016
```

If you don't see any output when running this tool and you are using a new board, you may need to flash a user sketch to the module via the Arduino IDE. See the 'Arduino IDE Setup' page in this guide for details on how to flash a user sketch.

[FAQ \(https://adafru.it/w4A\)](https://adafru.it/w4A). However, you can also update FeatherLib directly from the Arduino IDE as follows:

- Set the DFU pin to GND on your WICED Feather
- Reset the device with DFU connected to GND, which will force it to enter USB DFU mode, and you should see a fast blinky pattern on the RED LED, indicating that you are in DFU mode.
- Disconnect the DFU pin from GND.
- With **Adafruit WICED Feather** selected as the Board Target, changed the section to **Feather Lib (Release)**.
- Now compile any simple sketch, and flash it to the device. This will compile the sketch, but Feather Lib will actually be flashed, not the sketch you just compiled.
- Once the flashing process is done, change the section back to **User Code** and then flash your sketch again, which will now flash a simple sketch. A blinky example is best since you can see the results.
- When you reset your device, you should now have an updated FeatherLib as well as a valid user sketch, meaning that the two USB CDC ports can enumerate, since a valid code entry point has been found in the valid user sketch.

If you continue to have problems, please post to the [Adafruit Support Forum \(https://adafru.it/dYq\)](https://adafru.it/dYq) with the following information:

- A screenshot from the USB Device View tool showing the Adafruit Industries entries if present (to validate driver installation)
- Indicate whether you can successfully enter DFU mode by connected the DFU pin to GND and resetting.
- The results of running the **dfu-util -l** command with the WICED Feather connected, which will let us know if the USB DFU device was detected

OS X Setup

To setup the WICED Feather on OS X, the following steps are necessary:

This page assumes you have already installed the WICED Feather BSP, as detailed earlier in this guide.

Install dfu-util

The WICED Feather uses USB DFU to perform firmware updates from the Arduino IDE. To enable to Arduino IDE to talk to the board you will need to install dfu-util.

The easiest way to install dfu-util is to use [homebrew](https://adafru.it/df3) (<https://adafru.it/df3>), which can be installed with the following command if it doesn't already exist on your system:

```
/usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

Once homebrew is installed you can install **dfu-util** from the command line with the following command:

```
brew install dfu-util
```

Testing the Installation

You can check if dfu-util was installed correctly by running the following command with the WICED Feather connected:

Make sure your board is in DFU mode before running this command. You can enter DFU mode by double-clicking the RESET button quickly, or by setting the DFU pin to GND at startup. You'll know that are in DFU mode because the status LED will blink at a 5Hz rate.

```
dfu-util --list
```

This should give you results resembling the following output:

```
dfu-util 0.8

Copyright 2005-2009 Weston Schmidt, Harald Welte and OpenMoko Inc.
Copyright 2010-2014 Tormod Volden and Stefan Schmidt
This program is Free Software and has ABSOLUTELY NO WARRANTY
Please report bugs to dfu-util@lists.gnumonks.org

Deducing device DFU version from functional descriptor length
Found DFU: [239a:0008] ver=0200, devnum=12, cfg=1, intf=0, alt=0, name="@Internal Flash / 0x08000000/02*016Ka,02*016Kg,01*064Kg,07*128Kg", serial="000000000001C"
```

Install Python Tools

The WICED Feather BSP uses a few Python based tools (see the **tools/** folder for details).

To use these Python tools, you will need to have Python available on your system (which OS X does by default), but you will also need a few additional libraries to make the Python scripts work.

You will also need the [pip utility \(https://adafru.it/19KF\)](https://adafru.it/19KF) if it is not already on your system.

Running the following command from the command line will install these dependencies:

Depending on your system setup you may need to run the pip commands with 'sudo'

```
# On versions of OS X from 10.11.5 onward run ...
sudo pip install pyusb
sudo pip install click

# On versions of OS X before 10.11.5 run ...
sudo pip install --pre pyusb
sudo pip install click
```

If you get an error like '-bash: pip: command not found' you can install pip via 'sudo easy_install pip'

Testing the Installation

You can test if Python is setup correctly by going to the '/tools/source' folder in the WICED Feather BSP and running the following command with the WICED Feather connected:

As of BSP release 0.6.5 and higher the feather_dfu Python tool has been converted to a binary tool called wiced_dfu, and the section below should only be followed on earlier versions of the BSP. Version 0.6.5 and higher ship with pre-compiled versions of wiced_dfu, or you can build the binary yourself using the makefile in the tools/wiced_dfu folder.

```
cd tools/source/feather_dfu
python feather_dfu.py info
```

This should display something resembling the following output:

```
Feather
ST32F205RGY
353231313533470E00420037
FF:FF:FF:FF:FF:FF
1.0.0
3.5.2
0.5.0
0.5.0
Mar  8 2016
```

Optional: Install AdaLink

If you ever need to reflash the USB DFU bootloader on the WICED Feather (which will require either a [Segger J-Link](http://adafru.it/1369) (<http://adafru.it/1369>) or an [STLink/V2](http://adafru.it/2548) (<http://adafru.it/2548>)), you will also need to install a utility called [AdaLink](https://adafru.it/fPq) (<https://adafru.it/fPq>).

AdaLink acts as a simple python-based abstraction layer between various HW debuggers, and the different ARM MCU families that we use at Adafruit.

For installation instructions on AdaLink see the [Readme file](https://adafru.it/fPq) (<https://adafru.it/fPq>) in the git repository.

Linux Setup

To setup the WICED Feather on Linux (Ubuntu 14.04 was used here) the following steps are necessary:

UDEV Setup

On Linux you will need to add a small udev rule to make the WICED board available to non-root users. If you don't have this rule then you'll see permission errors from the Arduino IDE when it attempts to program the board.

Create or edit a file called `/etc/udev/rules.d/99-adafruit-boards.rules` and add the following lines:

```
PID 0008 = DFU Mode, 0010 = Application Mode/CDC, 8010 = Application
Mode/CDC + USB Mass Storage
```

```
# This file is used to gain permission for the WICED Feather module
# Copy this file to /etc/udev/rules.d/

ACTION!="add|change", GOTO="adafruit_rules_end"
```

```
SUBSYSTEM!="usb|tty|hidraw", GOTO="adafruit_rules_end"

# Please keep this list sorted by VID:PID

# WICED Feather in DFU mode
ATTRS{idVendor}=="239a", ATTRS{idProduct}=="0008", MODE="664", GROUP="plugdev"

# WICED Feather in Application mode
ATTRS{idVendor}=="239a", ATTRS{idProduct}=="0010", MODE="664", GROUP="plugdev"
ATTRS{idVendor}=="239a", ATTRS{idProduct}=="8010", MODE="664", GROUP="plugdev"

LABEL="adafruit_rules_end"
```

Depending on your distribution you might need to change `GROUP="plugdev"` to a different value like `"users"` or `"dialout"`. The dialout group should work for Ubuntu.

Then restart udev with:

```
sudo restart udev
```

Or on systemd-based systems like the latest Debian or Ubuntu 15.04+ restart udev with:

```
sudo systemctl restart udev
```

Install dfu-util

The WICED Feather uses USB DFU to perform firmware updates from the Arduino IDE. To enable to Arduino IDE to talk to the board you will need to install dfu-util.

Many Linux distributions include a binary version of dfu-util in their package management system, but they are often out of date and lower than the 0.8 version required by the WICED Feather.

If you are using Ubuntu 15.04 or higher, you can install dfu-util 0.8 via the following command:

```
sudo apt-get install dfu-util
```

If you are using an older version of Ubuntu or if `'dfu-util -v'` displays an older version like 0.5 you will need to build dfu-util from source, as described below.

Building dfu-util From Source (Ubuntu 14.04 etc.)

Ubuntu 14.04 and several other distributions use dfu-util 0.5 which is too old for the WICED Feather (which requires dfu-util version **0.8** or higher).

To build dfu-util from source run the following commands (Ubuntu 14.04 is assumed here), first install the required build dependencies:

```
sudo apt-get install git
sudo apt-get build-dep dfu-util
sudo apt-get install libusb-1.0-0-dev
```

Then download the git repo containing the dfu-util source:

```
git clone https://git.code.sf.net/p/dfu-util/dfu-util
cd dfu-util
```

Then build the dfu-util from source:

```
./autogen.sh
./configure # on most systems
make
```

You can then install and verify dfu-util via the following commands, which should show version 0.8 or 0.9 for dfu-util:

```
sudo make install
hash -r
dfu-util -V
```

Testing the Installation

You can check if dfu-util was installed correctly by running the following command with the WICED Feather connected:

```
dfu-util --list
```

This should give you the following output:

```
dfu-util 0.9

Copyright 2005-2009 Weston Schmidt, Harald Welte and OpenMoko Inc.
Copyright 2010-2016 Tormod Volden and Stefan Schmidt
This program is Free Software and has ABSOLUTELY NO WARRANTY
Please report bugs to http://sourceforge.net/p/dfu-util/tickets/

Found DFU: [239a:0008] ver=0200, devnum=6, cfg=1, intf=0, path="2-1", alt=0,
name="@Internal Flash /0x08000000/02*016Ka,02*016Kg,01*064Kg,07*128Kg",
serial="000000000001C"
```

Install Python Tools (BSP <= 0.6.2)

As of BSP release 0.6.5 and higher the feather_dfu Python tool has been converted to a binary tool called wiced_dfu, and the section below should only be followed on earlier versions of the BSP. Version 0.6.5 and higher ship with pre-compiled versions of wiced_dfu, or you can build the binary yourself using the makefile in the tools/wiced_dfu folder.

The WICED Feather BSP uses a few Python based tools to allow the Arduino IDE to talk to the hardware in a platform-independent manner (specifically **tools/feather_dfu/feather_dfu.py**).

To use these Python tools, you will need to have Python available on your system (which most Linux distributions do by default), but you will also need a few additional libraries to make the python scripts work.

Running the following command from the command line will install these dependencies:

```
sudo pip install --pre pyusb
sudo pip install click
```

Testing the Installation

You can test if Python is setup correctly by going to the '/tools/feather_dfu' folder in the WICED Feather BSP and running the following command with the WICED Feather connected:

```
cd tools/feather_dfu
sudo python feather_dfu.py info
```

This should display something resembling the following output:

```
Feather
ST32F205RGY
353231313533470E00420037
FF:FF:FF:FF:FF:FF
1.0.0
3.5.2
0.5.0
0.5.0
Mar  8 2016
```

Optional: Install AdaLink

If you ever need to reflash the USB DFU bootloader on the WICED Feather (which will require either a [Segger J-Link](http://adafru.it/1369) (<http://adafru.it/1369>) or an [STLink/V2](http://adafru.it/2548) (<http://adafru.it/2548>)), you will also need to install a utility called [AdaLink](https://adafru.it/fPq) (<https://adafru.it/fPq>).

AdaLink acts as a simple python-based abstraction layer between various HW debuggers, and the different ARM MCU families that we use at Adafruit.

For installation instructions on AdaLink see the [Readme file](https://adafru.it/fPq) (<https://adafru.it/fPq>) in the git repository.

External Resources

For further details on setting up Linux for the WICED Feather see the following links:

- [Adafruit Feather WICED and Ubuntu 14.04](https://adafru.it/mRc) (<https://adafru.it/mRc>)

Arduino IDE Setup

Once you have the WICED Feather board support package set up -- as described in [Get the WICED BSP](https://adafru.it/rod) (<https://adafru.it/rod>) earlier in this guide -- you can start compiling code against FeatherLib or update the firmware on your device directly from the Arduino IDE.

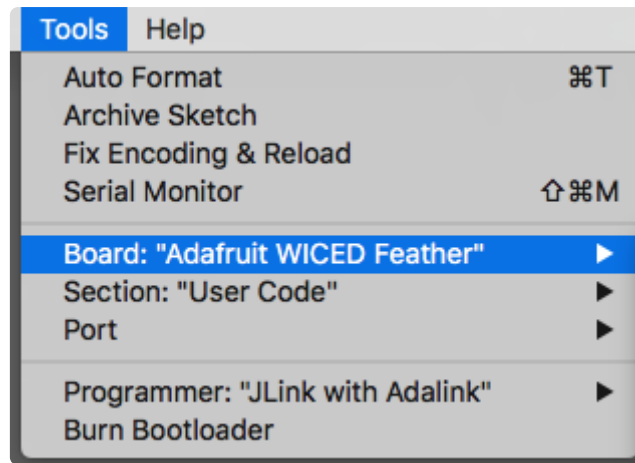
To make sure that the Arduino IDE has access to all of the tools, libraries and config data it needs, however, you will first need to make some adjustments in the IDE:

Board Selection

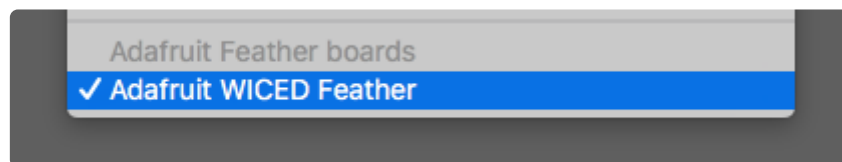
The first thing to do (assuming that you already have the WICED BSP installed on your system, as describe in **Get the WICED BSP** earlier in this guide!) is to make sure that you have **Adafruit WICED Feather** selected as the Board target.

Selecting the right board target is critical since the board target is what causes the FeatherLib support files to be included as part of the build process!

To change the board target, simply click the **Tools > Board** menu item and then select **Adafruit WICED Feather** under the 'Adafruit Feather Boards' heading:



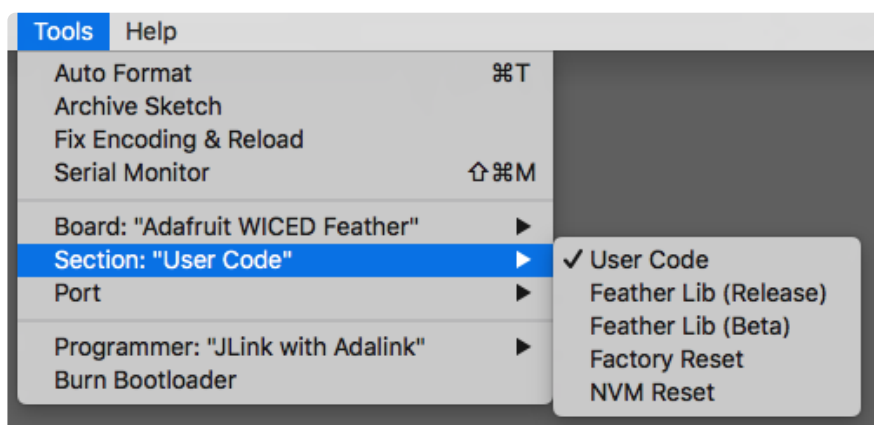
The actual position of the board in your menu will depend on your system setup, but it should resemble the following image:



Setting the 'Section'

As described in the System Architecture page in this guide, the WICED Feather is broken up into three separate firmware images: the user code, FeatherLib, and the USB DFU bootloader.

Each of these firmware images exists in a specific section of the flash memory on the STM32F205 MCU, and you can switch between the two user-modifiable sections via the **Tools > Section** menu:



The following sections are available in the menu:

- **User Code:** This section (which consists of 128KB flash and 16KB SRAM) is where your user sketches go, which is the project that you compile in the Arduino IDE. **This is the section you will want to use 99% of the time!**
- **Feather Lib:** This is the library that contains the low level WiFi stack and security layer, manage the RTOS (real time operating system) that schedules different tasks on the system, and does all of the heavy lifting for you. By selecting 'Feather Lib' as the section and then flashing your WICED Feather like you would for a normal project you can either reflash or update the FeatherLib on your hardware. If you update the WICED Feather BSP and a new version of FeatherLib is available, you would do this once to update your device and then switch back to 'User Code'.
 - **Feather Lib (Release):** This will flash the latest release version of FeatherLib
 - **Feather Lib (Beta):** This will flash the latest BETA release of FeatherLib if one is available. If no BETA version is available, this is generally identical to the release files. You should check the FeatherLib version numbers to verify if there is a difference.
- **Factory Reset:** Selecting this 'section' and then flashing your device is a bit of a hack since it won't actually flash a sketch, but it will use the feather_dfu.py tool to perform a factory reset on your device in case it went off into the weeds somehow.
- **NVM Reset:** Similar to the factory reset above, selectiing this section and then flashing your device will cause the non-volatile config memory on your WICED Feather to be reset to factory defaults (although the rest of the device, such as the user code, will be left untouched).

To flash the appropriate code to the device (or perform a factory reset or NVM reset), you simply need to change the section and click the **Sketch > Upload** tools menu, or click the arrow icon in the Arduino IDE (the second icon from the left below):



If you select FeatherLib, Factory Reset or NVM Reset (which require no code compilation themselves) a full project compilation will still take place before FeatherLib is flashed or a reset is performed. The compiled user code will not be used, but the compilation process can't be avoided due to the nature of sections in the Arduino IDE.

Selecting the Serial Port

Note: Per the [forums \(https://adafru.it/ven\)](https://adafru.it/ven):

When there is no sketch loaded, the WICED enters DFU mode automatically, and no comms ports will display.

You do not need comms ports to load a Sketch, just the libusb driver. When you load a non empty Sketch, featherlib is supposed to now enumerate the comms ports.

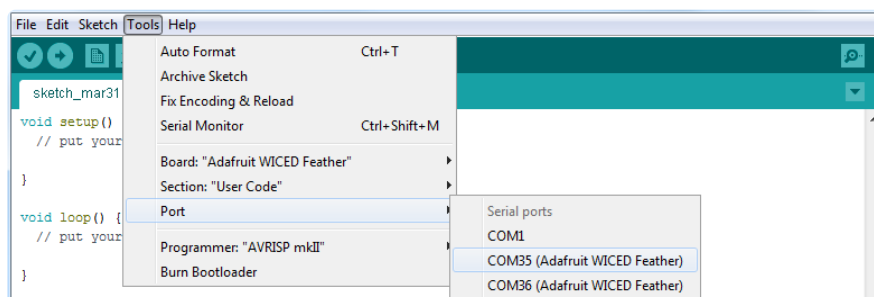
Reload the Feather Lib with a non empty Sketch, then the comms ports will show up.

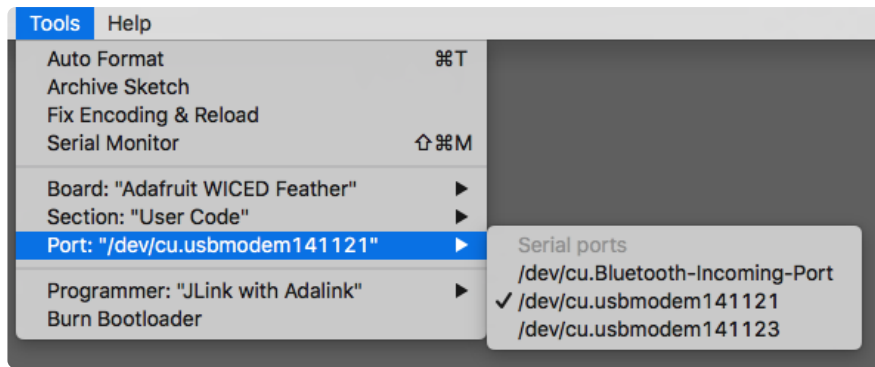
So in tools, select "section" "Feather Lib (Release)", and flash it with the "blink" code loaded in the IDE. If that doesn't work, switch to "User Code" and now flash "blink".

By default, two USB CDC serial ports will be enumerated with the WICED Feather. One serial port will be used for general purpose serial data and is connected to the **Serial Monitor**. This is the port you should normally select in the Arduino IDE.

The second port that is enumerated is for basic debugging and for future expansion, and enumerates a currently unused AT Parser that only supports a very basic set of commands (for example 'ATI' will return some basic information about the module).

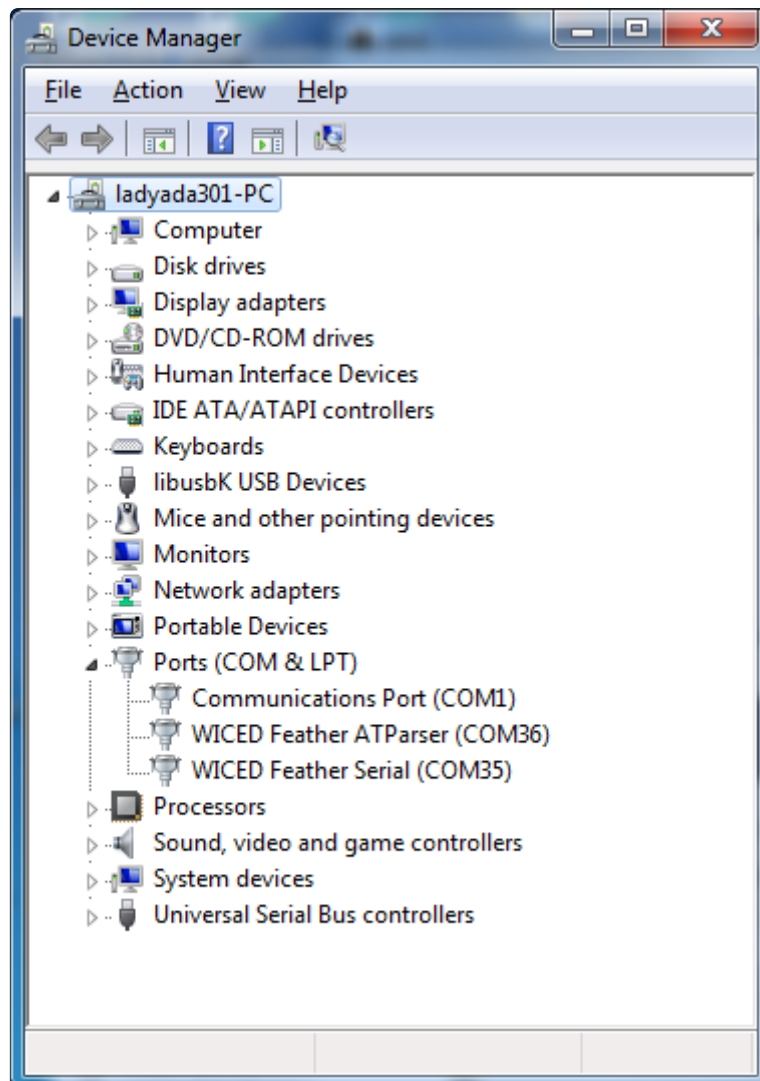
If you are not seeing a USB CDC port and are using a new WICED board, please see this FAQ: <https://learn.adafruit.com/introducing-the-adafruit-wiced-feather-wifi/faqs#faq-6>





With the right serial port selected (normally the numerically lowest number is the Serial Monitor COM port, though it's random and may change from one system to the next), you can open the **Serial Monitor** and you can send and receive serial data like you would with any other Arduino development board.

On Windows, you can verify which COM port corresponds to which function by opening the Device Manager and examining the list of serial ports. COM35 below is the Serial Monitor port (**WICED Feather Serial**) and COM36 is the AT parser port (**WICED Feather ATParser**).



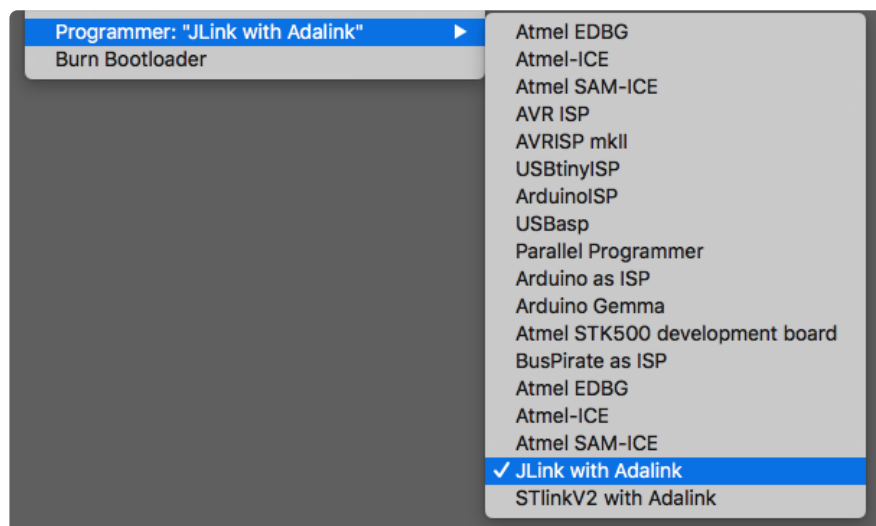
Optional: Updating the Bootloader

While you should never have to update the USB DFU bootloader on your WICED Feather, if you have a [Segger J-Link](http://adafru.it/1369) (<http://adafru.it/1369>) or an [STLink/V2](http://adafru.it/2548) (<http://adafru.it/2548>) you can reflash the normally read-only bootloader from within the Arduino IDE.

A J-Link or STLink is required since this is the only way to talk to the STM32F205 if the bootloader is somehow erased.

To reflash the bootloader hook the SWDIO, SWCLK, RESET and GND pins up to the pins of the same name on the WICED Feather (see the JLink or STLink/V2 pinout to know where to find these pins on your debugger). If you are using a JLink, make sure to also connect the VTRef pin to 3.3V on the WICED Feather since it needs to know the logic level for the target device.

Select the appropriate debugger from the **Tools > Programmer** menu (only the J-Link or STLink options will work!):



You can then click the **Burn Bootloader** menu entry and the Arduino IDE will attempt to use the JLink or STLink (via [AdaLink](https://adafru.it/fPq) (<https://adafru.it/fPq>)) to reflash the bootloader for you.

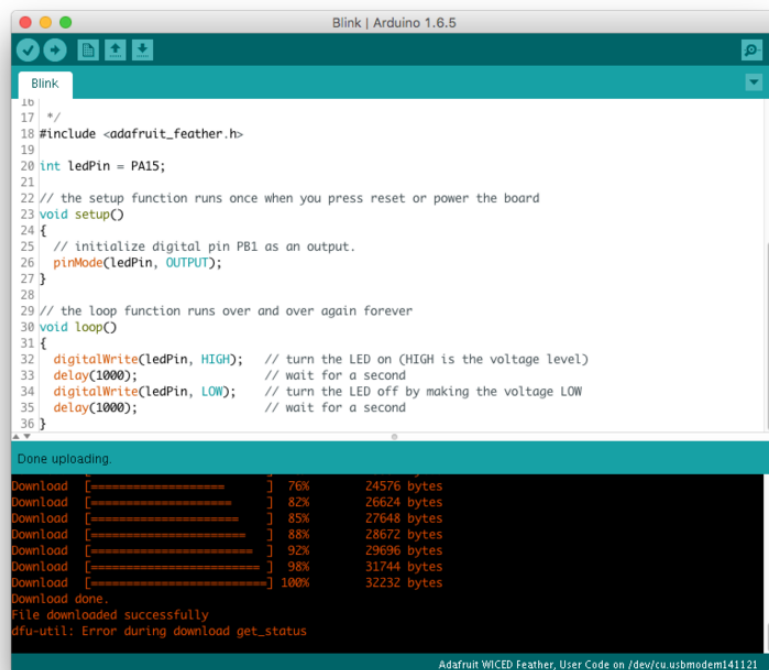
Compiling your Sketch

At this point you're ready to start flashing your projects to the WICED Feather as you would with any other Arduino compatible development board!

If you run into any problems, make sure that the WICED Feather BSP is properly configured, that you have installed the appropriate ARM Cortex M3 toolchain, and that the IDE is setup with the following values:

- **Board:** Adafruit WICED Feather
- **Section:** User Code
- **Serial Port:** Typically the numerically lowest WICED CDC port, but it should be set to the COM port that appears as '**WICED Feather Serial**' in the Device Manager on Windows where the order of enumeration may change.

Then just click the '**Upload**' arrow icon, and the compilation and USB DFU flashing process should start, which will result in the following output:



```
16
17 */
18 #include <adafruit_feather.h>
19
20 int ledPin = PA15;
21
22 // the setup function runs once when you press reset or power the board
23 void setup()
24 {
25   // initialize digital pin PB1 as an output.
26   pinMode(ledPin, OUTPUT);
27 }
28
29 // the loop function runs over and over again forever
30 void loop()
31 {
32   digitalWrite(ledPin, HIGH); // turn the LED on (HIGH is the voltage level)
33   delay(1000);                // wait for a second
34   digitalWrite(ledPin, LOW);  // turn the LED off by making the voltage LOW
35   delay(1000);                // wait for a second
36 }
```

Done uploading.

Download	[Progress bar]	76%	24576 bytes
Download	[Progress bar]	82%	26624 bytes
Download	[Progress bar]	85%	27648 bytes
Download	[Progress bar]	88%	28672 bytes
Download	[Progress bar]	92%	29696 bytes
Download	[Progress bar]	98%	31744 bytes
Download	[Progress bar]	100%	32232 bytes

Download done.
File downloaded successfully
dfu-util: Error during download get_status

Adafruit WICED Feather, User Code on /dev/cu.usbmodem141121

The 'Error during download get_status' message can be ignored and is related to the USB DFU interface as implemented on the MCU.

System Architecture

One of the key challenges creating the WICED Feather is that it is based on the Broadcom WICED WiFi stack, and due to the license terms we're unable to release any of the source files.

This poses a bit of a dilemma since we tested almost every embedded WiFi stack out there, and WICED easily climbed to the top in terms of features, performance and reliability. We want that reliability and speed, but we also want to make sure customers have the flexibility to bring all kind of projects to life as well, without having to sign restrictive license agreements themselves.

So how do we make this available to customers in a way they can use in the real world, without signing NDAs themselves?

The answer wasn't obvious at first, but **FeatherLib** was the end result of a lot of head scratching and firmware dead ends.

WICED WiFi + RTOS + SDEP = FeatherLib

The proprietary Broadcom WICED WiFi stack is designed around an RTOS (Real Time Operating System), which handles all of the various tasks involved in WiFi, such as async data requests, security and cryptography, etc. (If you're not familiar with them, an RTOS breaks tasks into 'threads', and then shares MCU cycles between those threads, allowing you to effectively multi-task on a single MCU.)

The RTOS and all of the proprietary Broadcom WiFi stack and code runs behind the scenes in a binary black box we call the **FeatherLib** (see the Flash Memory Layout section below for details). By providing a binary black box, we solve the legal hurdles of working with WICED, but this still leaves the problem of exposing the WiFi functionality to end user.

We solved this by essentially 'wrapping' every useful function in the WICED stack with a custom command (using an in house protocol called SDEP), and then routing these commands between the Broadcom SDK in FeatherLib and end users. We can freely expose the source related to the SDEP commands (since we wrote 100% of it), while still hiding the proprietary Broadcom functions, header files and structs. The lawyers are happy, and hopefully our customers are too!

By basically reimplementing the entire Broadcom WICED WiFi stack with a new set of SDEP commands and a more focused custom API, you get access to Broadcom's high quality stack, without any of the legal headaches. The headaches were all on our side reimplementing the wheel just to solve a legal problem. :)

Arduino User Code

This left the problem of how to allow users to write code themselves that talks to FeatherLib via SDEP.

Since FeatherLib runs on an RTOS, we start a single RTOS 'thread' at startup that is used for the user code. FeatherLib will start the Broadcom WiFi stack, and as part of that process it also start the 'user code' thread that runs the custom code that you write and compile in the Arduino IDE.

This custom user code is built in the Arduino IDE like you would for any other MCU, and gets written into a dedicated section of flash memory with it's own chunk of SRAM reserved purely for the user code in Arduino land.

This setup allows you to share the MCU processing time between your own code and the lower level WiFi stack, but the process should normally be invisible to you, and you never really need to worry about the FeatherLib black box.

Inter Process Communication (SDEP)

Communication between the user code and the Feather lib happens via an in-memory messaging system, sending and receiving commands using a custom protocol we call **SDEP** (Simple Data Exchange Protocol).

An SDEP command is sent to the Feather lib, and a standard response is sent back and interpreted, allowing the two binary blobs to exist entirely independent of each other, and be updated separately.

You normally never need to deal with SDEP commands directly since the commands are all hidden in the public WICED Feather helper classes (AdafruitFeather, AdafruitHTTP, etc.). These helper classes and functions send the SDEP commands for you, and convert the responses into meaningful data.

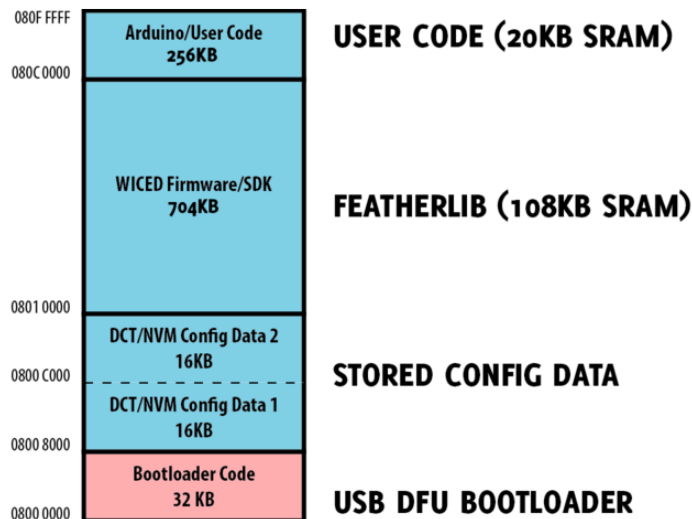
There is a special AdafruitSDEP helper class that allows you to send SDEP commands directly if the need does every arise, though, and the SDEP commands are all documented elsewhere in this learning guide.

Flash Memory Layout

To keep things as simple as possible, and to make updates easy, the flash-memory and SRAM on the STM32F205 MCU is broken up into several **Sections**, as shown in the diagram below.

Keeping the sections independent allows you to update the user code without having to recompile and reflash the rest of the system, significantly speeding up build and write times.

Earlier versions of the WICED Feather (<0.6.0) only reserved 128KB flash and 16KB SRAM for user code. If you have an older board, just update your FeatherLib and reset the board to benefit from the new 256KB flash and 20KB SRAM limit on FeatherLib 0.6.0 and higher.



User Code (256KB + 20KB SRAM)

Your own code ('User Code') will be compiled directly by the Arduino IDE, and has access to 256KB of flash and 20KB of SRAM.

Feather Lib (704 KB + 108KB SRAM)

The low level WiFi stack from Broadcom ('Feather Lib') is provided as a single pre-compiled .hex file that gets flashed to a dedicated location in flash memory on the STM32F205 MCU. Because most of the heavy lifting is done here, it has access to most of the flash and SRAM.

Config Data (32KB)

Two identical sets of non-volatile config data are stored in this section, and when any changes are made the bank used is switched to make sure that no data is lost during the updates. Normally you will never access this memory directly, and this is managed by the Feather Lib.

USB DFU Bootloader (32KB)

This code runs as soon as your device powers up and starts the Feather Lib, and also checks if any User Code is available.

This is what allows you to update the User Code or Feather Lib using USB DFU.

The bootloader code itself can be updated from the Arduino IDE as well, but it requires you have either a Segger J-Link or an STLink/V2 connected to the SWDIO and SWCLK pins on the WICED Feather, and you will normally never need to update the bootloader yourself.

USB Setup

The WICED Feather enumerates several USB classes, depending on the operating mode that the board is in.

DFU Mode (Fast Blinky)

When the WICED Feather is in DFU mode (which you can detect thanks to a fast, constant rate blinky on the LED), the following USB classes are available:

- **DFU** - Allows you to update the firmware on your board using dfu-util

When running in DFU mode the WICED Feather enumerates with the following VID/PID values:

- **VID:** 0x239A
- **PID:** 0x0008

Normal Operating Mode (User Code)

When the WICED Feather is running in normal operating mode, meaning it is running user code, three USB classes are enumerated:

- **WICED Feather Dummy:** Allows SDEP commands to be sent to the WICED Feather using the USB control endpoint (to force a reset, change operating modes, etc.). Note that this is actually just a work around to gain access to the USB control transfer endpoint with libusb since we can't access control transfers directly, ergo the name 'Dummy'.
- **Serial Monitor CDC:** This USB CDC class is used to handle Serial Monitor input and output

- **AT Parser CDC:** This (currently unused) USB CDC class enumerates for future expansion and currently exposes an AT Parser with a very limited set of commands, but may be repurposed for other uses in the future.

When running in normal operating mode, the WICED Feather will enumerate with the following VID/PID:

- **VID:** 0x239A
- **PID:** 0x0010

The WICED Feather also contains a currently unused 16 MBit (2MB) SPI Flash that will be enabled in a future firmware update. When the solder jumper on the bottom of the WICED Feather is enabled, an additional USB Mass Storage class will enumerate that points to the SPI flash. This feature is not yet enabled, but when enabled the WICED Feather will use the following VID/PID combination:

- **VID:** 0x239A
- **PID:** 0x8010

Flash Updates

All flash updates happen using **USB DFU**. There is no serial bootloader on the WICED Feather and the USB CDC ports are not required to perform a firmware update.

To perform a firmware update, the 'Enter DFU Mode' SDEP command is sent to the WICED Feather using the **WICED Feather Dummy** endpoint, which will cause the device to reset into DFU mode. At this point, dfu-util will be used to update the flash contents of the chip with the appropriate firmware image.

If FeatherLib is present, but no valid user code is available, the board will go into DFU mode by default.

Please note that if you have any errors in your user code, such as a blocking delay that the RTOS task manager can't escape from, you may see problems enumerating some USB interfaces like CDC. To resolve this problem, simply flash a valid user sketch and reset the device. USB CDC is not required to flash firmware images to the board.

WICED Feather API

In order to simplify the most common activities with the WICED Feather, several helper classes have been added to the board support package.

These helper classes are described below, and detailed explanations of each class can be found later in this guide.

AdafruitFeather

This is the main class you will use to configure the WICED Feather. It contains functions to connect or disconnect to an AP, ping another device, set certificate details when using TLS and HTTPS, as well as a few more specialized commands like some MQTT commands to use the internal MQTT stack in the WICED Feather WiFi stack.

For detailed information see: [AdafruitFeather](https://adafru.it/mfa) (<https://adafru.it/mfa>) and [AdafruitFeather: Profiles](https://adafru.it/mfb) (<https://adafru.it/mfb>)

AdafruitTCP

The AdafruitTCP class provides helper functions to open, close and work with TCP based socket connections. There are convenient callback functions for the socket disconnect events, as well as when data is received, and you can start an open or SSL based connection.

For detailed information see: [AdafruitTCP](https://adafru.it/mfc) (<https://adafru.it/mfc>) and [AdafruitTCPServer](https://adafru.it/mfd) (<https://adafru.it/mfd>)

AdafruitUDP

The AdafruitUDP class provides helper functions to open, close and work with UDP based socket connections. There is a callback function to handle data receive events.

For detailed information see: [AdafruitUDP](https://adafru.it/mfe) (<https://adafru.it/mfe>)

AdafruitHTTP

This class provides a convenient wrapper for the most common HTTP activities, including a callback for when data is received, and helpers to deal with response headers and and TLS (for secure HTTPS connections).

For detailed information see: [AdafruitHTTP](https://adafru.it/mff) (<https://adafru.it/mff>)

AdafruitMQTT

This class provides a basic MQTT client, allowing you to connect to remote MQTT brokers over a standard TCP connection. You can establish open or secure connections to the MQTT broker, publish to topics, subscribe to up to **eight** topics (including using subscribe wildcards like 'adafruit/+' to subscribe to all changes above '/adafruit'), and capture subscribe events via a convenient callback handler.

For detailed information see: [AdafruitMQTT](https://adafru.it/mfg) (<https://adafru.it/mfg>) and [AdafruitMQTTTopic](https://adafru.it/mfh) (<https://adafru.it/mfh>)

AdafruitAIO

The AdafruitAIO family is a specialized version of the AdafruitMQTT classes, and is designed to work specifically with [Adafruit IO](https://adafru.it/eIC) (<https://adafru.it/eIC>).

For detailed information see: [AdafruitAIO](https://adafru.it/mfi) (<https://adafru.it/mfi>) and [AdafruitAIOFeed](https://adafru.it/mfj) (<https://adafru.it/mfj>)

AdafruitSDEP

This class handles sending and receiving SDEP messages between the user code and the lower level feather lib. Normally you will never need to send SDEP messages yourself, and you will use the higher level helper classes mentioned elsewhere on this page, but AdafruitHTTP inherits from AdafruitSDEP, so you have access to all of the functions in AdafruitSDEP via the standard **Feather** object, such as **Feather.sdep_n(...)**, **Feather.errno()**, etc.

For detailed information see: [AdafruitSDEP](https://adafru.it/mfk) (<https://adafru.it/mfk>)

Client API

The WICED Feather board support package also includes support for the standard [Arduino Client](https://adafru.it/IFj) (<https://adafru.it/IFj>) interface, which is common to almost every networking device in the Arduino family. The Adafruit helper classes mentioned above expose many standard Client functions, and you should be able to adapt Client based example code to the WICED Feather with minimal changes and effort.

For detailed information see: Client

AdafruitFeather

AdafruitFeather is the main class that you will use for common operations like connecting to an access point (AP), checking error codes, getting your IP address, or working with stored AP profiles.

AdafruitFeather API

The following functions are available in AdafruitFeather (which is normally accessible as **Feather.*** in all of your sketches, for example '**Feather.factoryReset()**').

```
char const* bootloaderVersion ( void );
char const* sdkVersion        ( void );
char const* firmwareVersion   ( void );
char const* arduinoVersion     ( void );

int          scanNetworks      ( wl_ap_info_t ap_list[], uint8_t max_ap );

bool         connect           ( void );
bool         connect           ( const char *ssid );
bool         connect           ( const char *ssid, const char *key, int enc_type =
ENC_TYPE_AUTO );

bool         begin             ( void );
bool         begin             ( const char *ssid );
bool         begin             ( const char *ssid, const char *key, int enc_type =
ENC_TYPE_AUTO );

void         disconnect        ( void );

bool         connected         ( void );
uint8_t*     macAddress        ( uint8_t *mac );
uint32_t     localIP           ( void );
uint32_t     subnetMask       ( void );
uint32_t     gatewayIP        ( void );
char*        SSID              ( void );
int32_t      RSSI              ( void );
int32_t      encryptionType    ( void );
uint8_t*     BSSID             ( uint8_t* bssid );

IPAddress    hostByName        ( const char* hostname );
bool         hostByName        ( const char* hostname, IPAddress& result );
bool         hostByName        ( const String &hostname, IPAddress& result );

uint32_t     ping              ( char const* host );
uint32_t     ping              ( IPAddress ip );

void         factoryReset      ( void );
void         nvmReset          ( void );

bool         randomNumber      ( uint32_t* random32bit );

bool         getISO8601Time     ( iso8601_time_t* iso8601_time );
uint32_t     getUtcTime        ( void );

bool         useDefaultRootCA   ( bool enabled );
bool         initRootCA        ( void );
bool         addRootCA         ( uint8_t const* root_ca, uint16_t len);
bool         clearRootCA       ( void );

void         printVersions     ( Print& p = Serial );
```

```
void      printNetwork      ( Print& p = Serial );
void      printEncryption   ( int32_t enc, Print& p = Serial );

void setDisconnectCallback (void (*fp) (void));
```

Firmware Version Management

Since the Arduino/user code, FeatherLib binary, Broadcom WICED SDK and bootloader version need to work with each other, it's important to make sure that the version numbers of the various components of the WICED Feather are in sync.

The following helper functions are provided to retrieve the current version numbers for the the various components used by your device.

`char const* bootloaderVersion ( void )`

Returns the current bootloader version string.

Parameters: None

Returns: A null-terminated string containing the current bootloader version in the MAJOR, MINOR, REVISION format, ex: "1.0.0".

`char const* sdkVersion ( void )`

Returns the current Broadcom WICED SDK version string.

Parameters: None

Returns: A null-terminated string containing the current Broadcom WICED SDK version in the MAJOR, MINOR, REVISION format, ex: "3.5.2".

`char const* firmwareVersion ( void )`

Returns the current FeatherLib version string.

Parameters: None

Returns: A null-terminated string containing the current FeatherLib version in the MAJOR, MINOR, REVISION format, ex: "0.5.0".

`char const* arduinoVersion ( void )`

Returns the current Arduino library version string. This corresponds to the library used when building code in the Arduino IDE, which handles the low level communication to FeatherLib.

Parameters: None

Returns: A null-terminated string containing the current Arduino library version in the MAJOR, MINOR, REVISION format, ex: "0.5.0".

Scanning

The following function initiates an access point (AP) scan to determine which APs are in range of the WICED Feather.

int scanNetworks (wl_ap_info_t ap_list[], uint8_t max_ap)

Initiates a new access point scan and returns the device details for any access point(s) within range of the WICED Feather.

Parameters:

- **ap_list:** A pointer to an **wl_ap_info_t** array where the details for any AP found should be inserted. This array needs to be large enough to hold up to '**max_ap**' entries!
- **max_ap:** The maximum number of access points to write to '**ap_list**'.

Returns: The number of APs written into **ap_list**.

See the 'Constants' page in this learning guide for details on the **wl_ap_info_t** struct.

Connecting

The following functions are used to connect to an access point.

bool connect (void)

This function will attempt to connect using the list of **Profiles** stored in non-volatile config memory on the WICED Feather. See the AdafruitFeather: Profiles page in this learning guide for details on how to use the profile system.

Parameters: None

Returns: 'True' (1) if a connection was established with an AP based on the stored profile data, otherwise 'false' (0).

bool connect (const char *ssid)

Attempts to connect to the **open** (security type = **ENY_TYPE_OPEN**) access point matching the 'ssid' parameter.

Parameters:

- **ssid**: A string containing the name of the SSID to attempt to connect to.

Returns: 'True' (1) if the connection was successful, otherwise 'false' (0).

bool connect (const char *ssid, const char *key, int enc_type = ENC_TYPE_AUTO)

Attempts to connect to the specified SSID using the supplied password ('key') and optionally a specific security type ('enc_type').

The security type is optional and if no value is provided the WICED Feather will attempt to determine the security type on it's own, but the connection process will terminate more quickly if you provide the appropriate security type since this avoids the need to perform a full access point scan before the connection attempt starts.

Parameters:

- **ssid**: A string containing the name of the SSID to attempt to connect to.
- **key**: The password to use when connecting to the AP
- **enc_type**: The **wl_enc_type_t** value that indicates what type of security is used by the AP. The default value for this field is **ENC_TYPE_AUTO** which will cause the WICED Feather to determine this information for you, at the expense of a slower connection interval since we first have to perform a full access point scan. See the **Constants** page in this learning guide for a list of possible values for this parameter.

Returns: 'True' (1) if the connection was successful, otherwise 'false' (0).

bool begin (void)

This is an alias for ' **bool connect(void)** ' described above, and is provided to match the Arduino Client interface.

bool begin (const char *ssid)

This is an alias for ' **bool connect(const char* ssid)** ' described above, and is provided to match the Arduino Client interface.

```
bool begin ( const char *ssid, const char *key, int enc_type
= ENC_TYPE_AUTO )
```

This is an alias for ' `bool connect(const char *ssid, const char *key, int enc_type)` ' described above, and is provided to match the Arduino Client interface.

```
void disconnect (void)
```

Disconnects from the current access point.

Parameters: None

Returns: Nothing

Network and Connection Details

The following functions provide information about the connection or the network setup when your device is connected to an access point (AP).

```
bool connected ( void );
```

Checks if you are currently connected to an AP or not.

Parameters: None

Returns: 'True' (1) if you are currently connected to an access point (AP), otherwise 'false' (0).

```
uint8_t* macAddress ( uint8_t *mac );
```

Gets the HW mac address for the WICED Feather.

Parameters:

- **mac:** The 6-byte uint8_t array to assign the mac address to. If you don't wish to use this field and use the optional 'return value' instead simply provide NULL to this parameter.

Returns: A pointer to a 6-byte array containing the 48-bit HW MAC address for your WICED Feather.

```
uint32_t localIP ( void );
```

Returns the IPv4 address for your WICED Feather.

Parameters: None

Returns: A 32-bit integer containing the four bytes that make up the IPv4 address for your device.

uint32_t subnetMask (void);

Returns the IPv4 subnet mask.

Parameters: None

Returns: A 32-bit integer containing the four bytes that make up the IPv4 subnet mask.

uint32_t gatewayIP (void);

Returns the IPv4 gateway IP.

Parameters: None

Returns: A 32-bit integer containing the four bytes that make up the IPv4 gateway address.

char* SSID (void);

Returns the SSID for the current access point (AP).

Parameters: None

Returns: A null-terminated string containing the SSID name for the current AP.

int32_t RSSI (void);

Returns the current return signal strength indicator (RSSI) in dBm, which indicate the strength of the connection between the WICED Feather and the remote access point. The larger the number, the strong the signal is (ex. -90dBm is weaker than -65dBm).

Parameters: None

Returns: The return signal strength indicated in dBm.

int32_t encryptionType (void);

Returns the current encryption type used by the AP. See **wl_enc_type_t** on the **constants** page for a list of possible encryption types.

Parameters: None

Returns: An integer corresponding to the **wl_enc_type_t** enum list described on the **constants** page in this learning guide.

`uint8_t* BSSID ( uint8_t* bssid );`

Gets the access point mac address for the remote AP used by the WICED Feather.

Parameters:

- **bssid:** The 6-byte `uint8_t` array to assign the BSSID address to. If you don't wish to use this field and use the optional 'return value' instead simply provide `NULL` to this parameter.

Returns: A pointer to a 6-byte array containing the 48-bit MAC address for the access point your WICED Feather is connected to.

DNS Lookup

The following helper functions allow you to look up a host name on the DNS server, converting it to an IP address:

`IPAddress hostByName ( const char* hostname )`

Parameters:

- **hostname:** A string representing the domain name to lookup (ex. "www.adafruit.com").

Returns: The [IPAddress](https://adafru.it/IGd) (<https://adafru.it/IGd>) corresponding to the specified hostname.

`bool hostByName ( const char* hostname, IPAddress& result )`

Looks up the domain name specified in the 'hostname' string, and assigns it to the [IPAddress](https://adafru.it/IGd) (<https://adafru.it/IGd>) referenced by the 'result'.

Parameters:

- **hostname:** A string representing the domain name to lookup (ex. "www.adafruit.com").
- **result:** the [IPAddress](https://adafru.it/IGd) (<https://adafru.it/IGd>) object that the lookup results will be assigned to.

Returns: 'True' (1) if the DNS lookup was successful, otherwise 'false' (0).

bool hostByName (const String &hostname, IPAddress& result)

Looks up the domain name specified in the 'hostname' string, and assigns it to the [IPAddress](https://adafru.it/IGd) (<https://adafru.it/IGd>) referenced by the 'result'.

Parameters:

- **hostname:** A string representing the domain name to lookup (ex. "www.adafruit.com").
- **result:** the [IPAddress](https://adafru.it/IGd) (<https://adafru.it/IGd>) object that the lookup results will be assigned to.

Returns: 'True' (1) if the DNS lookup was successful, otherwise 'false' (0).

Ping

Ping can be used to detect if another server or device is available (although not all devices respond to ping requests!). The following helpers are available for this purpose:

uint32_t ping (char const* host)

Pings the domain name specified in the 'host' string.

Parameters:

- **host:** The domain name to ping (ex. "www.adafruit.com").

Returns: The response time in milliseconds if the domain responded to the ping request, or '0' if the ping failed.

uint32_t ping (IPAddress ip)

Pings the specified [IPAddress](https://adafru.it/IGd) (<https://adafru.it/IGd>).

Parameters:

- **ip:** The [IPAddress](https://adafru.it/IGd) (<https://adafru.it/IGd>) to ping.

Returns: The response time in milliseconds if the IP address responded to the ping request, or '0' if the ping failed.

Factory Reset

If you set your WICED Feather modules into an unknown state of encounter unexpected behaviour, you can try to perform a full factory reset or reset the non-volatile config memory using these helper functions.

`void factoryReset ( void )`

Performs a full factory reset on the module, with the following consequences:

- Erases all config data in non-volatile memory (NVM)
- Erases any user code ('Arduino' code) from flash memory
- Resets the device to the same state as when it shipped from the factory (although the current FeatherLib will be kept intact)
- Performs a system reset, which will send the device into DFU mode since no user code is present on the device.

Parameters: None

Returns: Nothing

`void nvmReset ( void )`

Erases the non-volatile config memory on the WICED module, resetting the config settings to factory defaults.

Parameters: None

Returns: Nothing

Hardware Random Number Generator

The STM32F205 includes a HW white-noise random number generator that provides better results than a purely software based approach.

This can be used to generate random strings or numeric values for security purposes.

`bool randomNumber ( uint32_t* random32bit )`

Assigns a random unsigned 32-bit integer value to 'random32bit'.

Parameters:

- **random32bit:** A pointer to the variable where the random number should be assigned

Returns: 'True' (1) if the random number generation was successful, otherwise 'false' (0).

Real Time Clock

The STM32F205 includes a real time clock, and as soon as you connect to an AP with internet access it will try to update the RTC clock based on an NTP server.

The RTC can be read in both [Linux Epoch \(UTC\)](https://adafru.it/INA) (<https://adafru.it/INA>) time or [ISO8601](https://adafru.it/INB) (<https://adafru.it/INB>) format.

Epoch time returns a 32-bit unsigned integer value representing the number of seconds since 1 January 1970. For example '1456472597' would convert to:

Fri, 26 Feb 2016 07:43:17 GMT

ISO8601 format timestamps return the time as a specifically formatted string similar to the timestamp below:

2016-02-18T17:12:46.061104

bool getISO8601Time (iso8601_time_t* iso8601_time)

Updates 'iso8601_time' with the current timestamp in ISO8601 format.

Time is based on **GMT** and will need to be adjusted for your local timezone, depending on your specific location and any seasonal adjustments (daylight savings time, etc.).

Parameters:

- **iso8601_time:** A pointer to the 'iso8601_time_t' struct that will hold the timestamp data. (The typedef itself is defined in adafruit_constants.h.)

Returns: 'True' (1) if the timestamp was successfully assigned, otherwise 'false' (0).

ISO8601 timestamps use the following struct (defined in 'adafruit_constants.h') to convert the timestamp into something that can be printed as a null-terminated string, but also easily manipulated in code:

```
typedef struct ATTR_PACKED
{
    char year[4];           /**&lt; Year          */
    char dash1;             /**&lt; Dash1         */
    char month[2];          /**&lt; Month         */
    char dash2;             /**&lt; Dash2         */
    char day[2];            /**&lt; Day          */
    char T;                 /**&lt; T            */
    char hour[2];           /**&lt; Hour         */
}
```

```

char colon1;          /**&lt; Colon1      */
char minute[2];       /**&lt; Minute     */
char colon2;          /**&lt; Colon2      */
char second[2];       /**&lt; Second      */
char decimal;         /**&lt; Decimal     */
char sub_second[6];   /**&lt; Sub-second */
char Z;               /**&lt; UTC timezone */

char nullterm;        // not part of the format, make printf easier
} iso8601_time_t;

```

uint32_t getUtcTime (void)

Returns the current 'Epoch' time (the number of seconds since the 1 January 1970).

Time is based on **GMT** and will need to be adjusted for your local timezone, depending on your specific location and any seasonal adjustments (daylight savings time, etc.).

Parameters: None

Returns: A 32-bit unsigned integer representing the number of seconds since the 'Epoch', or 1 January 1970.

TLS Root Certificate Management

Connecting to secure TLS/SSL based servers requires a root certificate to verify that the certificate data from the remote server is valid. A set of common root certificates is included in the Featherlib by default, but custom certificates can also be added to the chain via the **.addRootCA** helper function, described below.

See the AdafruitTCP documentation for more information on TLS and connecting to secure servers.

Default Root Certificates

By default, the following root certificates are included in Featherlib, meaning you only need to add a root certificate authority if it isn't included in the list below.

These default root certificates cover many common websites without any additional effort on your part:

- **Baltimore CyberTrust Root**

- adafruit-download.s3.amazonaws.com (may include other Amazon S3 servers)

- **DigiCert High Assurance EV Root CA**

- twitter.com

- facebook.com
- github.com
- **GeoTrust Global CA**
 - google.com
- **GeoTrust Primary Certification Authority - G3**
 - adafruit.com
- **Starfield Services Root Certificate Authority - G2**
 - aws.amazon.com

bool useDefaultRootCA (bool enabled)

Enables the default list of root CAs in FeatherLib.

Note: These will be enabled automatically by default if you try to use the **.connectSSL** functions without having previously added any custom root CAs via **.addRootCA**. This function is provided primarily to disable the default root certificates since they consume a reasonable chunk of heap memory.

Parameters:

- **enabled:** Set this to 'true' (1) to enable the default root CA list, or 'false' (0) to disable them.

Returns: 'True' (1) if the operation succeeded, otherwise 'false' (0).

The default root CA list will be enabled by default unless Feather.enableRootCA(false) or Feather.clearRootCA() is called explicitly.

bool initRootCA (void)

This function allocates memory for the default list of root certificates and any custom root certificates present.

Normally this function never needs to be called directly, and will be call on an as-needed bases by **.addRootCA** or **.connectSSL**. It is provided as a public function so that other classes can have access to it (AdafruitTCP, etc.).

Parameters: None

Returns: 'True' (1) if the root CA initialisation was successful, otherwise 'false' (0).

bool addRootCA (uint8_t const* root_ca, uint16_t len)

This will add the supplied root certificate to the default root certificate list. The combined root list (default plus custom root CAs) will be used when trying to verify any certificate chains provided by a remote secure server.

The root certificate chain supplied via 'root_ca' can contain more than one certificate, but must be a byte array converted from a binary .der file, generated using the python tool included in the '/tools/pycert' folder of the board support package.

Parameters:

- **root_ca:** A pointer to the .der file byte array generated by `/tools/pycert/pycert.py`
- **len:** The size in bytes of the .der byte array

Returns: 'True' (1) if the root certificate chain was successfully set, otherwise 'false' (0).

bool clearRootCA (void)

Clears any root certificates currently used by the system (freeing up associated heap memory in FeatherLib).

Parameters: None

Returns: 'True' (1) if the operation succeeded, otherwise 'false' (0).

Print Helpers

The following functions are provided to print out common data and simplify user sketches:

void printVersions (Print& p = Serial)

Displays the bootloader and firmware versions used by the WICED Feather in the following order:

- Bootloader version
- Broadcom WICED SDK version
- FeatherLib version
- Arduino (User Code) version

Parameters:

- **p**: The 'Print' implementation to use. Leave this field empty and it will default to 'Serial' which is used for the Serial Monitor output.

Returns: Nothing.

void printNetwork (Print& p = Serial)

Displays the following network details when connected to an AP:

- SSID Name
- SSID Encryption Method
- MAC Address
- Local IP Address
- Gateway Address
- Subnet Mask

Parameters:

- **p**: The 'Print' implementation to use. Leave this field empty and it will default to 'Serial' which is used for the Serial Monitor output.

Returns: Nothing.

void printEncryption (int32_t enc, Print& p = Serial)

Displays a string that corresponds to the specified encryption type (see .encryptionType elsewhere in this class):

Parameters:

- **enc**: The security encryption type used by the AP
- **p**: The 'Print' implementation to us. Leave this field empty and it will default to 'Serial' which is used for the Serial Monitor output.

Returns: Nothing

AdafruitFeather: Profiles

The WICED Feather API allows you to store 'profiles', which contain all of the settings about a specific AP (access point).

This means that you only need to enter your AP details once, and once connected you can store them in non-volatile config memory for later use, simplifying project management and speeding up connection time in certain instances.

This is useful in situations where your project might move from one physical location to another, and the AP will change between locations (for example at home and at the office).

Up to FIVE profiles can be stored at a time in non-volatile memory.

Connecting via Profiles

To connect to an AP using the stored profile data, simply call the bare **Feather.connect()** function with no parameters. This will attempt to connect to the profiles stored in non-volatile memory from the first entry to the last, and will return false if we were unable to connect to any of the stored APs.

Profiles API

The profile management API includes the following functions (defined as part of the **AdafruitFeather** class which is normally available as **Feather.***):

```
bool    saveConnectedProfile ( void );           // Save currently connected AP
bool    addProfile           ( char* ssid );    // Open
bool    addProfile           ( char* ssid, char* key, wl_enc_type_t enc_type);
bool    removeProfile        ( char* ssid );
bool    checkProfile         ( char* ssid );    // Check if profile exists
void    clearProfiles        ( void );
char*    profileSSID         ( uint8_t pos);
int32_t  profileEncryptionType ( uint8_t pos);
```

bool saveConnectedProfile (void)

Saves the currently connected access point details as a Profile. You must be connected when calling this functions.

Parameters: None

Returns: 'true' (1) if the profile was successfully added, otherwise 'false' (0).

bool addProfile (char* ssid)

Saves the specified **open** SSID to the profile list. This function should only be used with open access points that have no security/encoding enabled.

Parameters:

- **ssid**: A string containing the access point's SSID/name.

Returns: 'true' (1) if the profile was successfully added, otherwise 'false' (0).

bool addProfile (char* ssid, char* key, wl_enc_type_t enc_type)

Saves the specified **secure** SSID to the profile list. This function should not be used with open access points.

Parameters:

- **ssid**: A string containing the access point's SSID/name.
- **key**: A string containing the pass key for the SSID
- **enc_type**: The security encoding type for the access point, which can be one of the following values:

Encoding Types (wl_enc_type_t):

- **ENC_TYPE_WEP**
WEP security with open authentication
- **ENC_TYPE_WEP_SHARED**
WEP security with shared authentication
- **ENC_TYPE_WPA_TKIP**
WPA security with TKIP
- **ENC_TYPE_WPA_AES**
WPA security with AES
- **ENC_TYPE_WPA_MIXED**
WPA security with AES and TKIP
- **ENC_TYPE_WPA2_TKIP**
WPA2 security with TKIP
- **ENC_TYPE_WPA2_AES**
WPA2 security with AES
- **ENC_TYPE_WPA2_MIXED**
WPA2 security with TKIP and AES

- **ENC_TYPE_WPA_TKIP_ENT**
WPA enterprise security with TKIP
- **ENC_TYPE_WPA_AES_ENT**
WPA enterprise security with AES
- **ENC_TYPE_WPA_MIXED_ENT**
WPA enterprise security with TKIP and AES
- **ENC_TYPE_WPA2_TKIP_ENT**
WPA2 enterprise security with TKIP
- **ENC_TYPE_WPA2_AES_ENT**
WPA2 enterprise security with AES
- **ENC_TYPE_WPA2_MIXED_ENT**
WPA2 enterprise security with TKIP and AES
- **ENC_TYPE_WPS_OPEN**
WPS with open security
- **ENC_TYPE_WPS_SECURE**
WPS with AES security
- **ENC_TYPE_IBSS_OPEN**
BSS with open security

Returns: 'true' (1) if the profile was successfully added, otherwise 'false' (0).

bool removeProfile (char* ssid)

Removes the profile with the matching ssid from non-volatile memory.

Parameters:

- **ssid:** A string containing the access point's SSID/name.

Returns: 'true' (1) if the profile was successfully found and removed, otherwise 'false' (0).

void clearProfiles (void)

Clears all profiles from non-volatile memory.

Parameters: None

Returns: Nothing

char* profileSSID (uint8_t pos);

Returns a string containing the SSID name for the profile stored at the specified position.

Parameters:

- **pos:** The position in NVM for the profile, which can be a value between 0 and 4 (since the position is a zero-based integer).

Returns: NULL if no profile was found at the specified 'pos', otherwise a string corresponding to SSID name for the stored profile.

int32_t profileEncryptionType (uint8_t pos);

Returns the `wl\_enc\_type\_t` value for the profile stored at the specified position.

Parameters:

- **pos:** The position in NVM for the profile, which can be a value between 0 and 4 (since the position is a zero-based integer).

Returns: '-1' if no profile was found at the specified 'pos', otherwise an integer corresponding to one of the entries in 'wl_enc_type_t' (see the list of options in **addProfile** above).

AdafruitTCP

AdafruitTCP makes it easier to work with raw TCP sockets. You can open sockets -- including SSL based secure socket connections -- and send and receive data using a few basic commands.

The class also and exposes two convenient (optional) callbacks:

- **Data Received Callback:** Fires whenever incoming data is available (which can then be read via the `.read()` and related commands)
- **Disconnect Callback:** Fires whenever the TCP server cause you to disconnect

You're also free to 'poll' for incoming data and connection status, but these callbacks help keep your TCP code easy to understand and more maintainable as your project grows in complexity.

TCP Socket API

The AdafruitTCP class includes the following functions:

```
// Misc Functions
void      usePacketBuffering      (bool enable);
void      tlsRequireVerification (bool required);
```

```

uint32_t getHandle          (void);

// Client API
virtual int    connect      (IPAddress ip, uint16_t port);
virtual int    connect      (const char * host, uint16_t port);
virtual int    connectSSL   (IPAddress ip, uint16_t port);
virtual int    connectSSL   (const char* host, uint16_t port);
virtual uint8_t connected   (void);
virtual void    stop        (void);

// Stream API
virtual int    read         (void);
virtual int    read         (uint8_t * buf, size_t size);
virtual size_t write        (uint8_t);
virtual size_t write        (const uint8_t *content, size_t len);
virtual int    available    (void);
virtual int    peek         (void);
virtual void    flush       (void);

// Set callback handlers
void setReceivedCallback    (tcpcallback_t fp);
void setDisconnectCallback  (tcpcallback_t fp);

```

Packet Buffering

The AdafruitTCP class includes the option to enable or disable **packet buffering**.

If packet buffering is **enabled**, outgoing data will be buffered until the buffer is full (~1500 bytes) or until **.flush()** is called to manually force the buffered data to be sent.

If packet buffering is **disabled**, any write commands will send the data immediately, regardless of the packet or data size. This ensure writes happen right away, but at the cost of slower overall throughput since data can't be grouped together into larger packets.

By default packet buffering is DISABLED in AdafruitTCP

void usePacketBuffering (bool enable)

This will enable or disable packet buffering with AdafruitTCP data.

Parameters:

- **enable**: Set this to 'true' (1) to enable packet buffering, otherwise 'false' (0)

Returns: Nothing

TLS/SSL Certificate Verification

When opening a secure TCP connection to a TCP server, the client and server will begin to communicate with each other in an open connection to choose their cipher suite (AES, etc.), and the server will then send the client its certificate and public key data to start the secure connection.

Normally at this point, the client will **verify the server's certificate** using its root certificate chains. If verification is OK, the connection will continue, otherwise the connection will be rejected since the server has probably provided a false or invalid certificate and can't be trusted.

The problem with this approach on small embedded systems is that it takes a great deal of space (in embedded terms) to store all root certificate chains to verify server certificates against all certificate issuing authorities. We do store a default list of the most common root certificate chains, but it isn't possible on a small MCU with limited flash storage space to store every possible root certificate option.

The WICED Feather proposes two solutions to this problem, depending on if you prefer a more secure or a simpler solution:

Verifying Certificates with the WICED Feather (Safer)

Instead of storing all root certificates, the WICED Feather allows you to generate a certificate chain for a specific domain, and then use that in your sketch, which typically requires 1-4KB of flash memory or less per domain.

This is the most secure choice but requires some additional work on your part, and you have to know in advance which sites you will access.

The procedure to convert, load and use a custom root certificate list is as follows:

1. You use a python script (provided in the '/tools/pycert' folder) to read the root certificate data for your target domain. The script then converts the binary .der format data into a byte array in a C header (.h) file.
2. You then pass the root certificate data into the WICED API via **Feather.addRootCA** (from AdafruitFeather), which allows you to add your root certificate chain to the list of default certificates used when verifying the target domain
3. You can then enable certificate verification via **tlsRequireVerification(true)** in this class, which means that all server certificates must pass verification against the root certificate list on the WICED Feather or the certificate and connection will be rejected.

Ignoring Certificate Verification (Easier)

If you aren't able to store the certificate data for a specific site, or don't know which sites you will access, you can also **ignore the verification process** which has the effect of **accepting every certificate as valid**.

This still allows for an encrypted connection (using AES, etc.), but there is no guarantee that the server you are talking to is actually the server you think you're talking to, making it a less secure option.

By default certificate verification is enabled on WICED Feather boards. You can disable verification via `'tlsRequireVerification(false)'`, which will cause any certificate to be accepted, but it will also allow man-in-the-middle type attacks.

The approach you take will depend on your project requirements, but in either case you can indicate to the WICED Feather API whether you want to verify server certificates via the following function:

Default Root Certificates

By default, the following root certificates are included in Featherlib, meaning you only need to add a root certificate authority if it isn't included in the list below.

These default root certificates cover many common websites without any additional effort on your part:

- **Baltimore CyberTrust Root**
 - `adafruit-download.s3.amazonaws.com` (may include other Amazon S3 servers)
- **DigiCert High Assurance EV Root CA**
 - `twitter.com`
 - `facebook.com`
 - `github.com`
- **GeoTrust Global CA**
 - `google.com`
- **GeoTrust Primary Certification Authority - G3**
 - `adafruit.com`

- **Starfield Services Root Certificate Authority - G2**

- aws.amazon.com

void tlsRequireVerification (bool required)

Indicates whether the certificate data provided by the remote server should be verified against the local root certificate list or not. (Note: you can add new records to the root certificate list is set in the AdafruitFeather class via '**Feather.addRootCA**').

Parameters:

- **required:** Set this to 'true' (1) if certificate validation is required, or 'false' (0) if no verification is required (meaning that every certificate provided by a remote server will be considered valid!).

Returns: Nothing

Socket Handler Functions

In specialised cases (mostly when implementing sub-classes of AdafruitTCP) you may need access to the 'handle' for the TCP socket. The .getHandle function provides access to this.

void getHandle (void)

Returns the internal TCP socket handler value that uniquely identifies this TCP socket. This might be necessary when creating special sub-classes based on AdafruitTCP.

Parameters: None

Returns: The uint32_t socket handler value that uniquely identifies this TCP socket.

Client API

The [Client API \(https://adafru.it/IFj\)](https://adafru.it/IFj) includes the following functions to connect to a TCP server:

int connect (IPAddress ip, uint16_t port)

Attempts to connect to the specified IP address and port

Parameters:

- **ip:** The [IPAddress \(https://adafru.it/IGd\)](https://adafru.it/IGd) where the TCP server is located

- **port:** The port number to connect to (0..65535)

Returns: 'true' (1) if the connection was successfully established, otherwise 'false' (0).

int connect (const char * host, uint16_t port)

Attempts to connect to the specified domain name and port

Parameters:

- **host:** A string containing the domain name to connect to
- **port:** The port number to connect to (0..65536)

Returns: 'true' (1) if the connection was successfully established, otherwise 'false' (0).

int connectSSL (IPAddress ip, uint16_t port)

Connects to a secure server using SSL/TLS at the specified IP address and port.

Parameters:

- **ip:** The [IPAddress](https://adafru.it/IGd) (<https://adafru.it/IGd>) where the TCP server is located
- **port:** The port number to connect to (0..65536)

Returns: 'true' (1) if the connection was successfully established, otherwise 'false' (0).

If certificate verification fails when trying to connect to a secure server you will get `ERROR_TLS_UNTRUSTED_CERTIFICATE` (5035).

Note: A set of common root certificates are already included in the WICED Feather SDK, so most HTTPS websites will work out of the box, but if you need to add a new root certificate chain the TLS/certificate data is set using the following function in the Adafruit Feather class (accessible as `Feather.addRootCA(...)`):

```
bool addRootCA(uint8_t const* root_certs_der, uint16_t len);
```

int connectSSL (const char* host, uint16_t port)

Attempts to connect to a secure server using SSL/TLS at the specified domain name and port.

Parameters:

- **host:** A string containing the domain name to connect to
- **port:** The port number to connect to (0..65536)

Returns: 'true' (1) if the connection was successfully established, otherwise 'false' (0).

If certificate verification fails when trying to connect to a secure server you will get `ERROR_TLS_UNTRUSTED_CERTIFICATE` (5035).

Note: A set of common root certificates are already included in the WICED Feather SDK, so most HTTPS websites will work out of the box, but if you need to add a new root certificate chain the TLS/certificate data is set using the following function in the Adafruit Feather class (accessible as `Feather.addRootCA(...)`):

```
bool addRootCA(uint8_t const* root_certs_der, uint16_t len);
```

uint8_t connected (void)

Indicates whether we are currently connected to the TCP server or not

Parameters: None

Returns: 'true' (1) if we are currently connected to the TCP server, otherwise 'false' (0).

void stop (void)

Closes the current connection to the TCP server (if a connection is open).

Parameters: None

Returns: Nothing

Stream API

AdafruitTCP implements the [Stream](https://adafru.it/lGe) (<https://adafru.it/lGe>) class, with the following method overrides present in AdafruitTCP:

int read (void)

Reads the first available byte from the data buffer (if any data is available).

Parameters: None

Returns: The first byte of incoming data available, or -1 if no data is available.

`int read (uint8_t * buf, size_t size)`

Reads up to the specified number of bytes from the data buffer (if any data is available).

Parameters:

- **buf:** A pointer to the buffer where data should be written if any data is available
- **size:** The maximum number of bytes to read and copy into **buf**.

Returns: The actual number of bytes read back, and written in **buf**.

`size_t write (uint8_t data)`

Transmits a single byte to the TCP Server (or into the outgoing buffer until it can be sent if buffering is enabled).

Parameters:

- **data:** The byte of data to transmit

Returns: The number of bytes written. It is normally not necessary to read this value.

`size_t write (const uint8_t *content, size_t len)`

Transmits a number of bytes to the TCP Server (or into the outgoing buffer until the data can be sent if buffering is enabled).

Parameters:

- **content:** A pointer to the buffer containing the data to send
- **len:** The number of bytes contained in **content**

Returns: The number of bytes successfully written.

`int available (void)`

Checks the number of bytes available in the incoming data buffer.

Parameters: None

Returns: The number of bytes available in the incoming data buffer, or 0 if no data is available.

int peek (void)

Reads the first available byte from the incoming data buffer without removing it from the buffer.

Parameters: None

Returns: The value of the first available byte, or -1 if no data is available.

void flush (void)

Forces any buffered data to be transmitted to the TCP server, regardless of the size of the content.

Parameters: None

Returns: Nothing

Callback API

To make working with TCP sockets easier, a simple callback API is available in AdafruitTCP based on the following functions:

void setReceivedCallback (tcpcallback_t fp)

Registers the data received callback handler.

Parameters:

- **fp:** The name of the function that will be executed when received data is available from the TCP server. See the example below for details on the function signature.

Returns: Nothing

void setDisconnectCallback (tcpcallback_t fp)

Registers the disconnect callback handler (fired when you are disconnected from the TCP server).

Parameters:

- **fp:** The name of the function that will be executed when you are disconnected from the TCP server. See the example below for details on the function signature.

Returns: Nothing

Callback Function Signatures

The data received and disconnect callbacks both require a specific function definition to work. The function names ('receive_callback' and 'disconnect_callback') can change, but the exact signatures are shown below:

```
void receive_callback    ( void );
void disconnect_callback ( void );
```

You then register the callbacks with the dedicated set callback functions:

Make sure you register the callbacks BEFORE calling the .connect function!

```
// Set the callback handlers for RX and disconnect
tcp.setReceivedCallback(receive_callback);
tcp.setDisconnectCallback(disconnect_callback);
```

To read incoming data in the receive callback handler, you need to use the **pTCP** pointer, as shown in the sample code below:

```
void receive_callback(void)
{
    int c;

    // Print out any bytes available from the TCP server
    while ( (c = tcp.read()) > 0 )
    {
        Serial.write( (isprint(c) || iscntrl(c)) ? ((char)c) : '.');
    }
}

void disconnect_callback(void)
{
    Serial.println();
    Serial.println("-----");
    Serial.println("DISCONNECT CALLBACK");
    Serial.println("-----");
    Serial.println();
}
```

Example: Callback Based HTTP Request

The following example shows how you can register and use the two TCP callbacks, and performs a simple TCP operation. It opens a TCP socket to an HTTP server using port 80, requests a page, displays any incoming response data, and then waits for the

HTTP server to close the TCP connection (which will show up as a disconnect callback):

```
#include <adafruit_feather.h>;

#define WLAN_SSID      "SSID"
#define WLAN_PASS      "PASSWORD"
#define WLAN_SECURITY  ENC_TYPE_AUTO

#define TCP_DOMAIN     "www.adafruit.com"
#define TCP_FILENAME   "/testwifi/index.html"
#define TCP_PORT       80

void receive_callback ( void );
void disconnect_callback ( void );

AdafruitTCP tcp;

void setup()
{
  Serial.begin(115200);

  // Wait for Serial port to connect. Needed for native USB port only
  while (!Serial) { delay(1); }

  // Attempt to connect to the AP using the specified SSID/key/encoding
  if ( !Feather.connect(WLAN_SSID, WLAN_PASS, WLAN_SECURITY ) )
  {
    err_t err = Feather.errno();
    Serial.println("Connection Error:");
    switch (err)
    {
      case ERROR_WWD_ACCESS_POINT_NOT_FOUND:
        // SSID wasn't found when scanning for APs
        Serial.println("Invalid SSID");
        break;
      case ERROR_WWD_INVALID_KEY:
        // Invalid SSID passkey
        Serial.println("Invalid Password");
        break;
      default:
        // The most likely cause of errors at this point is that
        // you are just out of the device/AP operating range
        Serial.print(err);
        Serial.print(":");
        Serial.println(Feather.errstr());
        break;
    }
    // Wait around here forever!
    while(1);
  }

  // Optional: Disable TLS certificate verification (accept any server)
  Feather.tlsRequireVerification(false);

  // Optional: Set the default TCP timeout to 10s
  tcp.setTimeout(10000);

  // Set the callback handlers for RX and disconnect
  tcp.setReceivedCallback(receive_callback);
  tcp.setDisconnectCallback(disconnect_callback);

  // Try to connect to the HTTP Server
  if ( tcp.connect(TCP_DOMAIN, TCP_PORT) )
  {
    Serial.println("Connected to server");
    // Make a basic HTTP request
```

```

    tcp.printf("GET %s HTTP/1.1\r\n", TCP_FILENAME);
    tcp.printf("host: %s\r\n", TCP_DOMAIN);
    tcp.println();
}
else
{
    Serial.printf("TCP connection failed: %s (%d)", tcp.errstr(), tcp.errno());
    Serial.println();
    while(1);
}
}

void loop()
{
    // put your main code here, to run repeatedly:
}

void receive_callback(void)
{
    int c;

    // Print out any bytes available from the TCP server
    while ( (c = tcp.read()) > 0 )
    {
        Serial.write( (isprint(c) || iscntrl(c)) ? ((char)c) : '.');
    }
}

void disconnect_callback(void)
{
    Serial.println();
    Serial.println("-----");
    Serial.println("DISCONNECT CALLBACK");
    Serial.println("-----");
    Serial.println();
}

```

AdafruitTCPServer

This class allows you to create a simple TCP based server to communicate with other TCP clients.

This class is still a work in progress and may undergo significant changes in a future version of the WICED Feather library. It should be considered experimental for now.

Constructor

AdafruitTCPServer has the following constructor:

```
AdafruitTCPServer(uint16_t port)
```

Parameters:

- **port:** The port to use for the TCP server (1..65535)

Functions

The following public functions are defined as part of the class:

```
bool      begin      ( void )
AdafruitTCP accept    ( void )
AdafruitTCP available ( void )
void      stop       ( void )

void setConnectCallback ( tcpserver_callback_t fp )
```

bool begin (void)

Starts the TCP server and begins listening for connections.

Parameters: None

Returns: 'True' (1) if the operation was successful, otherwise 'false' (0).

AdafruitTCP accept (void)

Accepts a new connection with a Client, returning an instance of the AdafruitTCP class to handle the client details.

Parameters: None

Returns: An instance of the AdafruitTCP class that can be used to deal with the client reads and writes.

AdafruitTCP available (void)

This function is an alias for the .accept function described above.

void stop (void)

Stops the TCP server.

Parameters: None

Returns: Nothing.

void setConnectCallback (tcpserver_callback_t fp)

Sets the TCP server callback event handler function for any incoming connection requests.

Parameters:

- **fp:** The function that will be used to handling incoming connection requests.

Returns: Nothing.

The connect callback function handler has the following syntax:

```
/******  
/*!  
    @brief  This callback is fired when there is a connection request from  
            a TCP client. Use accept() to establish the connection and  
            retrieve the client 'AdafruitTCP' instance.  
*/  
/******  
void connect_request_callback(void)  
{  
    uint8_t buffer[256];  
    uint16_t len;  
  
    AdafruitTCP client = tcpserver.available();  
  
    if ( client )  
    {  
        // read data  
        len = client.read(buffer, 256);  
  
        // Echo data back to the TCP client  
        client.write(buffer, len);  
  
        // call stop() to free memory in the client class  
        client.stop();  
    }  
}
```

Example

The following example will listen for connection requests on port 80 and echo back any data that is received. The connection logic happens inside the connection request callback handler.

```
#include <adafruit_feather.h>  
  
#define WLAN_SSID      "yourSSID"  
#define WLAN_PASS      "yourPass"  
  
#define PORT            80                // The TCP port to use  
  
AdafruitTCPServer tcpserver(PORT);  
  
/******  
/*!  
    @brief  This callback is fired when there is a connection request from  
            a TCP client. Use accept() to establish the connection and  
            retrieve the client 'AdafruitTCP' instance.  
*/  
/******  
void connect_request_callback(void)  
{
```

```

uint8_t buffer[256];
uint16_t len;

AdafruitTCP client = tcpserver.available();

if ( client )
{
    // read data
    len = client.read(buffer, 256);

    // Print data along with peer's info
    Serial.print("[RX] from ");
    Serial.print(client.remoteIP());
    Serial.printf(" port %d : ", client.remotePort());
    Serial.write(buffer, len);
    Serial.println();

    // Echo back
    client.write(buffer, len);

    // call stop() to free memory by Client
    client.stop();
}
}

/*****
 *!
  @brief The setup function runs once when the board comes out of reset
 */
/*****/
void setup()
{
    Serial.begin(115200);

    // Wait for the serial port to connect. Needed for native USB port only.
    while (!Serial) delay(1);

    Serial.println("TCP Server Example (Callbacks)\r\n");

    // Print all software versions
    Feather.printVersions();

    while ( !connectAP() )
    {
        delay(500); // delay between each attempt
    }

    // Connected: Print network info
    Feather.printNetwork();

    // Tell the TCP Server to auto print error codes and halt on errors
    tcpserver.err_actions(true, true);

    // Setup callbacks: must be done before begin()
    tcpserver.setConnectCallback(connect_request_callback);

    // Starting server at defined port
    tcpserver.begin();

    Serial.print("Listening on port "); Serial.println(PORT);
}

/*****
 *!
  @brief This loop function runs over and over again
 */
/*****/
void loop()
{

```

```

}

/*****
/*!
  @brief  Connect to the pre-defined access point
*/
*****/
bool connectAP(void)
{
  // Attempt to connect to an AP
  Serial.print("Please wait while connecting to: '" WLAN_SSID "' ... ");

  if ( Feather.connect(WLAN_SSID, WLAN_PASS) )
  {
    Serial.println("Connected!");
  }
  else
  {
    Serial.printf("Failed! %s (%d)", Feather.errstr(), Feather.errno());
    Serial.println();
  }
  Serial.println();

  return Feather.connected();
}

```

AdafruitUDP

AdafruitUDP makes it easy to work with raw UDP sockets. It includes a convenient callback for incoming data, and a number of helper functions to read and write data over a UDP socket.

You're free to 'poll' for incoming data and connection status, but the 'data received' callback fires whenever incoming data is available, which can then be read via the `.read()` and related commands. Callbacks aren't mandatory, but help keep your code easy to understand and more maintainable as your project grows in complexity.

UDP Socket API

The AdafruitUDP class includes the following functions:

```

// UDP API
virtual uint8_t   begin          (uint16_t port);
virtual void      stop           (void);
virtual int       beginPacket    (IPAddress ip, uint16_t port);
virtual int       beginPacket    (const char *host, uint16_t port);
virtual int       endPacket      (void);
virtual int       parsePacket    (void);
virtual IPAddress remoteIP       (void);
virtual uint16_t  remotePort     (void);

// Stream API
virtual int       read           (void);
virtual int       read           (unsigned char* buffer, size_t len);
virtual int       read           (char* buffer, size_t len);
virtual int       peek           (void);
virtual int       available      (void);

```

```
virtual void      flush      (void);
virtual size_t   write      (uint8_t byte);
virtual size_t   write      (const uint8_t *buffer, size_t size);

// Callback
void setReceivedCallback (udpcallback_t fp);
```

UDP API

The following functions are primarily based on the [Arduino EthernetUDP \(https://adafruit.it/IGA\)](https://adafruit.it/IGA) class and enable you to work with UDP connections and packets.

uint8_t begin (uint16_t port)

Initialises the AdafruitUDP class for the specified **local** port.

Parameters:

- **port:** The **local** port number to listen on (0..65535)

Returns: 1 if successful, 0 if there are no sockets available to be used.

void stop (void)

Disconnects from the UDP server, and releases any resources used during the UDP session.

Parameters: None

Returns: Nothing

int beginPacket (IPAddress ip, uint16_t port)

Starts a UDP connection to write data to the specified **remote** IP address and port.

Parameters:

- **ip:** The **remote** [IPAddress \(https://adafruit.it/IGd\)](https://adafruit.it/IGd) where the UDP server is located
- **port:** The **remote** port number to connect to (0..65535)

Returns: '1' if successful, '0' if there was a problem connecting to the specified IP address or port.

int beginPacket (const char *host, uint16_t port)

Starts a UDP connection to write data to the specified domain name and remote port.

Parameters:

- **host:** A string containing the domain name to connect to
- **port:** The port number to connect to (0..65536)

Returns: '1' if the connection was successfully established, otherwise '0'.

int endPacket (void)

This function must be called after writing UDP data to the remote server.

Parameters: None

Returns: '1' if the packet was sent successfully, otherwise '0'.

int parsePacket (void)

Checks whether a UDP packet is available, and returns the size of the UDP packet as a return value.

You must call this function BEFORE reading any data from the buffer via `AdafruitUDP.read()`!

Parameters: None

Returns: The number of bytes available in the buffered UDP packet.

IPAddress remoteIP (void)

Returns the IP address of the remote UDP server.

`AdafruitUDP.parsePacket()` must be called BEFORE this function.

Parameters: None

Returns: The [IPAddress](https://adafru.it/IGd) (<https://adafru.it/IGd>) of the remote UDP server/connection.

uint16_t remotePort (void)

Returns the port for the remote UDP server.

AdafruitUDP.parsePacket() must be called BEFORE this function.

Parameters: None

Returns: The port of the remote UDP server/connection.

Stream API

The following functions are based on the [Stream \(https://adafru.it/lGe\)](https://adafru.it/lGe) class that [Arduino EthernetUDP \(https://adafru.it/lGA\)](https://adafru.it/lGA) implements.

int read (void)

Reads the first available byte in the UDP buffer.

This function must be called AFTER AdafruitUDP.parsePacket()!

Parameters: None

Returns: The first character available in the UDP buffer, or 'EOF' if no data is available.

int read (unsigned char* buffer, size_t len) int read (char* buffer, size_t len)

These two identical functions (other than the type used for the 'buffer') will read up to 'len' bytes from the UDP response data, copying them into the buffer provided in the first argument of this function.

This function must be called AFTER AdafruitUDP.parsePacket()!

Parameters:

- **buffer:** A pointer to the buffer where the UDP data will be copied
- **len:** The maximum number of bytes to read

Returns:

- The actual number of bytes read from the UDP data and copied into 'buffer'
- '0' if no data was read or available
- '-1' if an error occurred

int peek (void)

Reads a single byte from the UDP response buffer without advancing to the next position in the buffer.

This function must be called AFTER AdafruitUDP.parsePacket()!

Parameters: None

Returns: The first byte available in the UDP buffer, or '-1' if no data is available.

int available (void)

Returns the number of bytes available to be read in the UDP buffer.

This function must be called AFTER AdafruitUDP.parsePacket()!

Parameters: None

Returns: The number of bytes available to be read in the UDP buffer, otherwise '0' if the read buffer is empty.

void flush (void)

This function will flush the buffer of any outgoing data, and return when the buffered data has been sent and the buffer is empty.

Parameters: None

Returns: Nothing

size_t write (uint8_t byte)

Writes a single byte to the remote UDP server. This function must be placed after **AdafruitUDP.beginPacket()** and before **AdafruitUDP.endPacket()**. The packet will not be sent until **.endPacket** is called!

Parameters:

- **byte:** The single byte to write to the transmit buffer

Returns: The number of bytes written.

size_t write (const uint8_t *buffer, size_t size)

Writes the specified 'buffer' to the remote UDP server. This function must be placed after **AdafruitUDP.beginPacket()** and before **AdafruitUDP.endPacket()**. The packet will not be sent until **.endPacket** is called!

Parameters:

- **buffer:** The buffer where the data to transmit is stored
- **size:** The number of bytes contained in 'buffer'

Returns: The number of bytes written.

Callback Handlers

AdafruitUDP supports a 'read' callback that will fire every time incoming UDP data is recieved over the open socket connection.

The callback function has the following signature (although you are free to choose a different name if you wish to):

```
void received_callback(void);
```

Before you can use the callback function, you need to register your callback handler (using the function signature in the paragraph above).

You register the callback with the following function:

void setReceivedCallback (udpcallback_t fp)

Registers the function used to process 'data received' callbacks.

Parameters:

- **fp**: The name of the function where callback events should be redirected to

Returns: Nothing

See the example section at the bottom of this page for details on using the data received callback in the real world.

Examples

The examples below illustrate some basic UDP concepts to help you understand the class described above.

UDP Echo Server

The following example will listen on port 8888 for any incoming UDP requests, and then echo them back to the requesting device via the '**received**' callback handler:

```
#include <adafruit_feather.h>;

#define WLAN_SSID      "yourSSID"
#define WLAN_PASS      "yourPass"

#define LOCAL_PORT      8888

AdafruitUDP udp;

char packetBuffer[255];

bool connectAP(void)
{
  // Attempt to connect to an AP
  Serial.print("Attempting to connect to: ");
  Serial.println(WLAN_SSID);

  if ( Feather.connect(WLAN_SSID, WLAN_PASS) )
  {
    Serial.println("Connected!");
  }
}
```

```

else
{
    Serial.printf("Failed! %s (%d)", Feather.errstr(), Feather.errno());
    Serial.println();
}
Serial.println();

return Feather.connected();
}

void setup()
{
    Serial.begin(115200);

    // wait for Serial port to connect. Needed for native USB port only
    while (!Serial) delay(1);

    Serial.println("UDP Echo Callback Example");
    Serial.println();

    while( !connectAP() )
    {
        delay(500);
    }

    // Tell the UDP client to auto print error codes and halt on errors
    udp_err_actions(true, true);
    udp.setReceivedCallback(received_callback);

    Serial.printf("Opening UDP at port %d ... ", LOCAL_PORT);
    udp.begin(LOCAL_PORT);
    Serial.println("OK");

    Serial.println("Please use your PC/mobile and send any text to ");
    Serial.print( IPAddress(Feather.localIP()) );
    Serial.print(" UDP port ");
    Serial.println(LOCAL_PORT);
}

void loop()
{
}

/*****
/*!
    @brief Received something from the UDP port
*/
*****/
void received_callback(void)
{
    int packetSize = udp.parsePacket();

    if ( packetSize )
    {
        // Print out the contents with remote information
        Serial.printf("Received %d bytes from ", packetSize);
        Serial.print( IPAddress(udp.remoteIP()) );
        Serial.print(" : ");
        Serial.println( udp.remotePort() );

        udp.read(packetBuffer, sizeof(packetBuffer));
        Serial.print("Contents: ");
        Serial.write(packetBuffer, packetSize);
        Serial.println();

        // Echo back contents
        udp.beginPacket(udp.remoteIP(), udp.remotePort());
        udp.write(packetBuffer, packetSize);
    }
}

```

```
    udp.endPacket();  
  }  
}
```

AdafruitHTTP

AdafruitHTTP helps make working with HTTP requests easier, including HTTPS based servers with TLS certificates.

It includes convenient callbacks for incoming data, as well as helper functions to deal with HTTP response headers, response codes, and other HTTP specific details.

AdafruitHTTP API

The AdafruitHTTP class has the following public functions:

```
bool addHeader    ( const char* name, const char* value );
bool clearHeaders ( void );

bool get          ( char const *url );
bool get          ( char const * host, char const *url );

bool post         ( char const *url, char const* encoded_data );
bool post         ( char const * host, char const *url, char const* encoded_data );
```

HTTP Headers

The follow functions are provided as helpers working with 'header' entries in your HTTP requests.

bool addHeader (const char* name, const char* value)

Adds a new header name/value pair to the HTTP request.

Parameters:

- **name:** A null-terminated string representing the 'name' in the header name/value pair.
- **value:** A null-terminated string representing the 'value' in the name/value pair.

Returns: 'True' (1) if the header was successfully added, otherwise 'false' (0).

Up to ten (10) header name/value pairs can be inserted into your HTTP request.

```
// Setup the HTTP request with any required header entries
http.addHeader("User-Agent", "curl/7.45.0"); // Simulate curl
```

```
http.addHeader("Accept", "text/html");  
http.addHeader("Connection", "keep-alive");
```

bool clearHeaders (void)

Clears all user-defined headers in the pending HTTP request.

Parameters: None

Returns: 'True' (1) if the headers were successfully cleared, otherwise 'false' (0).

HTTP GET Requests

The following functions enable you to send HTTP **GET** requests to an HTTP server:

bool get (char const* url)

This is a shortcut for the function below and uses the 'host' specified in `.connect` instead of re-entering it in the get request. See below for details.

This shortcut function will only work if you used `.connect` with a domain name. It will return an error if you used `.connect` with an IP address. Please use the full `.get()` function below when connecting via an IP address.

bool get (char const* host, char const* url)

Sends a **GET** request to the specified host and url.

Parameters:

- **host:** A null-terminated string containing the host name for the HTTP server (ex. "www.adafruit.com"). This is normally the same as the host used in `.connect`, but you can also access other host names that resolve to the same domain or IP such as "learn.adafruit.com" or "io.adafruit.com".
- **url:** The path for the HTTP request (ex. "/home/about.html")

Returns: 'True' (1) if the request was successful, otherwise 'false' (0).

```
// Connect to the HTTP server  
http.connect("www.adafruit.com", 80);  
  
// Add the required HTTP header name/value pairs  
http.addHeader("User-Agent", "curl/7.45.0"); // Simulate curl  
http.addHeader("Accept", "text/html");  
http.addHeader("Connection", "keep-alive");
```

```
// Send the HTTP GET request
http.get("wifitest.adafruit.com", "/testwifi/index.html");
```

HTTP POST Requests

HTTP **POST** requests allow you to submit data to the HTTP server via optional encoded arguments in the URL.

The following functions help you work with **POST** requests:

`bool post (char const* url, char const* encoded_data)`

This is a shortcut for the function below and uses the 'host' specified in `.connect` instead of re-entering it in the post request. See below for details.

This shortcut function will only work if you used `.connect` with a domain name. It will return an error if you used `.connect` with an IP address. Please use the full `.post()` function below when connecting via an IP address.

`bool post (char const* host, char const* url, char const* encoded_data)`

Sends a **POST** request to the HTTP server at 'host'.

Parameters:

- **host:** A null-terminated string containing the host name for the HTTP server (ex. "www.adafruit.com"). This is normally the same as the host used in `.connect`, but you can also access other host names that resolve to the same domain or IP such as "learn.adafruit.com" or "io.adafruit.com".
- **url:** The path for the HTTP post, minus the encoded arguments ("ex. "/testwifi/testpost.php"
- **encoded_data:** The encoded data to send in the post request (minus the '?' characters, ex.: "name=feather&email=feather%40adafruit.com").

Note the "%40" for the '@' symbol in encoded_data above. All non alphanumeric characters must be encoded before being transmitted.

Returns: 'True' (1) if the post succeeded, otherwise 'false' (0).

```
// Connect to the HTTP server
http.connect("www.adafruit.com", 80);

// Add the required HTTP header name/value pairs
http.addHeader("User-Agent", "curl/7.45.0"); // Simulate curl
http.addHeader("Accept", "text/html");
http.addHeader("Connection", "keep-alive");

// Send the HTTP POST request
http.post("wifitest.adafruit.com",
          "/testwifi/testpost.php",
          "name=feather&email=feather%40adafruit.com");
```

HTTP GET Example

The following example shows a simple GET request using callbacks to handle the response from the HTTP server:

```
/******
This is an example for our WICED Feather WIFI modules

Pick one up today in the adafruit shop!

Adafruit invests time and resources providing this open source code,
please support Adafruit and open-source hardware by purchasing
products from Adafruit!

MIT license, check LICENSE for more information
All text above, and the splash screen below must be included in
any redistribution
*****/

#include <adafruit_feather.h>;
#include <adafruit_http.h>;

#define WLAN_SSID          "yourSSID"
#define WLAN_PASS          "yourPassword"

#define SERVER              "wifitest.adafruit.com"    // The TCP server to
connect to
#define PAGE                "/testwifi/index.html" // The HTTP resource to request
#define PORT                80                      // The TCP port to use

// Some servers such as Facebook check the user_agent header to
// return data accordingly. Setting 'curl' mimics a command line browser.
// For a list of popular user agents see: http://www.useragentstring.com/pages/
useragentstring.php
#define USER_AGENT_HEADER  "curl/7.45.0"

int ledPin = PA15;

// Use the HTTP class
AdafruitHTTP http;

/******
@brief TCP/HTTP received callback
*/
/******
void receive_callback(void)
{
  // If there are incoming bytes available
  // from the server, read then print them:
  while ( http.available() )
```

```

    {
        int c = http.read();
        Serial.write( (isprint(c) || iscntrl(c)) ? ((char)c) : '.');
    }
}

/*****
 *!
  @brief TCP/HTTP disconnect callback
 */
/*****
void disconnect_callback(void)
{
    Serial.println();
    Serial.println("-----");
    Serial.println("DISCONNECTED CALLBACK");
    Serial.println("-----");
    Serial.println();

    http.stop();
}

/*****
 *!
  @brief The setup function runs once when the board comes out of reset
 */
/*****
void setup()
{
    Serial.begin(115200);

    // Wait for the USB serial to connect. Needed for native USB port only.
    while (!Serial) delay(1);

    Serial.println("HTTP Get Example (Callback Based)\r\n");

    // Print all software versions
    Feather.printVersions();

    // Try to connect to an AP
    while ( !connectAP() )
    {
        delay(500); // delay between each attempt
    }

    // Connected: Print network info
    Feather.printNetwork();

    // Tell the HTTP client to auto print error codes and halt on errors
    http.err_actions(true, true);

    // Set the callback handlers
    http.setReceivedCallback(receive_callback);
    http.setDisconnectCallback(disconnect_callback);

    // Connect to the HTTP server
    Serial.printf("Connecting to %s port %d ... ", SERVER, PORT);
    http.connect(SERVER, PORT); // Will halt if an error occurs
    Serial.println("OK");

    // Setup the HTTP request with any required header entries
    http.addHeader("User-Agent", USER_AGENT_HEADER);
    http.addHeader("Accept", "text/html");
    http.addHeader("Connection", "keep-alive");

    // Send the HTTP request
    Serial.printf("Requesting '%s' ... ", PAGE);
    http.get(SERVER, PAGE); // Will halt if an error occurs
    Serial.println("OK");
}

```

```

}

/*****
/*!
  @brief  The loop function runs over and over again
*/
*****/
void loop()
{
  togglePin(ledPin);
  delay(250);
}

/*****
/*!
  @brief  Connect to the defined access point (AP)
*/
*****/
bool connectAP(void)
{
  // Attempt to connect to an AP
  Serial.print("Attempting to connect to: ");
  Serial.println(WLAN_SSID);

  if ( Feather.connect(WLAN_SSID, WLAN_PASS) )
  {
    Serial.println("Connected!");
  }
  else
  {
    Serial.printf("Failed! %s (%d)", Feather.errstr(), Feather.errno());
    Serial.println();
  }
  Serial.println();

  return Feather.connected();
}

```

AdafruitHTTPServer

The AdafruitHTTPServer class requires WICED Feather Lib 0.6.0 or higher to run.

AdafruitHTTPServer makes it easy to run an HTTP server on the WICED feather in either SoftAP or normal operating mode, allowing you to implement custom admin consoles, rich data visualisations, or to publish 'always available' documentation for your project right on the board itself.

The helper class allows you to serve static content stored in flash memory (compiled as part of the Arduino sketch itself), link to files on the 16MBit SPI flash on the WICED Feather (if enabled via the optional solder jumper on the bottom of the board), or to dynamically generate page content on to go.

AdafruitHTTPServer API

The AdafruitHTTPServer class has the following public functions:

```
AdafruitHTTPServer(uint8_t max_pages, uint8_t interface = WIFI_INTERFACE_STATION);
uint8_t interface ( void );
void addPages(HTTPPage const* http_pages, uint8_t count = 1);
bool begin(uint16_t port,
           uint8_t max_clients,
           uint32_t stacksize = HTTPSERVER_STACKSIZE_DEFAULT);
void stop(void);
bool started(void);
```

Dynamic page content can be generated with the following callback handler signature, changing the function name to something appropriate:

```
void dynamic_page_generator (const char* url,
                             const char* query,
                             httprequest_t* http_request);
```

Constructor

When declaring a new instance of the AdafruitHTTPServer class you must declare the maximum number of pages that the server will host (based on available memory since each page record will require a chunk of SRAM to be allocated), and whether the server is running in normal (non access point) mode, or in AP mode.

You indicate the operating mode via the ' **interface** ' field, which has one of the following values:

- **WIFI_INTERFACE_STATION** : Default value, meaning this should run in normal non AP mode
- **WIFI_INTERFACE_AP** : Indicates that the HTTP server should run on the AP (Access Point) interface

For example, to use the default (non AP) interface for the HTTP server you might use the following constructor declaration:

```
const char hello_html[] = "<html><body> <h1>Hello World!</h1> </body></html>";

HTTPPage pages[] =
{
    HTTPPageRedirect("/", "/hello.html"), // Redirect root to hello page
    HTTPPage("/hello.html", HTTP_MIME_TEXT_HTML, hello_html),
};
```

```
uint8_t pagecount = sizeof(pages)/sizeof(HTTPPage);  
  
// Declare HTTPServer with max number of pages  
AdafruitHTTPServer httpserver(pagecount);
```

Adding Pages

All pages served by the HTTP server must be declared at compile time in a specifically formatted list made up of the following record types:

1. HTTPPageRedirect Records (Page Redirection Entries)

An **HTTPPageRedirect** entry redirects all requests for the specified resource to another location, and contains a string with the page to redirect from and the page to redirect to.

2. HTTPPage Records (Standard Pages)

An **HTTPPage** is composed of the page path + name, the mime type string (so that the browser knows how to render the resource), and the reference to the resource itself, which can be one of the following:

- **A Raw String** : The text contained in the specified string will be served as the page contents
- **An HTTPResource (Static File)** : The variable name for the binary contents of a file, converted using the [pyresource](https://adafru.it/qoD) (<https://adafru.it/qoD>) tool. This tool takes binary or text files, and converts them to standard C headers, with the file contents added as an **HTTPResource** that AdafruitHTTPServer understands. This allows you to insert static pages, images or other file types, and the mime type will be used to indicate how the resource should be rendered in the browser.
- **A Dynamic Callback Handler** : The specified callback handler function will be called when this resource is requested, and you can generate the page contents dynamically in the callback handler
- **An SPI Flash Filename** : The path and filename to retrieve a file from on the on board SPI flash if enabled (files can be added to SPI flash over USB mass storage when the SPI flash is enabled via the optional solder jumper on the bottom of the board).

A sample list of a well formatted page list can be seen below, where raw string data ('hello_html'), and dynamic content ('info_html_generator' and 'file_not_found_generator') are both present, as well as a redirection of root ('/') to '/hello.html':

```
void info_html_generator      (const char* url, const char* query,  
httppage_request_t* http_request);
```

```

void file_not_found_generator (const char* url, const char* query,
httppage_request_t* http_request);

const char hello_html[] = "<html><body> <h1>Hello World!</h1> </body></html>";

HTTPPage pages[] =
{
    HTTPPageRedirect("/", "/hello.html"), // redirect root to hello page
    HTTPPage("/hello.html", HTTP_MIME_TEXT_HTML, hello_html),
    HTTPPage("/info.html", HTTP_MIME_TEXT_HTML, info_html_generator),
    HTTPPage("/404.html", HTTP_MIME_TEXT_HTML, file_not_found_generator),
};

# Note that we need to indicate the page count in the constructor!
// Declare HTTPServer with max number of pages
uint8_t pagecount = sizeof(pages)/sizeof(HTTPPage);
AdafruitHTTPServer httpserver(pagecount);

```

Converting Static Content (HTTPResources)

It's easy to convert a set of static files to resources that AdafruitHTTPServer can use and embed in the sketch itself. For details see the dedicated [pyresource \(https://adafru.it/qoD\)](https://adafru.it/qoD) tool page.

Implementing Dynamic Page Handlers

Two of the HTTPPage entries in the example above ('/info.html' and '/404.html') show how dynamic pages can be added to the HTTP server.

The dynamic page function prototypes are declared at the top of the code above, and the functions can then be implemented following the example below, which is called when a 404 error occurs:

The HTTP Server will always redirect to '/404.html' when a 404 error occurs (meaning a user requested a URL that is not available in the HTTPPage list included at compile time). As such, it is a good idea to always include this page in your project.

```

/*****
 *!
 * @brief HTTP 404 generator. The HTTP Server will automatically redirect
 * to "/404.html" when it can't find the requested url in the
 * list of registered pages
 *
 * The url and query string are already separated when this function
 * is called.
 *
 * @param url url of this page
 * @param query query string after '?' e.g "var=value"
 * @param http_request Details about this HTTP request
 */
/*****
void file_not_found_generator (const char* url, const char* query,
httppage_request_t* http_request)
{

```

```

(void) url;
(void) query;
(void) http_request;

httpserver.print("<html><body>");
httpserver.print("<h1>Error 404 File Not Found!</h1>");
httpserver.print("<br>");

httpserver.print("Available pages are:");
httpserver.print("<br>");

// Show a link list of all available pages:
httpserver.print("<ul>");
for(int i=0; i<pagecount; i++)
{
    httpserver.print("<li>");
    httpserver.print(pages[i].url);
    httpserver.print("</li>");
}
httpserver.print("</ul>");

httpserver.print("</body></html>");
}

```

Registering the Pages

Once you have create your file list and implemented any dynamic page handlers, you must register your page list with the class via `.addPages`.

You must call the `.addPages` function BEFORE calling the `.begin` function which starts the HTTP server!

`.addPages` can be called multiple times before `.begin` if you wish to organize your page list into several sets, but be sure that the 'max_pages' value used in the constructor is big enough to accommodate all the pages.

```

// Configure HTTP Server Pages
Serial.println("Adding Pages to HTTP Server");
httpserver.addPages(pages, pagecount);

Serial.print("Starting HTTP Server ... ");
httpserver.begin(PORT, MAX_CLIENTS);
Serial.println(" running");

```

Starting/Stopping the HTTP Server

You can start the HTTP server using the `.begin` function (and stop it via `.stop`), with the following function signatures:

Make sure you call the `.addPages` function BEFORE calling the `.begin` function which starts the HTTP server!

```
bool begin(uint16_t port,
           uint8_t  max_clients,
           uint32_t stacksize = HTTPSERVER_STACKSIZE_DEFAULT);

void stop(void);
```

- **port:** The port number to expose the HTTP server on (generally 80 or 8080, but this can be any port you wish and you can even have multiple instances of the HTTP server running on different ports if you wish).
- **max_clients:** The maximum number of client connections to accept before refusing requests. This should generally be kept as low as possible since there is limited SRAM available on the system. 3 is a good number if there will be multiple file requests at once, for example.
- **stacksize:** This should generally be left at the default value, but if you require a larger stack for the HTTP server you can adjust the value here within the limit of available system resources.

Complete Example

The following code shows an example using the `AdafruitHTTPServer` class, but numerous examples are included as part of the library in the **HTTPServer** folder, and the latter may be more up to date.

To use this example, update the **WLAN_SSID** and **WLAD_PASS** fields, flash the sketch to the **User Code** section of your WICED Feather, and then open the Serial Monitor and wait for the connection to finish. Once connected, the HTTP server will start and you can navigate to the IP address of your board to browse the pages added below.

```
/* This example uses the AdafruitHTTPServer class to create a simple webserver */

#include <adafruit_feather.h>;
#include <adafruit_http_server.h>;

#define WLAN_SSID      "yourSSID"
#define WLAN_PASS      "yourPassword"

#define PORT           80           // The TCP port to use
#define MAX_CLIENTS    3

int ledPin = PA15;
int visit_count = 0;

void info_html_generator      (const char* url, const char* query,
```

```

httppage_request_t* http_request);
void file_not_found_generator (const char* url, const char* query,
httppage_request_t* http_request);

const char hello_html[] = "<html><body> <h1>Hello World!</h1> </body></html>";

HTTPPage pages[] =
{
    HTTPPageRedirect("/", "/hello.html"), // redirect root to hello page
    HTTPPage("/hello.html", HTTP_MIME_TEXT_HTML, hello_html),
    HTTPPage("/info.html" , HTTP_MIME_TEXT_HTML, info_html_generator),
    HTTPPage("/404.html" , HTTP_MIME_TEXT_HTML, file_not_found_generator),
};

uint8_t pagecount = sizeof(pages)/sizeof(HTTPPage);

// Declare HTTPServer with max number of pages
AdafruitHTTPServer httpserver(pagecount);

/*****
*!
* @brief Example of generating dynamic HTML content on demand
*
* Link is separated to url and query
*
* @param url          url of this page
* @param query        query string after '?' e.g "var=value"
*
* @param http_request This request's information
*/
void info_html_generator (const char* url, const char* query, httppage_request_t*
http_request)
{
    (void) url;
    (void) query;
    (void) http_request;

    httpserver.print("<b>Bootloader</b> : ");
    httpserver.print( Feather.bootloaderVersion() );
    httpserver.print("<br>");

    httpserver.print("<b>WICED SDK</b> : ");
    httpserver.print( Feather.sdkVersion() );
    httpserver.print("<br>");

    httpserver.print("<b>FeatherLib</b> : ");
    httpserver.print( Feather.firmwareVersion() );
    httpserver.print("<br>");

    httpserver.print("<b>Arduino API</b> : ");
    httpserver.print( Feather.arduinoVersion() );
    httpserver.print("<br>");
    httpserver.print("<br>");

    visit_count++;
    httpserver.print("<b>visit count</b> : ");
    httpserver.print(visit_count);
}

/*****
*!
* @brief HTTP 404 generator. The HTTP Server will automatically redirect
*       to "/404.html" when it can't find the requested url in the
*       list of registered pages
*
* The url and query string are already separated when this function
* is called.

```

```

*
* @param url          url of this page
* @param query        query string after '?' e.g "var=value"
* @param http_request Details about this HTTP request
*/
/*****/
void file_not_found_generator (const char* url, const char* query,
httppage_request_t* http_request)
{
    (void) url;
    (void) query;
    (void) http_request;

    httpserver.print("<html><body>");
    httpserver.print("<h1>Error 404 File Not Found</h1>");
    httpserver.print("<br>");

    httpserver.print("Available pages are:");
    httpserver.print("<br>");

    httpserver.print("<ul>");
    for(int i=0; i<pagecount; i++)
    {
        httpserver.print("<li>");
        httpserver.print(pages[i].url);
        httpserver.print("</li>");
    }
    httpserver.print("</ul>");

    httpserver.print("</body></html>");
}

/*****/
/*!
 @brief The setup function runs once when the board comes out of reset
*/
/*****/
void setup()
{
    Serial.begin(115200);

    // Wait for the USB serial to connect. Needed for native USB port only.
    while (!Serial) delay(1);

    Serial.println("Simple HTTP Server Example\r\n");

    // Print all software versions
    Feather.printVersions();

    // Try to connect to an AP
    while ( !connectAP() )
    {
        delay(500); // delay between each attempt
    }

    // Connected: Print network info
    Feather.printNetwork();

    // Tell the HTTP client to auto print error codes and halt on errors
    httpserver.err_actions(true, true);

    // Configure HTTP Server Pages
    Serial.println("Adding Pages to HTTP Server");
    httpserver.addPages(pages, pagecount);

    Serial.print("Starting HTTP Server ... ");
    httpserver.begin(PORT, MAX_CLIENTS);
    Serial.println(" running");
}

```

```

/*****
/*!
  @brief The loop function runs over and over again
*/
/*****
void loop()
{
  togglePin(ledPin);
  delay(1000);
}

/*****
/*!
  @brief Connect to the defined access point (AP)
*/
/*****
bool connectAP(void)
{
  // Attempt to connect to an AP
  Serial.print("Please wait while connecting to: '" WLAN_SSID "' ... ");

  if ( Feather.connect(WLAN_SSID, WLAN_PASS) )
  {
    Serial.println("Connected!");
  }
  else
  {
    Serial.printf("Failed! %s (%d)", Feather.errstr(), Feather.errno());
    Serial.println();
  }
  Serial.println();

  return Feather.connected();
}

/*****
/*!
  @brief TCP/HTTP disconnect callback
*/
/*****
void disconnect_callback(void)
{
  Serial.println();
  Serial.println("-----");
  Serial.println("DISCONNECTED CALLBACK");
  Serial.println("-----");
  Serial.println();

  httpserver.stop();
}

```

AdafruitMQTT

The Adafruit WICED Feather API includes an internal MQTT client that allows you perform basic MQTT operations directly with any MQTT broker.

AdafruitMQTT inherits from AdafruitTCP and also has access to all of the functions defined in the parent class.

Note: You are also free to use an external [Client \(https://adafru.it/IFj\)](https://adafru.it/IFj) based MQTT library (for example [Adafruit_MQTT_Library \(https://adafru.it/fp6\)](https://adafru.it/fp6)) if you prefer or need

something fully under your control. AdafruitMQTT is provided for convenience sake, and to avoid external dependencies, but isn't the only option at your disposal.

Constructors

Some MQTT brokers require a username and password to connect. If necessary, the two values should be provided in the constructor when declaring an instance of AdafruitMQTT.

If no username and password are required, simply use the default empty constructor.

```
AdafruitMQTT()  
AdafruitMQTT(const char* username, const char* password)
```

Functions

```
bool connected ( void );  
  
bool connect    ( IPAddress ip,  
                  uint16_t  port      = 1883,  
                  bool      cleanSession = true,  
                  uint16_t  keepalive_sec = MQTT_KEEPALIVE_DEFAULT);  
bool connect    ( const char* host,  
                  uint16_t  port      = 1883,  
                  bool      cleanSession = true,  
                  uint16_t  keepalive_sec = MQTT_KEEPALIVE_DEFAULT);  
  
bool connectSSL ( IPAddress ip,  
                  uint16_t  port      = 8883,  
                  bool      cleanSession = true,  
                  uint16_t  keepalive_sec = MQTT_KEEPALIVE_DEFAULT);  
bool connectSSL ( const char* host,  
                  uint16_t  port      = 8883,  
                  bool      cleanSession = true,  
                  uint16_t  keepalive_sec = MQTT_KEEPALIVE_DEFAULT);  
  
bool disconnect ( void );  
  
bool publish    ( UTF8String topic,  
                  UTF8String message,  
                  uint8_t   qos      = MQTT_QOS_AT_MOST_ONCE,  
                  bool      retained = false );  
  
bool subscribe  ( const char* topicFilter,  
                  uint8_t   qos,  
                  messageHandler mh);  
  
bool unsubscribe( const char* topicFilter );  
  
void will       ( const char* topic,  
                  UTF8String message,  
                  uint8_t   qos      = MQTT_QOS_AT_MOST_ONCE,  
                  uint8_t   retained = 0);  
  
void clientID   ( const char* client)  
  
void setDisconnectCallback ( void (*fp) (void) )
```

Connection Management

AdafruitMQTT can connect to an MQTT broker using both 'open' (unencrypted) or 'secure' (TLS/SSL encrypted) connections.

bool connected(void)

Indicates if we are currently connected to the MQTT broker or not.

Parameters: None

Returns: 'True' (1) if we are connected to the MQTT broker, otherwise 'false' (0).

```
bool connect ( IPAddress ip, uint16_t port = 1883, bool  
cleanSession = true, uint16_t keepalive_sec =  
MQTT_KEEPALIVE_DEFAULT);
```

Establishes an open connection with the specified MQTT broker.

Parameters:

- **ip:** The IP address for the MQTT broker
- **port:** The port to use (default = 1883)
- **cleanSession:** Indicates whether the client and broker should remember 'state' across restarts and reconnects (based on the 'Client ID' value set in the constructor):
 - If set to **false** (0) both the client and server will maintain state across restarts of the client, the server and the connection. As state is maintained:
 - Message delivery will be reliable meeting the specified QOS even if the client, server or connection are restarted.
 - The server will treat a subscription as durable.
 - If set to **true** (1) the client and server will not maintain state across restarts of the client, the server or the connection. This means:
 - Message delivery to the specified QOS cannot be maintained if the client, server or connection are restarted
 - The server will treat a subscription as non-durable
- **keepalive_sec:** This value defines the maximum interval (in seconds) between messages being sent or received. Setting a value here ensures that at least one message is sent between the client and the broker within every 'keep alive' period. If no data was sent within 'keepalive_sec' seconds, the Client will send a

simple ping to the broker to keep the connection alive. Setting this value to '0' disables the keep alive feature. **The default value is 60 seconds.**

Returns: 'True' (1) if the connection was successful, otherwise 'false' (0).

```
bool connect ( const char* host, uint16_t port = 1883, bool
cleanSession = true, uint16_t keepalive_sec =
MQTT_KEEPALIVE_DEFAULT);
```

Establishes an open connection with the specified MQTT broker.

Parameters:

- **host:** The domain name for the MQTT broker
- **port:** The port to use (default = 1883)
- **cleanSession:** Indicates whether the client and broker should remember 'state' across restarts and reconnects (based on the 'Client ID' value set in the constructor):
 - If set to **false** (0) both the client and server will maintain state across restarts of the client, the server and the connection. As state is maintained:
 - Message delivery will be reliable meeting the specified QOS even if the client, server or connection are restarted.
 - The server will treat a subscription as durable.
 - If set to **true** (1) the client and server will not maintain state across restarts of the client, the server or the connection. This means:
 - Message delivery to the specified QOS cannot be maintained if the client, server or connection are restarted
 - The server will treat a subscription as non-durable
- **keepalive_sec:** This value defines the maximum interval (in seconds) between messages being sent or received. Setting a value here ensures that at least one message is sent between the client and the broker within every 'keep alive' period. If no data was sent within 'keepalive_sec' seconds, the Client will send a simple ping to the broker to keep the connection alive. Setting this value to '0' disables the keep alive feature. The default value is 60 seconds.

Returns: 'True' (1) if the connection was successful, otherwise 'false' (0).

```
bool connectSSL ( IPAddress ip, uint16_t port = 8883, bool  
cleanSession = true, uint16_t keepalive_sec =  
MQTT_KEEPALIVE_DEFAULT)
```

Establishes a secure connection with the specified MQTT broker.

Parameters:

- **ip:** The IP address of the MQTT broker
- **port:** The port to use (default = 8883)
- **cleanSession:** Indicates whether the client and broker should remember 'state' across restarts and reconnects (based on the 'Client ID' value set in the constructor):
 - If set to **false** (0) both the client and server will maintain state across restarts of the client, the server and the connection. As state is maintained:
 - Message delivery will be reliable meeting the specified QOS even if the client, server or connection are restarted.
 - The server will treat a subscription as durable.
 - If set to **true** (1) the client and server will not maintain state across restarts of the client, the server or the connection. This means:
 - Message delivery to the specified QOS cannot be maintained if the client, server or connection are restarted
 - The server will treat a subscription as non-durable
- **keepalive_sec:** This value defines the maximum interval (in seconds) between messages being sent or received. Setting a value here ensures that at least one message is sent between the client and the broker within every 'keep alive' period. If no data was sent within 'keepalive_sec' seconds, the Client will send a simple ping to the broker to keep the connection alive. Setting this value to '0' disables the keep alive feature. The default value is 60 seconds.

Returns: 'True' (1) if the connection was successful, otherwise 'false' (0).

```
bool connectSSL ( const char* host, uint16_t port =  
8883, bool cleanSession = true, uint16_t keepalive_sec =  
MQTT_KEEPALIVE_DEFAULT)
```

Establishes a secure connection with the specified MQTT broker.

Parameters:

- **host:** The domain name of the MQTT broker

- **port**: The port to use (default = 8883)
- **cleanSession**: Indicates whether the client and broker should remember 'state' across restarts and reconnects (based on the 'Client ID' value set in the constructor):
 - If set to **false** (0) both the client and server will maintain state across restarts of the client, the server and the connection. As state is maintained:
 - Message delivery will be reliable meeting the specified QOS even if the client, server or connection are restarted.
 - The server will treat a subscription as durable.
 - If set to **true** (1) the client and server will not maintain state across restarts of the client, the server or the connection. This means:
 - Message delivery to the specified QOS cannot be maintained if the client, server or connection are restarted
 - The server will treat a subscription as non-durable
- **keepalive_sec**: This value defines the maximum interval (in seconds) between messages being sent or received. Setting a value here ensures that at least one message is sent between the client and the broker within every 'keep alive' period. If no data was sent within 'keepalive_sec' seconds, the Client will send a simple ping to the broker to keep the connection alive. Setting this value to '0' disables the keep alive feature. The default value is 60 seconds.

Returns: 'True' (1) if the connection was successful, otherwise 'false' (0).

bool disconnect (void)

Disconnects from the remote MQTT broker.

Parameters: None

Returns: 'True' (1) if the disconnect was successful, otherwise 'false' (0) if an error occurred (check `.errno`, `.errstr`, etc.).

Messaging

The following functions allow you to publish, subscribe and unsubscribe to MQTT topics:

```
bool publish ( UTF8String topic, UTF8String message,
uint8_t qos = MQTT_QOS_AT_MOST_ONCE, bool
retained = false );
```

Published the supplied 'message' to the specified 'topic'.

Parameters:

- **topic:** The topic where the data should be published (ex: "adafruit/data" or "home/rooms/bedroom/temp"). UTF8String is used to make it easier to work with UTF8 data.
- **message:** The string of data to write to the specified 'topic'. UTF8String is used to make it easier to work with UTF8 data.
- **qos:** The quality of service level (see the MQTT spec for details). Default = 'At Most Once', meaning the message tries to send once but isn't persisted if the send fails. Possible values are:
 - MQTT_QOS_AT_MOST_ONCE
 - MQTT_QOS_AT_LEAST_ONCE
 - MQTT_QOS_EXACTLY_ONCE
- **retained:** Whether or not the published message should be 'retained' by the MQTT broker. Sending a message with the retained bool set to 'false' (0) will clear any previously retained message from the broker. The default value is false.

Returns: 'True' (1) if the publish was successful, otherwise 'false' (0) if an error occurred (check `.errno`, `.errstr`, etc.).

bool subscribe (const char* topicFilter, uint8_t qos, messageHandler mh);

Subscribes to a specific topic, using a callback mechanism to alert you when new data is available on the specific topicFilter.

Parameters:

- **topicFilter:** The topic name or topic 'filter' to subscribe to. This can be either a single topic ("home/kitchen/fridge/temp") or make use of a standard MQTT wildcard like "home/+", which will subscribe to changes to any topic above the 'home/' level.
- **qos:** A subscribing client can set the maximum quality of service a broker uses to send messages that match the client subscriptions. The QoS of a message forwarded to a subscriber thus might be different to the QoS given to the message by the original publisher. The lower of the two values is used to forward a message. The value of **qos** can be one of:
 - MQTT_QOS_AT_MOST_ONCE
 - MQTT_QOS_AT_LEAST_ONCE
 - MQTT_QOS_EXACTLY_ONCE

- **mh**: The MQTT subscribe callback function that will handle callback events (see **Subscribe Callback Handler** below for details).

Returns: 'True' (1) if the subscribe was successful, otherwise 'false' (0) if an error occurred (check `.errno`, `.errstr`, etc.).

Subscribe Callback Handler(s)

When you subscribe to a specific topic or topic filter, you also need to pass in a callback function that will be used to handle any subscribe matches or events.

You can subscribe to up to EIGHT (8) topics with the internal MQTT client.

The same callback handler can be used for multiple subscriptions, or you can use individual callbacks for each subscribe. The choice will depend on your specific project.

MQTT subscribe callback functions must have the following format:

```

/*****
/*!
  @brief  MQTT subscribe event callback handler

  @param  topic      The topic causing this callback to fire
  @param  message     The new value associated with 'topic'

  @note    'topic' and 'message' are UTF8Strings (byte array), which means
           they are not null-terminated like C-style strings. You can
           access its data and len using .data & .len, although there is
           also a Serial.print override to handle UTF8String data types.
*/
*****/
void subscribed_callback(UTF8String topic, UTF8String message)
{
  // Print out topic name and message
  Serial.print("[Subscribed] ");
  Serial.print(topic);
  Serial.print(" : ");
  Serial.println(message);

  // Echo back
  Serial.print("Echo back to " TOPIC_ECHO " ... ");
  mqtt.publish(TOPIC_ECHO, message); // Will halt if an error occurs
  Serial.println("OK");

  // Unsubscribe from SUBSCRIBED_TOPIC2 if we received an "stop" message
  // Won't be able to echo anymore
  if ( message == "stop" )

```

```
{  
  Serial.print("Unsubscribing from " TOPIC_SUBSCRIBE " ... ");  
  mqtt.unsubscribe(TOPIC_SUBSCRIBE); // Will halt if fails  
  Serial.println("OK");  
}  
}
```

Callback Handler Parameters

- **topic:** The topic that caused the subscribe callback to fire (UTF8-encoded)
- **message:** The UTF8 encoded message associated with topic_data

Note the use of UTF8String for 'topic' and 'message' since the strings that are returned are UTF8 encoded and NOT NULL terminated, so we need to use this helper to convert them to something we can safely print.

bool unsubscribe(const char* topicFilter);

Unsubscribes from a specific topic or topic filter.

Parameters: The topic or topic filter to unsubscribe from

Returns: 'True' (1) is the unsubscribe was successful, otherwise 'false' (0).

Last Will

MQTT has a concept called a 'Last Will' message. The optional 'Last Will' message can be set using a user-define topic, and this message will be sent if the server/broker is unable to contact the client for a specific amount of time.

This functionality isn't a mandatory part of MQTT, but can be used to detect when nodes are online and offline. When you connect, you can for example set a string like "Online" to a specific topic, and then set a last will message of "Offline" to that same topic. If the node goes offline (battery failure, disconnect, etc.), the broker will use the last will to set the topic to "Offline" once the server/client timeout occurs.

Be sure to set the last will BEFORE calling the .connect function since the last will is set during the connect phase!

```
void will ( const char* topic, UTF8String message, uint8_t
qos = MQTT_QOS_AT_MOST_ONCE, uint8_t retained =
0);
```

Sets the last will message.

Parameters:

- **topic:** The topic where the data should be published (ex: "adafruit/data" or "home/rooms/bedroom/temp").
- **message:** The string of data to write to the specified 'topic' (UTF8String is used to make it easier to work with UTF8 data).
- **qos:** The quality of service level (see the MQTT spec for details). Default = 'At Most Once', meaning the message tries to send once but isn't persisted if the send fails. Possible values are:
 - MQTT_QOS_AT_MOST_ONCE
 - MQTT_QOS_AT_LEAST_ONCE
 - MQTT_QOS_EXACTLY_ONCE
- **retained:** Whether or not the published message should be 'retained' by the MQTT broker. Sending a message with the retained bool set to 'false' (0) will clear any previously retained message from the broker. The default value is false.

Returns: 'True' (1) is the last will message was successfully set, otherwise 'false' (0).

Client ID

The client identifier (Client ID) is an string that identifies each MQTT client connecting to an MQTT broker.

This value should be unique on the broker since the broker uses it for identifying the client and the client's current 'state' of the client (subscriptions, QoS, etc.).

By default, a random 10-23 character string will be generated for the unique Client ID that gets passed to the broker during the connection process. If you wish to maintain a consistent client ID across connections, however, you can override the random client ID by using the **.clientId** function below:

```
void clientId(const char* client)
```

Sets a manual Client ID, overriding the default random value.

Parameters:

- **client:** A null-terminated string representing the client ID to pass to the MQTT broker.

Returns: Nothing

Disconnect Callback

An optional disconnect callback is available in AdafruitMQTT. This callback handler will fire when you are disconnected from the remote MQTT broker.

To use the callback, add the following function to your sketch (the function name and the contents of the function can change depending on your project requirements):

```
void disconnect_callback(void)
{
  Serial.println();
  Serial.println("-----");
  Serial.println("DISCONNECTED FROM MQTT BROKER");
  Serial.println("-----");
  Serial.println();
}
```

Then pass this function name into the **.setDisconnectCallback** function BEFORE calling **.connect** or **.connectSSL**:

```
// Set the disconnect callback handler
mqtt.setDisconnectCallback(disconnect_callback);
```

AdafruitMQTT Example

The following example illustrates how to subscribe to topics, set the last will message, publish, and implement one or more subscribe callback handlers:

This example uses the freely accessible test MQTT broker at test.mosquitto.org. This server is publicly accessible, so be careful what data you push to it since anyone can see the publications!

```
/******
This is an example for our Feather WIFI modules

Pick one up today in the adafruit shop!

Adafruit invests time and resources providing this open source code,
please support Adafruit and open-source hardware by purchasing
products from Adafruit!
******/
```

```

MIT license, check LICENSE for more information
All text above, and the splash screen below must be included in
any redistribution
*****/

#include <adafruit_feather.h>;
#include <adafruit_mqtt.h>;
#include "certificate_mosquitto.h"

/* This sketch demonstrates subscribe/unsubscribe activity with
 * callbacks.
 *
 * It will connect to a public MQTT server (with/without TLS)
 * and subscribe to TOPIC_SUBSCRIBE (defined below).
 *
 * - When a message is received, it will echo back to TOPIC_ECHO
 * - If the received message is "stop", we will
 *   unsubscribe from TOPIC_SUBSCRIBE and you won't be able to
 *   echo content back to the broker any longer.
 *
 * Note: TOPIC_SUBSCRIBE and TOPIC_ECHO must not be the same topic!
 * Ex. They must not be "adafruit/+" and "adafruit/echo", since this will
 * cause an infinite loop (received -> echo -> received -> ....)
 *
 * For details on the MQTT broker server see http://test.mosquitto.org/
 * - Port 1883 : MQTT, unencrypted
 * - Port 8883 : MQTT, encrypted (TLS)
 *
 * Note: may You need an MQTT desktop client such as the lightweight
 * Java client included in this repo: org.eclipse.paho.mqtt.utility-1.0.0.jar
 *
 * For information on configuring your system to work with MQTT see:
 * - https://learn.adafruit.com/desktop-mqtt-client-for-adafruit-io/installing-software
 *
 * To run this demo
 * 1. Change the WLAN_SSID/WLAN_PASS to match your access point
 * 2. Decide whether you want to use TLS/SSL or not (USE_TLS)
 * 3. Change TOPIC*, WILL*, enable CLIENTID if needed
 * 4. Compile and run
 * 5. Use an MQTT desktop client to connect to the same MQTT broker and
 *    publish to any topic beginning with "adafruit/feather/" (depending
 *    on TOPIC_SUBSCRIBE). To be able to receive the echo message, please
 *    also subscribe to "adafruit/feather_echo" (TOPIC_ECHO).
 */

#define WLAN_SSID      "yourSSID"
#define WLAN_PASS      "yourPass"

#define USE_TLS        0

#define BROKER_HOST     "test.mosquitto.org"
#define BROKER_PORT    (USE_TLS ? 8883 : 1883)

// Uncomment to set your own ClientID, otherwise a random ClientID is used
// #define CLIENTID      "Adafruit Feather"

#define TOPIC_SUBSCRIBE "adafruit/feather/+"
#define TOPIC_ECHO      "adafruit/feather_echo"

#define WILL_TOPIC      "adafruit/feather"
#define WILL_MESSAGE    "Goodbye!!"

AdafruitMQTT mqtt;

/*****
 *!
 * @brief Disconnect handler for MQTT broker connection

```

```

*/
/*****
void disconnect_callback(void)
{
    Serial.println();
    Serial.println("-----");
    Serial.println("DISCONNECTED FROM MQTT BROKER");
    Serial.println("-----");
    Serial.println();
}

/*****
/*!
    @brief The setup function runs once when the board comes out of reset
*/
/*****
void setup()
{
    Serial.begin(115200);

    // Wait for the USB serial port to connect. Needed for native USB port only
    while (!Serial) delay(1);

    Serial.println("MQTT Subscribe Example\r\n");

    // Print all software versions
    Feather.printVersions();

    while ( !connectAP() )
    {
        delay(500); // delay between each attempt
    }

    // Connected: Print network info
    Feather.printNetwork();

    // Tell the MQTT client to auto print error codes and halt on errors
    mqtt.err_actions(true, true);

    // Set ClientID if defined
    #ifdef CLIENTID
    mqtt.clientID(CLIENTID);
    #endif

    // Last will must be set before connecting since it is part of the connection data
    mqtt.will(WILL_TOPIC, WILL_MESSAGE, MQTT_QOS_AT_LEAST_ONCE);

    // Set the disconnect callback handler
    mqtt.setDisconnectCallback(disconnect_callback);

    Serial.printf("Connecting to " BROKER_HOST " port %d ... ", BROKER_PORT);
    if (USE_TLS)
    {
        // Disable default RootCA to save SRAM since we don't need to
        // access any other site except test.mosquitto.org
        Feather.useDefaultRootCA(false);

        // mosquitto CA is pre-generated using pycert.py
        Feather.addRootCA(rootca_certs, ROOTCA_CERTS_LEN);

        // Connect with SSL/TLS
        mqtt.connectSSL(BROKER_HOST, BROKER_PORT);
    } else
    {
        mqtt.connect(BROKER_HOST, BROKER_PORT);
    }
    Serial.println("OK");

    Serial.print("Subscribing to " TOPIC_SUBSCRIBE " ... ");

```

```

    mqtt.subscribe(TOPIC_SUBSCRIBE, MQTT_QOS_AT_MOST_ONCE, subscribed_callback); //
    Will halted if an error occurs
    Serial.println("OK");
}

/*****
/*!
    @brief This loop function runs over and over again
*/
*****/
void loop()
{

}

/*****
/*!
    @brief MQTT subscribe event callback handler

    @param topic    The topic causing this callback to fire
    @param message   The new value associated with 'topic'

    @note 'topic' and 'message' are UTF8Strings (byte array), which means
           they are not null-terminated like C-style strings. You can
           access its data and len using .data & .len, although there is
           also a Serial.print override to handle UTF8String data types.
*/
*****/
void subscribed_callback(UTF8String topic, UTF8String message)
{
    // Print out topic name and message
    Serial.print("[Subscribed] ");
    Serial.print(topic);
    Serial.print(" : ");
    Serial.println(message);

    // Echo back
    Serial.print("Echo back to " TOPIC_ECHO " ... ");
    mqtt.publish(TOPIC_ECHO, message); // Will halt if an error occurs
    Serial.println("OK");

    // Unsubscribe from SUBSCRIBED_TOPIC2 if we received an "stop" message
    // Won't be able to echo anymore
    if ( message == "stop" )
    {
        Serial.print("Unsubscribing from " TOPIC_SUBSCRIBE " ... ");
        mqtt.unsubscribe(TOPIC_SUBSCRIBE); // Will halt if fails
        Serial.println("OK");
    }
}

/*****
/*!
    @brief Connect to defined Access Point
*/
*****/
bool connectAP(void)
{
    // Attempt to connect to an AP
    Serial.print("Attempting to connect to: ");
    Serial.println(WLAN_SSID);

    if ( Feather.connect(WLAN_SSID, WLAN_PASS) )
    {
        Serial.println("Connected!");
    }
    else
    {
        Serial.printf("Failed! %s (%d)", Feather.errstr(), Feather.errno());
    }
}

```

```
    Serial.println();  
  }  
  Serial.println();  
  return Feather.connected();  
}
```

AdafruitMQTTTopic

AdafruitMQTT includes an **OPTIONAL** helper class called **AdafruitMQTTTopic** that can be used to publish data to a single topic on an MQTT broker.

This helper class inherits from [Print](https://adafru.it/IFk) (<https://adafru.it/IFk>), which allows you to write data to MQTT topics similarly to how you would write data to the 'Serial Monitor', using `.print` statements.

See 'MQTT/MqttTopicClass' in the examples folder for an example of how to use AdafruitMQTTTopic.

Constructor

```
AdafruitMQTTTopic(AdafruitMQTT& mqtt,  
                  const char* topic,  
                  uint8_t qos = MQTT_QOS_AT_MOST_ONCE,  
                  bool retain = false)
```

Parameters:

- **mqtt**: A reference to the AdafruitMQTT instance associated with this helper (since the connection to the MQTT broker is defined and managed there).
- **topic**: A null-terminated string containing the topic to publish to
- **qos**: An optional quality of server (QoS) level to use when publishing. If left empty, this argument will default to 'At Most Once', meaning it will try to publish the data but if the operation fails it won't persist the attempt and retry again later.
- **retain**: Sets the 'retain' bit to indicate if any messages published to the MQTT broker should be retained on the broker for the next client(s) that access that topic.

The following example shows how to properly declare an instance of the AdafruitMQTTTopic class (note that the default QoS and retain values are used):

```
#define CLIENTID      "Adafruit Feather"
#define TOPIC         "adafruit/feather"

AdafruitMQTT mqtt (CLIENTID);
AdafruitMQTTTopic publisher (mqtt, TOPIC);
```

Functions

In addition to the functions defined in the [Print base class \(https://adafru.it/IFk\)](https://adafru.it/IFk) (see the [Print.h source \(https://adafru.it/IFI\)](https://adafru.it/IFI) as well), the following functions are defined as part of AdafruitMQTTTopic:

```
void retain(bool on)
```

void retain (bool on)

Enables or disabled the 'retain' feature when publishing messages. This indicates whether the published message should be maintained on the broker when a message is written to the topic.

The default value for 'retain' is false, unless it is modified using this function.

Parameters:

- **on**: Whether or not the published message should be 'retained' by the MQTT broker. Sending a message with the this set to 'false' (0) will clear any previously retained message from the broker.

Returns: Nothing

Subscribe Callbacks

You can also subscribe or unsubscribe to publications on the topic using the following functions:

```
bool subscribe (messageHandler_t mh);
bool unsubscribe (void);
bool subscribed (void);
```

```
bool subscribe (messageHandler_t mh)
```

This function will subscribe to the topic and any changes will be sent to the specified callback handler.

Parameters:

- **mh:** The callback handler where the subscription event should be redirected to.

Returns: 'True' (1) is the subscribe was successful, otherwise 'false' (0).

Subscription callback handlers have the following format:

```

/*****
/ *!
  @brief  MQTT subscribe event callback handler

  @param  topic      The topic causing this callback to fire
  @param  message     The new value associated with 'topic'

  @note    'topic' and 'message' are UTF8Strings (byte array), which means
           they are not null-terminated like C-style strings. You can
           access its data and len using .data & .len, although there is
           also a Serial.print override to handle UTF8String data types.
*/
/*****
void subscribed_callback(UTF8String topic, UTF8String message)
{
  // Print out topic name and message
  Serial.printf("["); Serial.print(topic); Serial.printf("]");
  Serial.print(" : message = ") ;
  Serial.println(message);

  // Unsubscribe if message = "stop"
  if ( message == "stop" )
  {
    Serial.print("Unsubscribing ... ");
    mqttTopic.unsubscribe(); // Will halt if fails
    Serial.println("OK");
  }
}
}

```

```
bool unsubscribe (void)
```

Unsubscribes to the topic if you previously called `.subscribe`.

Parameters: None

Returns: 'True' (1) if the operation succeeded, otherwise 'false' (0).

bool subscribed (void)

Indicates whether you are currently susbscribed to this topic or not.

Parameters: None

Returns: 'True' (1) if you are subscribed, otherwise 'false' (0).

Publishing Data via 'Print'

One important thing to keep in mind with AdafruitMQTTTopic is that **every .print* function corresponds to an MQTT publication request.**

The following code will result in three different MQTT publications:

```
int    number_of_days = 7;
char*  place = "somewhere";

pub.print(number_of_days);
pub.print(" days since something happened ");
pub.print(place);
```

You can work around this '1 print = 1 publication' restriction by using the **printf** function, as shown in the example below:

```
int    number_of_days = 7;
char*  place = "somewhere";

pub.printf("%d days since something happened %s", number_of_days, place);
```

For a full list of printf modifiers (the special '%' character sequences that get replaced with variables after the main string) see [printf here \(https://adafru.it/IFm\)](https://adafru.it/IFm).

The most common modifiers are described below though (all preceded by '%' so '%d' for a signed decimal value, etc.) :

- **d** or **i**: Signed decimal value ('int', 'int16_t', etc.)
- **u**: unsigned decimal value ('uint32_t', etc.)
- **x**: lower-case hexadecimal integer (ex. 'a12b' for 0xA12B)
- **X**: upper-case hexadecimal integer (ex. 'A12B' for 0xA12B)
- **f**: floating point value ('float', etc.)
- **s**: null-terminated string of characters (ex. "sample")
- **c**: A single characters (ex. 'a')

Example

The following sketch shows how you might use AdafruitMQTTTopic in the real world. The latest source can be found in the **MQTT/MqttTopicClass** folder in 'examples'.

```
/******
This is an example for our Feather WIFI modules

Pick one up today in the adafruit shop!
******/
```

Adafruit invests time and resources providing this open source code,
please support Adafruit and open-source hardware by purchasing
products from Adafruit!

MIT license, check LICENSE for more information
All text above, and the splash screen below must be included in
any redistribution

```
*****/

#include <adafruit_feather.h>;
#include <adafruit_mqtt.h>;
#include "certificate_mosquitto.h"

/* This sketch connects to a public MQTT server (with/without TLS)
 * and publishes a message to a topic every 5 seconds.
 *
 * For server details see http://test.mosquitto.org/
 * - Port 1883 : MQTT, unencrypted
 * - Port 8883 : MQTT, encrypted (TLS)
 *
 * Note: may You need an MQTT desktop client such as
 * - The lightweight Java client included in this repo:
org.eclipse.paho.mqtt.utility-1.0.0.jar or
 * - A full desktop client like MQTT.fx https://learn.adafruit.com/desktop-mqtt-
client-for-adafruit-io/installing-software
 *
 * To run this demo
 * 1. Change WLAN_SSID/WLAN_PASS
 * 2. Decide whether you want to use TLS/SSL or not (USE_TLS)
 * 3. Change CLIENTID, TOPIC, PUBLISH_MESSAGE, WILL_MESSAGE if you want
 * 4. Compile and run
 * 5. Use your MQTT desktop client to connect to the same sever and subscribe
 * to the defined topic to monitor the published message(s).
 */

#define WLAN_SSID      "yourSSID"
#define WLAN_PASS      "yourPass"

#define USE_TLS        0

#define BROKER_HOST    "test.mosquitto.org"
#define BROKER_PORT    (USE_TLS ? 8883 : 1883 )

// Uncomment to set your own ClientID, otherwise a random ClientID is used
// #define CLIENTID      "Adafruit Feather"

#define TOPIC           "adafruit/feather"
#define WILL_MESSAGE    "Goodbye!!"

AdafruitMQTT      mqtt;
AdafruitMQTTTopic mqttTopic(&mqtt, TOPIC, MQTT_QOS_EXACTLY_ONCE);

char old_value = '0';
char value = '0';

/*****
 *!
 @brief The setup function runs once when the board comes out of reset
 */
/*****
void setup()
{
  Serial.begin(115200);

  // Wait for the USB serial port to connect. Needed for native USB port only
  while (!Serial) delay(1);

  Serial.println("MQTT Publish using Publisher Example\r\n");
}
```

```

// Print all software versions
Feather.printVersions();

while ( !connectAP() )
{
    delay(500); // delay between each attempt
}

// Connected: Print network info
Feather.printNetwork();

// Tell the MQTT client to auto print error codes and halt on errors
mqtt.err_actions(true, true);

// Set ClientID if defined
#ifdef CLIENTID
mqtt.clientID(CLIENTID);
#endif

// Last will must be set before connecting since it is part of the connection data
mqtt.will(TOPIC, WILL_MESSAGE, MQTT_QOS_AT_LEAST_ONCE);

// Connect to broker
Serial.printf("Connecting to " BROKER_HOST " port %d ... ", BROKER_PORT);
if (USE_TLS)
{
    // Disable default RootCA to save SRAM since we don't need to
    // access any other site except test.mosquitto.org
    Feather.useDefaultRootCA(false);

    // mosquitto CA is pre-generated using pycert.py
    Feather.addRootCA(rootca_certs, ROOTCA_CERTS_LEN);

    // Connect with SSL/TLS
    mqtt.connectSSL(BROKER_HOST, BROKER_PORT);
}
else
{
    mqtt.connect(BROKER_HOST, BROKER_PORT);
}
Serial.println("OK");

// Subscribe with callback
mqttTopic.subscribe(subscribed_callback);

Serial.println("Please use desktop client to subscribe to \"' TOPIC \"' to
monitor");

// Initial publish
Serial.printf("Publishing \'%d\' ... ", value);
mqttTopic.print( value ); //use .write to send in binary format
Serial.println("OK");
}

/*****
/*!
    @brief This loop function runs over and over again
*/
*****/
void loop()
{
    // value changed due to subscribed callback
    if (old_value != value)
    {
        // check if still subscribed
        if ( mqttTopic.subscribed() )
        {
            old_value = value;
            Serial.println();
            Serial.printf("Publishing \'%c\' ... \r\n", value);
        }
    }
}

```

```

        mqttTopic.print( value ); // use .write to send in binary format
    }
}
}

/*****
/!
@brief MQTT subscribe event callback handler

@param topic    The topic causing this callback to fire
@param message  The new value associated with 'topic'

@note 'topic' and 'message' are UTF8Strings (byte array), which means
they are not null-terminated like C-style strings. You can
access its data and len using .data & .len, although there is
also a Serial.print override to handle UTF8String data types.
*/
/*****/
void subscribed_callback(UTF8String topic, UTF8String message)
{
    // Copy received data to 'value'
    memcpy(&value, message.data, 1);

    // Print out topic name and message
    Serial.printf("["); Serial.print(topic); Serial.printf("]");
    Serial.print(" : value = " );
    Serial.println(value);

    // Increase value by 1
    value++;

    // wrap around
    if (value > '9') value = '0';

    // Unsubscribe if we received an "stop" message
    // Won't be able to echo anymore
    if ( message == "stop" )
    {
        Serial.print("Unsubscribing ... ");
        mqttTopic.unsubscribe(); // Will halt if fails
        Serial.println("OK");
    }
}

/*****
/!
@brief Connect to defined Access Point
*/
/*****/
bool connectAP(void)
{
    // Attempt to connect to an AP
    Serial.print("Attempting to connect to: ");
    Serial.println(WLAN_SSID);

    if ( Feather.connect(WLAN_SSID, WLAN_PASS) )
    {
        Serial.println("Connected!");
    }
    else
    {
        Serial.printf("Failed! %s (%d)", Feather.errstr(), Feather.errno());
        Serial.println();
    }
    Serial.println();

    return Feather.connected();
}

```

AdafruitAIO

AdafruitAIO is a special class that inherits from **AdafruitMQTT** (described earlier in this learning guide). It takes the core features from **AdafruitMQTT** and adds some helper functions that make working with [Adafruit IO](https://adafru.it/fsU) (<https://adafru.it/fsU>) easier.

If you're unfamiliar with Adafruit IO have a look at our introductory learning guide here: <https://learn.adafruit.com/adafruit-io>

Constructor

```
AdafruitAIO(const char* username, const char* password)
```

Parameters:

- **username:** The username associated with your Adafruit IO account (normally visible [here](https://adafru.it/dyy) (<https://adafru.it/dyy>)).
- **password:** The Adafruit IO key associated with your account. This is available by logging into Adafruit IO and clicking the yellow 'key' icon labelled 'Your secret AIO key'.

By default **AdafruitAIO** will generate a random 10..23 character string for the `ClientID`. If required you can override the default value via the `.clientId` function if it is called BEFORE the `.connect` or `.connectSSL` functions.

Functions

In addition to the functions defined in the **AdafruitMQTT** base class, The following functions are included as part of **AdafruitAIO**:

```
bool connect      ( bool cleanSession = true,
                    uint16_t keepalive_sec = MQTT_KEEPALIVE_DEFAULT )

bool connectSSL   ( bool cleanSession = true,
                    uint16_t keepalive_sec = MQTT_KEEPALIVE_DEFAULT )

bool updateFeed   ( const char* feed,
                    UTF8String message,
                    uint8_t qos=MQTT_QOS_AT_MOST_ONCE,
                    bool retain=true )
```

```
bool followFeed ( const char* feed,
                  uint8_t qos,
                  messageHandler_t mh )

bool unfollowFeed ( const char* feed )
```

Connecting

The following functions are available to connect to the Adafruit IO server:

```
bool connect (bool cleanSession = true, uint16_t
keepalive_sec = MQTT_KEEPALIVE_DEFAULT)
```

This function will attempt to connect to the Adafruit IO servers using a standard (unencrypted) connection.

Parameters:

- **cleanSession**: Indicates whether the client and broker should remember 'state' across restarts and reconnects. 'State' maintenance is based on the Client ID so be sure to set a reusable value via `.clientId` if you set `cleanSession` to `false`:
 - If set to **false** (0) both the client and server will maintain state across restarts of the client, the server and the connection. As state is maintained:
 - Message delivery will be reliable meeting the specified QOS even if the client, server or connection are restarted.
 - The server will treat a subscription as durable.
 - If set to **true** (1) the client and server will not maintain state across restarts of the client, the server or the connection. This means:
 - Message delivery to the specified QOS cannot be maintained if the client, server or connection are restarted
 - The server will treat a subscription as non-durable
- **keepalive_sec**: This value defines the maximum interval (in seconds) between messages being sent or received. Setting a value here ensures that at least one message is sent between the client and the broker within every 'keep alive' period. If no data was sent within 'keepalive_sec' seconds, the Client will send a simple ping to the broker to keep the connection alive. Setting this value to '0' disables the keep alive feature. **The default value is 60 seconds.**

Returns: 'True' (1) if the connection was successful, otherwise 'false' (0).

`bool connectSSL (bool cleanSession = true, uint16_t keepalive_sec = MQTT_KEEPALIVE_DEFAULT)`

This function will attempt to connect to the Adafruit IO servers using a secure (TLS/SSL) connection.

Parameters:

- **cleanSession:** Indicates whether the client and broker should remember 'state' across restarts and reconnects. 'State' maintenance is based on the Client ID so be sure to set a reusable value via `.clientId` if you set `cleanSession` to `false`!
 - If set to **false** (0) both the client and server will maintain state across restarts of the client, the server and the connection. As state is maintained:
 - Message delivery will be reliable meeting the specified QOS even if the client, server or connection are restarted.
 - The server will treat a subscription as durable.
 - If set to **true** (1) the client and server will not maintain state across restarts of the client, the server or the connection. This means:
 - Message delivery to the specified QOS cannot be maintained if the client, server or connection are restarted
 - The server will treat a subscription as non-durable
- **keepalive_sec:** This value defines the maximum interval (in seconds) between messages being sent or received. Setting a value here ensures that at least one message is sent between the client and the broker within every 'keep alive' period. If no data was sent within 'keepalive_sec' seconds, the Client will send a simple ping to the broker to keep the connection alive. Setting this value to '0' disables the keep alive feature. **The default value is 60 seconds.**

Returns: 'True' (1) if the connection was successful, otherwise 'false' (0).

Feed Management

The following functions are available to work with AIO feeds:

`bool updateFeed ( const char* feed, UTF8String message, uint8_t qos=MQTT_QOS_AT_MOST_ONCE, bool retain=true )`

Updates the value associated with the specified 'feed' ('topic' in MQTT terminology).

Parameters:

- **feed:** The feed to update, not including the 'username/feeds/' prefix. So to work with 'username/feeds/onoff' you should simply supply 'onoff' as the feedname.
- **qos:** The quality of service level (see the MQTT spec for details). Default = 'At Most Once', meaning the message tries to send once but isn't persisted if the send fails. Possible values are:
 - MQTT_QOS_AT_MOST_ONCE
 - MQTT_QOS_AT_LEAST_ONCE
 - MQTT_QOS_EXACTLY_ONCE
- **retained:** Whether or not the published message should be 'retained' by the MQTT broker. Sending a message with the retained bool set to 'false' (0) will clear any previously retained message from the broker. The default value is false.

Returns: 'True' (1) if the feed was successfully updated, otherwise 'false' (0).

bool followFeed (const char* feed, uint8_t qos, messageHandler_t mh)

Follows (or 'subscribes' in MQTT terminology) to the specified AIO feed, which will cause the specific callback handler function to fire every time the feed is changed on the AIO server.

Parameters:

- **feed:** The feed to follow, not including the 'username/feeds/' prefix. So to work with 'username/feeds/onoff' you should simply supply 'onoff' as the feedname.
- **qos:** The quality of service level (see the MQTT spec for details). Default = 'At Most Once', meaning the message tries to send once but isn't persisted if the send fails. Possible values are:
 - MQTT_QOS_AT_MOST_ONCE
 - MQTT_QOS_AT_LEAST_ONCE
 - MQTT_QOS_EXACTLY_ONCE
- **mh:** The callback handler function to fire whenever the feed is changed. The callback handler should have the following signature:

```
/*  
*****  
*/  
/*!  
  @brief  'follow' event callback handler  
  
  @param  message    The new value associated with this feed
```

```

    @note    'message' is a UTF8String (byte array), which means
            it is not null-terminated like C-style strings. You can
            access its data and len using .data & .len, although there is
            also a Serial.print override to handle UTF8String data types.
*/
/*****
void feed_callback(UTF8String message)
{
    // Print message
    Serial.println(message);
}

```

Returns: 'True' (1) if the follow operation was successful, otherwise 'false' (0).

bool unfollowFeed (const char* feed)

Unfollows (or 'unsubscribes' in MQTT terminology) to the specified feed.

Parameters:

- **feed:** The feed to update, not including the 'username/feeds/' prefix. So to work with 'username/feeds/onoff' you should simply supply 'onoff' as the feedname.

Returns: 'True' (1) if the operation was successful, otherwise 'false' (0).

Example

The following example show how you can use the AdafruitAIO class to communicate with the Adafruit IO servers:

```

/*****
This is an example for our Feather WIFI modules

Pick one up today in the adafruit shop!

Adafruit invests time and resources providing this open source code,
please support Adafruit and open-source hardware by purchasing
products from Adafruit!

MIT license, check LICENSE for more information
All text above, and the splash screen below must be included in
any redistribution
*****/

#include <adafruit_feather.h>;
#include <adafruit_mqtt.h>;
#include "certificate_mosquitto.h"

/* This sketch demonstrates subscribe/unsubscribe activity with
 * callbacks.
 *
 * It will connect to a public MQTT server (with/without TLS)
 * and subscribe to TOPIC_SUBSCRIBE (defined below).
 *
 * - When a message is received, it will echo back to TOPIC_ECHO
 * - If the received message is "stop", we will
 *   unsubscribe from TOPIC_SUBSCRIBE and you won't be able to
 *   echo content back to the broker any longer.

```

```

*
* Note: TOPIC_SUBSCRIBE and TOPIC_ECHO must not be the same topic!
* Ex. They must not be "adafruit/+" and "adafruit/echo", since this will
* cause an infinite loop (received -> echo -> received -> ....)
*
* For details on the MQTT broker server see http://test.mosquitto.org/
* - Port 1883 : MQTT, unencrypted
* - Port 8883 : MQTT, encrypted (TLS)
*
* Note: may You need an MQTT desktop client such as the lightweight
* Java client included in this repo: org.eclipse.paho.mqtt.utility-1.0.0.jar
*
* For information on configuring your system to work with MQTT see:
* - https://learn.adafruit.com/desktop-mqtt-client-for-adafruit-io/installing-software
*
* To run this demo
* 1. Change the WLAN_SSID/WLAN_PASS to match your access point
* 2. Decide whether you want to use TLS/SSL or not (USE_TLS)
* 3. Change TOPIC*, WILL*, enable CLIENTID if needed
* 4. Compile and run
* 5. Use an MQTT desktop client to connect to the same MQTT broker and
*    publish to any topic beginning with "adafruit/feather/" (depending
*    on TOPIC_SUBSCRIBE). To be able to receive the echo message, please
*    also subscribe to "adafruit/feather_echo" (TOPIC_ECHO).
*/

#define WLAN_SSID      "yourSSID"
#define WLAN_PASS      "yourPass"

#define USE_TLS        0

#define BROKER_HOST    "test.mosquitto.org"
#define BROKER_PORT    (USE_TLS ? 8883 : 1883)

// Uncomment to set your own ClientID, otherwise a random ClientID is used
// #define CLIENTID      "Adafruit Feather"

#define TOPIC_SUBSCRIBE "adafruit/feather/+"
#define TOPIC_ECHO      "adafruit/feather_echo"

#define WILL_TOPIC      "adafruit/feather"
#define WILL_MESSAGE    "Goodbye!!"

AdafruitMQTT mqtt;

/*****
 *!
 * @brief Disconnect handler for MQTT broker connection
 */
/*****
void disconnect_callback(void)
{
    Serial.println();
    Serial.println("-----");
    Serial.println("DISCONNECTED FROM MQTT BROKER");
    Serial.println("-----");
    Serial.println();
}

/*****
 *!
 * @brief The setup function runs once when the board comes out of reset
 */
/*****
void setup()
{
    Serial.begin(115200);

```

```

// Wait for the USB serial port to connect. Needed for native USB port only
while (!Serial) delay(1);

Serial.println("MQTT Subscribe Example\r\n");

// Print all software versions
Feather.printVersions();

while ( !connectAP() )
{
    delay(500); // delay between each attempt
}

// Connected: Print network info
Feather.printNetwork();

// Tell the MQTT client to auto print error codes and halt on errors
mqtt.err_actions(true, true);

// Set ClientID if defined
#ifdef CLIENTID
mqtt.clientID(CLIENTID);
#endif

// Last will must be set before connecting since it is part of the connection data
mqtt.will(WILL_TOPIC, WILL_MESSAGE, MQTT_QOS_AT_LEAST_ONCE);

// Set the disconnect callback handler
mqtt.setDisconnectCallback(disconnect_callback);

Serial.printf("Connecting to " BROKER_HOST " port %d ... ", BROKER_PORT);
if (USE_TLS)
{
    // Disable default RootCA to save SRAM since we don't need to
    // access any other site except test.mosquitto.org
    Feather.useDefaultRootCA(false);

    // mosquitto CA is pre-generated using pycert.py
    Feather.addRootCA(rootca_certs, ROOTCA_CERTS_LEN);

    // Connect with SSL/TLS
    mqtt.connectSSL(BROKER_HOST, BROKER_PORT);
}
else
{
    mqtt.connect(BROKER_HOST, BROKER_PORT);
}
Serial.println("OK");

Serial.print("Subscribing to " TOPIC_SUBSCRIBE " ... ");
mqtt.subscribe(TOPIC_SUBSCRIBE, MQTT_QOS_AT_MOST_ONCE, subscribed_callback); //
Will halted if an error occurs
Serial.println("OK");
}

/*****
/*!
    @brief This loop function runs over and over again
*/
/*****/
void loop()
{

}

/*****
/*!
    @brief MQTT subscribe event callback handler
    @param topic      The topic causing this callback to fire

```

```

    @param message    The new value associated with 'topic'

    @note    'topic' and 'message' are UTF8Strings (byte array), which means
             they are not null-terminated like C-style strings. You can
             access its data and len using .data & .len, although there is
             also a Serial.print override to handle UTF8String data types.
*/
/*****
void subscribed_callback(UTF8String topic, UTF8String message)
{
    // Print out topic name and message
    Serial.print("[Subscribed] ");
    Serial.print(topic);
    Serial.print(" : ");
    Serial.println(message);

    // Echo back
    Serial.print("Echo back to " TOPIC_ECHO " ... ");
    mqtt.publish(TOPIC_ECHO, message); // Will halt if an error occurs
    Serial.println("OK");

    // Unsubscribe from SUBSCRIBED_TOPIC2 if we received an "stop" message
    // Won't be able to echo anymore
    if ( message == "stop" )
    {
        Serial.print("Unsubscribing from " TOPIC_SUBSCRIBE " ... ");
        mqtt.unsubscribe(TOPIC_SUBSCRIBE); // Will halt if fails
        Serial.println("OK");
    }
}

/*****
/*!
    @brief    Connect to defined Access Point
*/
/*****
bool connectAP(void)
{
    // Attempt to connect to an AP
    Serial.print("Attempting to connect to: ");
    Serial.println(WLAN_SSID);

    if ( Feather.connect(WLAN_SSID, WLAN_PASS) )
    {
        Serial.println("Connected!");
    }
    else
    {
        Serial.printf("Failed! %s (%d)", Feather.errstr(), Feather.errno());
        Serial.println();
    }
    Serial.println();

    return Feather.connected();
}

```

AdafruitAIOWFeed

AdafruitAIOWFeed is an optional helper class based on **AdafruitMQTTTopic**. It aims to make working with feeds in Adafruit IO a bit easier, with the goal of implementing specialised classes that correspond to AIO feed types in the future.

Be sure to look at the documentation for `AdafruitMQTTTopic` as well, since this class is a specialized version of that aimed at Adafruit IO.

Constructor

`AdafruitAIOFeed` uses the following constructor:

```
AdafruitAIOFeed(AdafruitAIO* aio,
                const char* feed,
                uint8_t qos = MQTT_QOS_AT_MOST_ONCE,
                bool retain = true)
```

Parameters:

- **aio**: A reference to the **AdafruitAIO** class instance, which will be used when sending and receiving data to the AIO server.
- **feed**: A string containing the name of the AIO feed to work with, minus the 'username/feeds/' text which will be automatically added by this class. For example, to work with 'testuser/feeds/status' you would provide 'status' to the feed parameter.
- **qos**: An optional quality of server (QoS) level to use when publishing. If left empty, this argument will default to 'At Most Once', meaning it will try to publish the data but if the operation fails it won't persist the attempt and retry again later.
- **retain**: Sets the 'retain' bit to indicate if any messages published to the MQTT broker should be retained on the broker for the next client(s) that access that topic. By default this will be set to 'true' for AIO feeds.

Functions

The following functions are defined as part of `AdafruitAIOFeed`, but you also have access to the public functions that are defined in **AdafruitMQTTTopic** since `AdafruitAIOFeed` class inherits from it.

```
bool follow    ( feedHandler_t fp )
bool unfollow  ( void )
bool followed  ( void )

virtual size_t write ( const uint8_t *buf, size_t len )
virtual size_t write ( uint8_t ch )
```

bool follow (feedHandler_t fp)

Enables you to 'follow' this feed, meaning that you subscribe to any changes that are published to this feed on the AIO server. To follow the feed, you simply set the callback handler, which is the function that will be called when this feed changes in AIO.

Parameters:

- **fp:** The callback handler function that will be fired when the feed changes on the AIO server. This function should have the following signature:

The name of the callback handler function can be set to anything you like, although the parameters and return type must be identical.

```
void feed_callback(UTF8String message)
{
  Serial.println(message);
}
```

Returns: 'True' (1) if the operation was successful, otherwise 'false' (0).

bool unfollow (void)

Calling this function will stop the follow callback and unsubscribe from the feed, meaning any changes will no longer be received by this class.

Parameters: None

Returns: 'True' (1) if the operation was successful, otherwise 'false' (0).

bool followed (void)

Checks whether 'follow' is currently enabled or not (indicate whether or not we are subscribed to the AIO feed).

Parameters: None

Returns: 'True' (1) if the operation was successful, otherwise 'false' (0).

Example

For more examples of working with AdafruitIO and AdafruitIOFeed see the **/AIO** folder in /examples in the WICED Feather board support package.

```

/*****
  This is an example for our Feather WIFI modules

  Pick one up today in the adafruit shop!

  Adafruit invests time and resources providing this open source code,
  please support Adafruit and open-source hardware by purchasing
  products from Adafruit!

  MIT license, check LICENSE for more information
  All text above, and the splash screen below must be included in
  any redistribution
  *****/

#include <adafruit_feather.h>;
#include <adafruit_mqtt.h>;
#include <adafruit_aio.h>;

/* This sketch connects to the Adafruit IO server at io.adafruit.com
 * and updates a 'PHOTOCELL_FEED' every 5 seconds.
 *
 * It also follow 'ONOFF_FEED' to receive updates from the AIO server via
 * the built-in follow/subscribe callback handler.
 *
 * To run this demo
 * 1. Change WLAN_SSID/WLAN_PASS
 * 2. Decide whether you want to use TLS/SSL or not (USE_TLS)
 * 3. Change AIO_USERNAME, AIO_KEY to match your own account details
 * 4. If you want, change PHOTOCELL_FEED and ONOFF_FEED to use different feeds
 * 5. Compile and run
 * 6. Optionally log into the AIO webserver to see any changes in data, etc.
 */

#define WLAN_SSID      "yourSSID"
#define WLAN_PASS      "yourPass"

#define AIO_USERNAME    "...your AIO username (see https://accounts.adafruit.com)..."
#define AIO_KEY          "...your AIO key..."

// AdafruitAIO will auto append the "username/feeds/" prefix to your feed(s)
#define PHOTOCELL_FEED  "photocell"
#define ONOFF_FEED      "onoff"

// Connect using TLS/SSL or not
#define USE_TLS          0

// Uncomment to set your own ClientID, otherwise a random ClientID is used
// #define CLIENTID      "Adafruit Feather"

AdafruitAIO      aio(AIO_USERNAME, AIO_KEY);
AdafruitAIOFeed  photocell (&aio, PHOTOCELL_FEED);
AdafruitAIOFeed  onoff      (&aio, ONOFF_FEED);

int value = 0;

/*****
  /*!
   * @brief The setup function runs once when the board comes out of reset
   */
  *****/
```

```

void setup()
{
  Serial.begin(115200);

  // Wait for the USB serial port to connect. Needed for native USB port only
  while (!Serial) delay(1);

  Serial.println("AIO Test Example\r\n");

  // Print all software versions
  Feather.printVersions();

  while ( !connectAP() )
  {
    delay(500); // delay between each attempt
  }

  // Connected: Print network info
  Feather.printNetwork();

  // Tell the MQTT client to auto print error codes and halt on errors
  aio.err_actions(true, true);

  // Set ClientID if defined
  #ifdef CLIENTID
  aio.clientID(CLIENTID);
  #endif

  Serial.print("Connecting to io.adafruit.com ... ");
  if ( USE_TLS )
  {
    aio.connectSSL(); // Will halted if an error occurs
  }else
  {
    aio.connect(); // Will halted if an error occurs
  }
  Serial.println("OK");

  // 'Follow' the onoff feed to capture any state changes
  onoff.follow(feed_callback);
}

/*****
/*!
  @brief This loop function runs over and over again
  */
*****/
void loop()
{
  value = (value+1) % 100;

  Serial.print("Updating feed " PHOTOCELL_FEED " : ");
  Serial.print(value);
  photocell.print(value);
  Serial.println(" ... OK");

  delay(5000);
}

/*****
/*!
  @brief 'follow' event callback handler

  @param message The new value associated with this feed

  @note 'message' is a UTF8String (byte array), which means
  it is not null-terminated like C-style strings. You can
  access its data and len using .data & .len, although there is
  also a Serial.print override to handle UTF8String data types.
  */
*****/

```

```

*/
/*****
void feed_callback(UTF8String message)
{
    // Print message
    Serial.print("[ONOFF Feed] : ");
    Serial.println(message);
}

/*****
/*!
    @brief  Connect to defined Access Point
*/
/*****
bool connectAP(void)
{
    // Attempt to connect to an AP
    Serial.print("Please wait while connecting to: '" WLAN_SSID "' ... ");

    if ( Feather.connect(WLAN_SSID, WLAN_PASS) )
    {
        Serial.println("Connected!");
    }
    else
    {
        Serial.printf("Failed! %s (%d)", Feather.errstr(), Feather.errno());
        Serial.println();
    }
    Serial.println();

    return Feather.connected();
}

```

AdafruitTwitter

The AdafruitTwitter class requires WICED Feather Lib 0.5.5 or higher to run.

The AdafruitTwitter class makes sending tweets easy via a custom Application that you can setup using this learning guide.

1. Creating a WICED Twitter Application

In order to enable WICED to interact with Twitter, you first need to log in to twitter's app admin console at <http://apps.twitter.com> (<https://adafru.it/qof>) and create a new app:

Twitter Apps

You don't currently have any Twitter Apps.

Create New App

Enter the Application Details

Next you need to enter your application details, based on the following data:

The NAME field must be globally unique, so you should make it something personal like adding your twitter username.

Create an application

Application Details

Name *

Adafruit_WICED_Feather

Your application name. This is used to attribute the source of a tweet and in user-facing authorization screens. 32 characters max.

Description *

Adafruit WICED Feather

Your application description, which will be shown in user-facing authorization screens. Between 10 and 200 characters max.

Website *

https://www.adafruit.com/product/3056

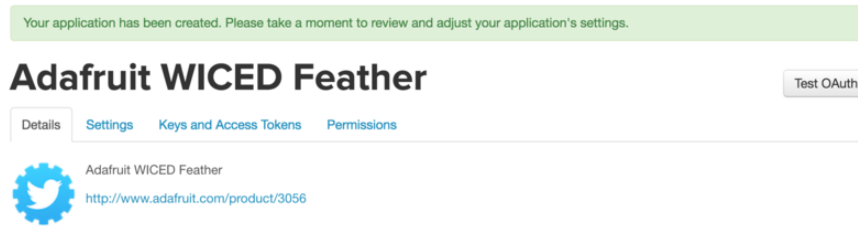
Your application's publicly accessible home page, where users can go to download, make use of, or find out more information. A qualified URL is used in the source attribution for tweets created by your application and will be shown in user-facing authorization screens. (If you don't have a URL yet, just put a placeholder here but remember to change it later.)

Callback URL

Where should we return after successfully authenticating? OAuth 1.0a applications should explicitly specify their oauth_callback regardless of the value given here. To restrict your application from using callbacks, leave this field blank.

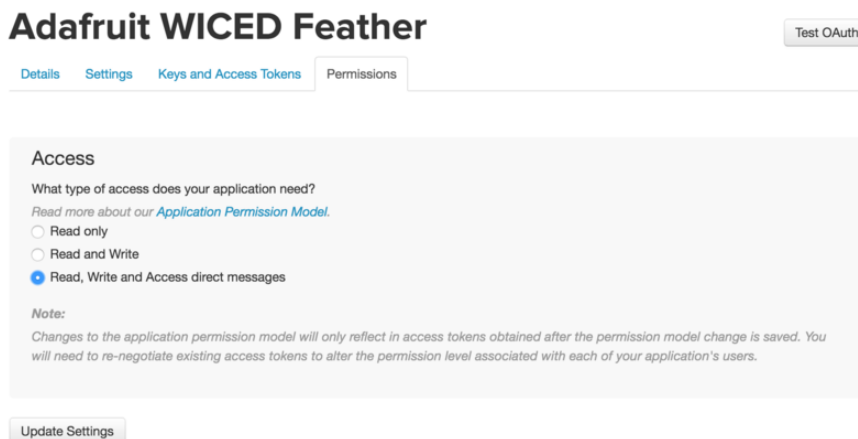
Then accept the license terms and click the **Create your Twitter application** button at the bottom of the page.

This will redirect you to the main app config page, as shown below:



Set the Application Permissions

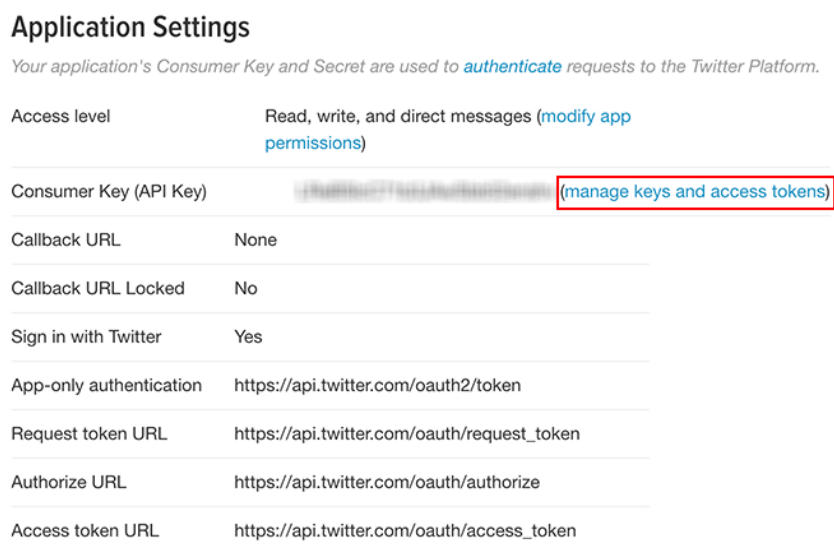
Click on the **Permissions** tab and set the appropriate permissions:



Click the **Update Settings** button to save the permissions changes.

Manage the Access Keys

Go back to the **Details** tab and scroll down to the **Application Settings** section:



Click the **manage keys and access tokens** link.

Copy the Appropriate Key Data

Make a note of the consumer key values blurred out below since you will need them in your sketch:

Application Settings

Keep the "Consumer Secret" a secret. This key should never be human-readable in your application.

Consumer Key (API Key)	[blurred]
Consumer Secret (API Secret)	[blurred]
Access Level	Read, write, and direct messages (modify app permissions)
Owner	[blurred]
Owner ID	[blurred]

Create your Access Token

On the same page shown above, click the **Create my access token** button to give your account access to your new application:

Your Access Token

You haven't authorized this application for your own account yet.

By creating your access token here, you will have everything you need for your application's current permission level.

Token Actions

Create my access token

Make a note of the access token data shown blurred out below, which you will also need in your sketch:

Your Access Token

This access token can be used to make API requests on your own account's behalf. Do not share your access token secret with anyone.

Access Token	[blurred]
Access Token Secret	[blurred]
Access Level	Read, write, and direct messages
Owner	[blurred]
Owner ID	[blurred]

2. Using the AdafruitTwitter Class

Next, open the **Applications/SendTweet** example for WICED or create a new sketch with the following code, updating it with your access point details, as well as the **Consumer** and **Access Tokens** generated above:

```
/******
This is an example for our Feather WIFI modules

Pick one up today in the adafruit shop!

Adafruit invests time and resources providing this open source code,
please support Adafruit and open-source hardware by purchasing
products from Adafruit!

MIT license, check LICENSE for more information
All text above, and the splash screen below must be included in
any redistribution
*****/

#include "adafruit_feather.h"
#include "adafruit_http.h"
#include "adafruit_twitter.h"

/* This example demonstrates how to use AdafruitTwitter class
 * to send out a tweet
 *
 * To run this demo:
 * 1. Goto https://apps.twitter.com/ and login
 * 2. Create an application to use with this WICED Feather
 * 3. (Optional) You could change the access level to give the application
 * permission to send DM. It is advised to do so, so that you could use WICED
 * to send DM in other example
 * 4. In the app management click "manage keys and access tokens"
 * and then click "Create my access token"
 * 5. Change CONSUMER_KEY, CONSUMER_SECRET, TOKEN_ACCESS, TOKEN_SECRET accordingly
 * 6. Change your TWEET status
 * 7. Compile and run, if you run this sketch too often, Twitter server may reject
 * your connection request, just wait a few minutes and try again.
 */

// Network
#define WLAN_SSID          "yourSSID"
#define WLAN_PASS          "yourPassword"

// Twitter Account
#define CONSUMER_KEY       "YOUR_CONSUMER_KEY"
#define CONSUMER_SECRET    "YOUR_CONSUMER_SECRET"

#define TOKEN_ACCESS       "YOUR_TOKEN_ACCESS"
#define TOKEN_SECRET       "YOUR_TOKEN_SECRET"

#define TWEET              "Hello from Adafruit WICED Feather"

AdafruitTwitter Twitter;

/******
 *!
  @brief The setup function runs once when reset the board
 */
*****/
void setup()
{
  Serial.begin(115200);
```

```

// wait for serial port to connect. Needed for native USB port only
while (!Serial) delay(1);

Serial.println("Twitter Send Tweet Example\r\n");

// Print all software versions
Feather.printVersions();

while ( !connectAP() )
{
    delay(500); // delay between each attempt
}

// Connected: Print network info
Feather.printNetwork();

Twitter.begin(CONSUMER_KEY, CONSUMER_SECRET, TOKEN_ACCESS, TOKEN_SECRET);
Twitter.err_actions(true, true);

Serial.print("Sending tweet: " TWEET " ... ");
Twitter.tweet(TWEET);
Serial.println("OK");

Twitter.stop();
}

/*****
 *!
  @brief The loop function runs over and over again forever
 */
/*****/
void loop()
{
    togglePin(PA15);
    delay(1000);
}

/*****
 *!
  @brief Connect to defined Access Point
 */
/*****/
bool connectAP(void)
{
    // Attempt to connect to an AP
    Serial.print("Please wait while connecting to: '" WLAN_SSID "' ... ");

    if ( Feather.connect(WLAN_SSID, WLAN_PASS) )
    {
        Serial.println("Connected!");
    }
    else
    {
        Serial.printf("Failed! %s (%d)", Feather.errstr(), Feather.errno());
        Serial.println();
    }
    Serial.println();

    return Feather.connected();
}

```

AdafruitSDEP

All communication between the Arduino user code (your sketch) and the lower level WiFi stack from Broadcom happens over SDEP commands.

SDEP stands for 'Simple Data Exchange Protocol', an in house protocol we use in a number of our products.

This is similar to the way you would talk to an external I2C or SPI sensor via a set of pre-defined registers defined in the sensor datasheet. You send specifically formatted data to known registers (or addresses), and sometimes you get data back in a known format (depending on the command).

SDEP is the **s**imple **d**ata **e**xchange **p**rotocol that we use for the command and response messages between the user code and the lower level Feather Lib that contains the WICED WiFi stack.

Normally you won't need to deal with SDEP commands yourself since these are hidden in the AdafruitFeather, AdafruitHTTP, etc., helper classes, but a specialized helper classed name **AdafruitSDEP** is available to send SDEP commands yourself and get the response data back if the need should ever arise to talk directly to the WICED stack yourself.

AdafruitSDEP API

```
// Send a simple SDEP command (1 parameter value or less)
bool sdep (uint16_t cmd_id
           , uint16_t param_len
           , void const* p_param
           , uint16_t* p_result_len
           , void* p_result);

// Send a complex SDEP command (multiple parameter values)
bool sdep_n (uint16_t cmd_id
            , uint8_t para_count
            , sdep_cmd_para_t const* para_arr
            , uint16_t* p_result_len
            , void* p_result);

// SDEP error handling functions
err_t      errno      (void);
char const* errstr     (void);
char const* cmdstr     (uint16_t cmd_id);
void        err_actions (bool print, bool halt);
```

Constructor

AdafruitFeather inherits from AdafruitSDEP, meaning that you don't need to instantiate AdafruitSDEP directly yourself. Simply call the functions described below from your AdafruitFeather class instance, which is normally available as 'Feather', so 'Feather.sdep(...)', 'Feather.sdep_n(...)', 'Feather.errno()', etc.

Functions

The following functions and parameters are present in AdafruitSDEP:

sdep

This function sends an SDEP command with up to one parameter (or no parameters if NULL is provided in the 'p_param' field or 'param_len' is set to 0).

Function Prototype:

```
bool sdep(uint16_t cmd_id      ,
          uint16_t param_len   , void const* p_param,
          uint16_t* p_result_len, void* p_result)
```

Parameters:

- **cmd_id**: The 16-bit SDEP command ID
- **param_len**: The length of the **p_param** field containing the parameter data. Set this to '0' if no parameter is provided.
- **p_param**: A pointer to the parameter value to pass into the SDEP command handler. Set this to NULL if no parameter is provided.
- **p_result_len**: A pointer to the 16-bit value where the response length will be written by the SDEP command handler
- **p_result**: A pointer to where the response data should be written by the SDEP command handler

Return Value:

'true' if the function executed properly, otherwise 'false' if an error occurred (check .errno or .errstr for details).

Examples

The simplest possible example of using this function can be seen below.

No parameter data is sent to the SDEP command, we don't check any response data (there is none from **SDEP_CMD_FACTORYRESET** anyway), and we don't even check if 'sdep' returned false to indicate that there was an error executing the command:

```
void AdafruitFeather::factoryReset(void)
{
    sdep(SDEP_CMD_FACTORYRESET, 0, NULL, NULL, NULL);
}
```

A more complex example of sending a simple SDEP command with this function can be seen below, where we flush the contents of the TCP buffer.

'_tcp_handle' is an internal 32-bit value (so 4 bytes), and we pass a pointer to the value to the SDEP command handler (notice the '&' symbol before the name saying that we should pass the address in memory for '_tcp_handle').

No response data is read back, so the last two parameters are set to NULL.

```
void AdafruitTCP::flush()
{
    if ( _tcp_handle == 0 ) return;

    // flush write
    sdep(SDEP_CMD_TCP_FLUSH, 4, &_tcp_handle, NULL, NULL);
}
```

This last example checks if any TCP data is available in the buffer, and the command will set the 'result' variable to a non-zero value if any data is available.

Since we know the size of the results variable, we don't need to read back the length of the response data, and we can insert NULL for 'p_result_len':

```
int AdafruitTCP::available()
{
    if ( _tcp_handle == 0 ) return 0;

    uint32_t result = 0;
    sdep(SDEP_CMD_TCP_AVAILABLE, 4, &_tcp_handle, NULL, &result);

    return result;
}
```

sdep_n

This function sends an SDEP command with an array of parameter values, using a dedicated parameter array typedef called **sdep_cmd_para_t**.

Function Prototype:

```
bool sdep_n(uint16_t cmd_id      ,
            uint8_t  para_count , sdep_cmd_para_t const* para_arr,
            uint16_t* p_result_len, void* p_result)
```

Parameters:

- **cmd_id**: The 16-bit SDEP command ID
- **para_count**: The number of parameters in **para_arr**
- **para_arr**: An array of **sdep_cmd_para_t** values, consisting of a 16-bit length value and a pointer to the actual parameter data

- **p_results_len**: A pointer to the 16-bit value where the response length will be written by the SDEP command handler
- **p_result**: A pointer to where the response data should be written by the SDEP command handler

Each entry in **para_arr** has the following structure:

```
typedef struct {
    uint16_t    len;
    void const* p_value;
} sdep_cmd_para_t;
```

Return Value:

'true' if the function executed properly, otherwise 'false' if an error occurred (check `.errno` or `.errstr` for details).

Examples

The example below uses the **SDEP_CMD_WIFI_PROFILE_ADD** command to store the connection details to non-volatile memory.

This is a blocking command that only returns when the procedure succeeds or fails. As such, we will ignore any return data from the command other than a possible SDEP error code. As such, **p_results_len** and **p_result** are both set to **NULL** here:

```
bool AdafruitFeather::addProfile(char* ssid)
{
    sdep_cmd_para_t para_arr[] =
    {
        { .len = strlen(ssid), .p_value = ssid },
    };
    uint8_t para_count = sizeof(para_arr)/sizeof(sdep_cmd_para_t);

    return sdep_n(SDEP_CMD_WIFI_PROFILE_ADD, para_count, para_arr,
                  NULL, NULL);
}
```

A more complex example is shown below where we read the SDEP response, and a pointer to certain parameter values is also used (noticed the '&' character below some parameter values). The use of pointers is necessary when passing large or complex parameters to the SDEP command handler.

In this particular example we use **SDEP_CMD_TCP_READ** but we also want to read the response data.

```
int AdafruitTCP::read(uint8_t* buf, size_t size)
{
    if ( _tcp_handle == 0 ) return 0;
```

```

uint16_t size16 = (uint16_t) size;
sdep_cmd_para_t para_arr[] =
{
    { .len = 4, .p_value = &_tcp_handle },
    { .len = 2, .p_value = &size16 },
    { .len = 4, .p_value = &_timeout },
};
uint8_t para_count = sizeof(para_arr)/sizeof(sdep_cmd_para_t);

uint16_t readlen = size16;
VERIFY_RETURN( sdep_n(SDEP_CMD_TCP_READ, para_count, para_arr, &readlen, buf),
0);

_bytesRead += readlen;
return readlen;
}

```

We pass in three parameters to **SDEP_CMD_TCP_READ**:

- The TCP handle (**_tcp_handle**)
- The number of bytes we want to read (**size16**)
- The timeout before returning an error (**_timeout**)

The command will then return the data that was read back, populating the **buf** and **size16** fields. The 'size16' field will contain the numbers of bytes written to 'buf' so that we can compare the numbers of bytes requested with the number of bytes actually read out.

The VERIFY macro in the example above is simply a helper to check the response from sdep_n, and it will return '0' if an error was encountered.

Error Handling Functions

The following functions are defined to work with any SDEP errors generated by the system:

err_t errno (void)

If **sdep** or **sdep_n** returned **false** as a return value, it means the SDEP command failed. To determine the error message, you can read the results from **.errno()** immediately after the **.sdep** or **.sdep_n** command, which will give you a 16-bit (uint16_t) error code.

char const* errstr(void)

To provide further details on the value returned in **errno** you can also call **.errstr()** which will return a char array containing the internam enum name for the last error code.

Unfortunately, for copyright reasons we're not able to release the Broadcom WICED WiFi stack source, but seeing the string associated with your **errno** provides an excellent indicator of what went wrong executing the SDEP command.

char const* cmdstr (uint16_t cmd_id)

Returns the name of the command associated with the specified SDEP command ID.

Parameters:

- **cmd_id**: The 16-bit SDEP command ID to lookup (based on **.errno**, for example)

Returns: A string representing the name of the SDEP command associated with 'cmd_id'.

void err_actions (bool print, bool halt)

This function allows you to enable various optional 'actions' that should be automatically taken when an SDEP error occurs. By default all actions are disabled.

Parameters:

- **print**: If set to true, any SDEP error will be displayed in the Serial Monitor via **Serial.print**, including both the **.errstr** and **.errno** values. This can help keep your code clean and make it easier to switch between debug and release mode.
- **halt**: If set to true, the code will stop executing and wait in a 'while(1)' loop as soon as an SDEP error is encountered.

Returns: Nothing

Error Handling Example

The following example shows an example of how you can use the **.errno** and **.errstr** functions to handle the last SDEP error generated by the system:

```
// Attempt to connect to the AP
if ( Feather.connect("SSID", "PASSWORD", ENC_TYPE_AUTO ) )
{
    int8_t rssi = Feather.RSSI();
    uint32_t ipAddress = Feather.localIP();
    // Do something now that you are connected to the AP!
```

```

}
else
{
  // Display the error message
  err_t err = Feather.errno();
  Serial.println("Connection Error:");
  switch (err)
  {
    case ERROR_WWD_ACCESS_POINT_NOT_FOUND:
      // SSID wasn't found when scanning for APs
      Serial.println("Invalid SSID");
      break;
    case ERROR_WWD_INVALID_KEY:
      // Invalid SSID passkey
      Serial.println("Invalid Password");
      break;
    default:
      // The most likely cause of errors at this point is that
      // you are just out of the device/AP operating range
      Serial.print(Feather.errno());
      Serial.print(":");
      Serial.println(Feather.errstr());
      break;
  }
}
}

```

Client

The WICED Feather supports the standard [Arduino Client \(https://adafru.it/IFj\)](https://adafru.it/IFj) interface that is used by many networking boards in the Arduino ecosystem.

Adapting Client Examples

Most existing Client based examples can easily be adapted to work with the WICED Feather board family if the following changes are made to the sketches:

1. Update Header Includes

You will need to change the default WiFi (etc.) headers to the Adafruit versions, as shown below.

Remove Existing Headers

Existing headers like 'WiFi.h', 'WiFiUDP.h', etc., should be removed from the top of your sketch.

For example ...

```

#include <WiFi.h>;
#include <WiFiUdp.h>;
#include <WiFiTcp.h>;

```

Add Adafruit WICED Feather Header

... should be replaced with the single 'adafruit_feather.h' header file:

```
#include <adafruit_feather.h>;
```

Only one header is required with the WICED Feather board family, since the key related headers are also referenced in that one file.

2. Change 'WiFi.*' References to 'Feather.*'

References to functions like **WiFi.begin(ssid, pass)** or **WiFi.available()** should be replaced with **Feather.begin(ssid, pass)** or **Feather.available()**:

Previous Client Code

```
// Attempt to connect to Wifi network:
while (status != WL_CONNECTED) {
  Serial.print("Attempting to connect to SSID: ");
  Serial.println(ssid);
  // Connect to WPA/WPA2 network. Change this line if using open or WEP network:
  status = WiFi.begin(ssid, pass);

  // wait 10 seconds for connection:
  delay(10000);
}
```

WICED Feather Client Code

Note that at present **.begin** in the WICED Feather library returns a bool, not a status byte (as in the WiFi example above), so the example has been modified slightly to detect connection status via the **.connected** (<https://adafru.it/IFn>) function that is also part of the **Client** (<https://adafru.it/IFj>) interface.

The Adafruit WICED Feather API is still a work in progress and we're trying to make the transition to the WICED as easy as possible, but there may be some implementation differences between platforms. Hopefully these will be addressed over time.

```
// Attempt to connect to Wifi network
while (!Feather.connected()) {
  Serial.print("Attempting to connect to SSID: ");
  Serial.println(ssid);
  // Connect to any network.
  // The Feather stack will try to determine the network
  // security type automatically
  bool results = Feather.begin(ssid, pass);
}
```

```
// Optional: wait a bit before checking for a connection
delay(3000);
}
```

3. Change WiFiUDP and WiFiTCP Class Types

If your example uses classes like **WiFiUDP** and **WiFiTCP**, simply replace the class names with **AdafruitUDP** or **AdafruitTCP**.

Existing WiFiUDP Class

```
// A UDP instance to let us send and receive packets over UDP
WiFiUDP Udp;
```

Updated AdafruitUDP Class

```
// A UDP instance to let us send and receive packets over UDP
AdafruitUDP Udp;
```

The UDP and TCP classes should generally be compatible with each other, so simply changing the class type and using the same field name should solve 90% of your problems.

Constants

The WICED Feather library uses a handful of public constants, enums, typedefs and defines. In some situations, you will have to use these constants, enums, typedefs or defines in your own sketches, and the most common values are documented below:

wl_enc_type_t

This typedef (which resolves to an `int32_t` value) is used to indicate the security encoding mechanism used by your AP when establishing a connection. You can indicate the following values in the encoding type parameter of **Feather.connect**:

- **ENC_TYPE_AUTO**

Attempts to automatically detect the security encoding type. This is the default option if no encoding type is specified in **Feather.connect**, but is also the slowest since it has to scan for all APs in range and determine the security type if the requested AP is found.

- **ENC_TYPE_OPEN**

Open AP (no security or password required)

- **ENC_TYPE_WEP**

WEP security with open authentication

- **ENC_TYPE_WEP_SHARED**
WEP security with shared authentication
- **ENC_TYPE_WPA_TKIP**
WPA security with TKIP
- **ENC_TYPE_WPA_AES**
WPA security with AES
- **ENC_TYPE_WPA_MIXED**
WPA security with AES and TKIP
- **ENC_TYPE_WPA2_TKIP**
WPA2 security with TKIP
- **ENC_TYPE_WPA2_AES**
WPA2 security with AES
- **ENC_TYPE_WPA2_MIXED**
WPA2 security with TKIP and AES
- **ENC_TYPE_WPA_TKIP_ENT**
WPA enterprise security with TKIP
- **ENC_TYPE_WPA_AES_ENT**
WPA enterprise security with AES
- **ENC_TYPE_WPA_MIXED_ENT**
WPA enterprise security with TKIP and AES
- **ENC_TYPE_WPA2_TKIP_ENT**
WPA2 enterprise security with TKIP
- **ENC_TYPE_WPA2_AES_ENT**
WPA2 enterprise security with AES
- **ENC_TYPE_WPA2_MIXED_ENT**
WPA2 enterprise security with TKIP and AES
- **ENC_TYPE_WPS_OPEN**
WPS with open security
- **ENC_TYPE_WPS_SECURE**
WPS with AES security
- **ENC_TYPE_IBSS_OPEN**
BSS with open security

err_t

The most frequently encountered error codes are defined below:

- **ERROR_NONE (0)**
This means that no error occurred and that execution completed as expected
- **ERROR_OUT_OF_HEAP_SPACE (3)**
This error indicates that you have run out of heap memory in Feather Lib

- **ERROR_NOT_CONNECTED (20)**
You will get this error if you try to perform an operation that requires a connection to an AP or the Internet when you aren't connected.
- **ERROR_WWD_INVALID_KEY (1004)**
You will get this error if the password you provided for your AP is invalid
- **ERROR_WWD_AUTHENTICATION_FAILED (1006)**
You will get this error if authentication failed trying to connect to the AP
- **ERROR_WWD_NETWORK_NOT_FOUND (1024)**
You will get this error if the requested AP could not be found in an AP scan. A likely cause of this error message is that you are out of range of the AP.
- **ERROR_WWD_UNABLE_TO_JOIN (1025)**
You will get this error if you are unable to join the requested AP. A likely cause of this error message is that you are out of range of the AP.
- **ERROR_WWD_ACCESS_POINT_NOT_FOUND (1066)**
This error message indicates that the requested AP could not be found
- **ERROR_TLS_UNTRUSTED_CERTIFICATE (5035)**
Indicates that the certificate from the remote secure server could not be validated against any of the root certificates available to WICED. You may need to add another root certificate via `Feather.addRootCA(...)`.
- **ERROR_SDEP_INVALIDPARAMETER (30002)**
This error indicates that an invalid parameter was provided to the underlying SDEP command, or a parameter was rejected by the command handler.

There are hundreds of other possible error codes, and they can't all be documented here, but using the `.errno()` and `.errstr()` functions in `AdafruitFeather` you can get either the 16-bit error code or a string that provides a basic description for that error code.

The following code shows how you might use a combination of `.errno()` and `.errstr()` to handle common error codes:

```
// Attempt to connect to the AP
if ( Feather.connect("SSID", "PASSWORD", ENC_TYPE_AUTO ) )
{
    int8_t rssi = Feather.RSSI();
    uint32_t ipAddress = Feather.localIP();
    // Do something now that you are connected to the AP!
}
else
{
    // Display the error message
    err_t err = Feather.errno();
    Serial.println("Connection Error:");
    switch (err)
    {
        case ERROR_WWD_ACCESS_POINT_NOT_FOUND:
            // SSID wasn't found when scanning for APs
            Serial.println("Invalid SSID");
            break;
```

```

case ERROR_WWD_INVALID_KEY:
    // Invalid SSID passkey
    Serial.println("Invalid Password");
    break;
default:
    // The most likely cause of errors at this point is that
    // you are just out of the device/AP operating range
    Serial.print(Feather.errno());
    Serial.print(":");
    Serial.println(Feather.errstr());
    break;
}
}

```

wl_ap_info_t

Access points are described with the following typedef/struct, which you may need to access on certain specific occasions:

```

typedef struct ATTR_PACKED
{
    char        ssid[WIFI_MAX_SSID_LEN+1];
    uint8_t     bssid[6];
    int16_t     rssi;
    uint32_t    max_data_rate;
    uint8_t     network_type;
    int32_t     security;
    uint8_t     channel;
    uint8_t     band_2_4ghz;
} wl_ap_info_t;

```

WIFI_MAX_SSID_LEN is equal to 32 and is set in adafruit_feather.h

Each AP described using this typedef will require 52 bytes of memory

Python Tools

A set of python based tools are included as part of the WICED Feather SDK. You generally only need to use these tools in very specific circumstances, but they are listed below and then discussed in further detail elsewhere in this learning guide.

On Windows, the BSP package that contains the tools folder is normally found in the

```
%LOCALAPPDATA%\Arduino15\packages\adafruit\hardware\wiced\version
```

folder. On OS X it can usually be found in the

```
~/Library/Arduino15/packages/adafruit/hardware/wiced/version
```

folder.

pyresource.py (Convert static files to C headers)

pyresource.py can be used to recursively convert text and binary files into C headers that can be used by modules like AdafruitHTTPServer. These files can then be embedded as part of your user sketch, and served as resources like images, HTML or JavaScript content, etc.

For more information see the dedicated [pyresource.py page \(https://adafru.it/qoD\)](https://adafru.it/qoD) in this guide.

pycert.py (Python TLS Certificate Converter)

pycert.py is a python tool that will retrieve the root certificate chain for a specific domain, converting it into a byte array and placing it in a standard C header file.

This header file can then be referenced in your code, and added to the default WICED root certificate list (via **Feather.addRootCA**) that validates security data sent from secure domains and websites.

For more information see the dedicated [pycert.py page \(https://adafru.it/peF\)](https://adafru.it/peF) in this guide.

feather_dfu.py (Python USB DFU Utility)

This python tool is used by the Arduino IDE to perform common operations like resetting into DFU mode, updating the flash contents of the MCU, performing a factory reset, or getting some basic information about the modules.

While the tool is intended to be used by the Arduino IDE, you are also free to use it from the command line.

For more information see the dedicated [feather_dfu.py page \(https://adafru.it/qtA\)](https://adafru.it/qtA) in this guide.

pyresource.py

This tool will recursively scan the contents of a folder, and convert any files found into 'HTTPResource' entries that can be used with modules like the [AdafruitHTTPServer](https://adafru.it/qoE) (<https://adafru.it/qoE>).

Location: /tools/pyresource/pyresource.py

On Windows, the BSP package that contains the tools folder is normally found in the '%APPDATA%\Arduino15\packages\adafruit\hardware\wiced\0.6.0' folder. On OS X it can usually be found in the '~/Library/Arduino15/packages/adafruit/hardware/wiced/0.6.0' folder.

This tool was added in version 0.6.0 of FeatherLib

Usage

This tool accepts a single argument: the path to the folder where the files you wish to convert (recursively) are stored, relative to the current directory.

All HTTPResource header files will be written to the folder that the script is executed from.

Note that when using this tool folder separators ('/' or '\') will be converted to '_dir_' and spaces and periods will be converted to '_'.

```
Usage: pyresource.py [OPTIONS] DIR
```

```
Adafruit Python HTTP Resource Tool
```

```
This tool recursively converts the folder contents into HTTP server
resources in a C header format. These headers can then be imported into
WICED Feather HTTP server sketches.
```

```
Example of recursively converting the contents of the 'resources' folder:
```

```
$ python pyresource.py resources
```

```
Options:
--help  Show this message and exit.
```

As an example, if you place all of your static files in the '**resources**' folder of your Arduino sketch, and you wish to generate a set of **HTTPResource** records in the main sketch folder (one level higher than resources) you would run the tool as follows:

```
# Run from 'libraries/AdafruitWicedExamples/HTTPServer/D3Graphic'
python ../../../../tools/pyresource/pyresource.py resources
```

Assuming the same D3Graphic example mentioned above, this would generate the following output:

```
Looking for files in 'resources'
Converted 'resources/d3.min.js' to '_d3_min_js.h'
Converted 'resources/favicon.ico' to 'favicon_ico.h'
Converted 'resources/index.html' to 'index_html.h'
Wrote resource index to 'resources.h'
```

HTTPResource Records

Looking at the example above, we can see that three static files were converted to headers and **HTTPResource** records ('d3.min.js', 'favicon.ico', and 'index.html').

Each output header file contains a single **HTTPResource**, which has the binary equivalent of the file encoded inside it.

For example, for favicon_ico.h we get a 10990 byte long **HTTPResource** named **favicon_ico**, shown below:

```
/* Auto-generated by pyresource. Do not edit this file. */
const uint8_t favicon_ico_data[10990] = {
    0x00, 0x00, 0x01, 0x00, 0x03, 0x00, 0x10, 0x10, 0x00, 0x00, 0x01, 0x00, 0x08,
    0x00, 0x68, 0x05, 0x00, 0x00, 0x36, 0x00, 0x00, 0x00, 0x20, 0x20, 0x00, 0x00,
    // ... data removed for brevity ...
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00
};

const HTTPResource favicon_ico(favicon_ico_data, 10990);
```

HTTPResource Collection: resources.h

The tool will also generate a single header file named **resources.h**, which is the only file that you need to reference in your sketch.

The resources.h file lists all of the **HTTPResource** records available, and you can then insert these resources into a page collection for your sketch, [adding them to the AdafruitHTTPServer](#) (<https://adafru.it/qoE>) individually or as a list.

The tool will attempt to automatically determine the **MIME type** for the file based on the file extension, selecting from the list of MIME types supported by FeatherLib.

Using the example from above, we would get the following content in resources.h from D3Graphic:

```
#ifndef _RESOURCE_H_
#define _RESOURCE_H_

/* Auto-generated by pyresource. Do not edit this file. */

#include "http_common.h"
#include "_d3_min_js.h"
#include "favicon_ico.h"
#include "index_html.h"

/* HTTPPage collection from generated headers

HTTPPage("/d3.min.js", HTTP_MIME_JAVASCRIPT, &_d3_min_js),
HTTPPage("/favicon.ico", HTTP_MIME_IMAGE_MICROSOFT, &favicon_ico),
HTTPPage("/index.html", HTTP_MIME_TEXT_HTML, &index_html),

*/

#endif /* ifndef _RESOURCE_H_ */
```

For details on using the static content references in resources.h, see the appropriate section in the [AdafruitHTTPServer \(https://adafru.it/qoE\)](https://adafru.it/qoE) classes documentation.

pycert.py

pycert.py is a python tool that will retrieve the root certificate chain for a specific domain, converting it into a byte array and placing it in a standard C header file.

This header file can then be referenced in your code, and added to the default WICED root certificate list (via **Feather.addRootCA**) that validates security data sent from secure domains and websites.

Location: /tools/pycert/pycert.py

On Windows, the BSP package that contains the tools folder is normally found in the '%APPDATA%\Arduino15\packages\adafruit\hardware\wiced\0.6.0' folder. On OS X it can usually be found in the '~/Library/Arduino15/packages/adafruit/hardware/wiced/0.6.0' folder..

If you are using this tool on Windows you will need to install pyopenssl via 'pip install pyopenssl' from the command line.

Downloading the Root Certificate for a Domain

The most common command used with pycert.py is `download`, which accepts one or more domain names as a parameter, downloads the certificate chain for that domain, and then converts the root certificate(s) into a single header file.

Parameters

The '`download`' command has the following parameters:

```
Usage: pycert.py download [OPTIONS] [DOMAIN]...
```

<code>-p, --port INTEGER</code>	port to use for reading certificate (default 443, SSL)
<code>-c, --cert-var TEXT</code>	name of the variable in the header which will contain certificate data (default: <code>rootca_certs</code>)
<code>-l, --cert-length-var TEXT</code>	name of the define in the header which will contain the length of the certificate data (default: <code>ROOTCA_CERTS_LEN</code>)
<code>-o, --output FILENAME</code>	name of the output file (default: <code>certificates.h</code>)
<code>-f, --full-chain</code>	use the full certificate chain and not just the root/last cert (default: <code>false</code> , root cert only)
<code>-d, --keep-dupes</code>	write all certs including any duplicates across domains (default: <code>remove duplicates</code>)
<code>--help</code>	Show this message and exit.

Usage

To download and convert the root certificate for `adafruit.com`, for example, you would issue the following command:

```
python pycert.py download adafruit.com
```

If you want to change the output filename (in case you have multiple header files to deal with), and convert two domains at the same time into a single header file, you would issue the following command:

```
pycert download --output data.h google.com adafruit.com
```

Converting PEM Files

You can also use the `convert` command to convert a text PEM/.pem file to a C header, which is provided as a convenience since many browsers will allow you to navigate to a specific domain and export the certificate chain in .pem format.

Parameters

The '`convert`' command has the following parameters:

```
Usage: pycert.py convert [OPTIONS] [CERT]...
```

<code>-c, --cert-var TEXT</code>	name of the variable in the header which will contain certificate data (default: <code>rootca_certs</code>)
<code>-l, --cert-length-var TEXT</code>	name of the define in the header which will contain the length of the certificate data (default: <code>ROOTCA_CERTS_LEN</code>)
<code>-o, --output FILENAME</code>	name of the output file (default: <code>certificates.h</code>)
<code>-f, --full-chain</code>	use the full certificate chain and not just the root/last cert (default: <code>false</code> , root cert only)
<code>-d, --keep-dupes</code>	write all certs including any duplicates (default: <code>remove duplicates</code>)
<code>--help</code>	Show this message and exit.

Usage

To convert a single .pem file to a C header you could use the following command:

```
python pycert.py convert foo.pem
```

You can also convert multiple .pem files into one C header as follows:

```
python pycert.py convert foo.pem bar.pem
```

feather_dfu.py

WINDOWS USERS: Recent versions of the BSP include a pre-compiled version of feather_dfu for Windows. If you are using Windows as a platform, look in the 'tools/win32-x86/feather_dfu' folder for the executable file to use.

This python tool is used by the Arduino IDE to perform common operations like resetting into DFU mode, updating the flash contents of the MCU, performing a factory reset, or getting some basic information about the modules.

While the tool is intended to be used by the Arduino IDE, you are also free to use this tool from the command line.

Location: /tools/feather_dfu/feather_dfu.py

On Windows, the BSP package that contains the tools folder is normally found in the '%APPDATA%\Local\Arduino15\packages\adafruit\hardware\wiced\0.6.0' folder. On OS X it can usually be found in the '~/Library/Arduino15/packages/adafruit/hardware/wiced/0.6.0' folder..

'feather_dfu.py' depends on 'sdep.py' in the same directory, which handles sending SDEP commands over USB. If you wish to talk to the WICED Feather over USB using SDEP commands, this may be a useful reference to look at.

Commands

feather_dfu.py exposes the following commands:

arduino_upgrade

This command will flash your user code (the code compiled in the Arduino IDE) to the appropriate section in flash memory.

You must provide a .bin file as an argument with this command, for example:

```
python feather_dfu.py arduino_upgrade mycode.bin
```

featherlib_upgrade

This command will flash the FeatherLib section of flash memory.

You must provide an appropriate .bin file as an argument with this command, for example:

```
python feather_dfu.py featherlib_upgrade ../../stm32/featherlib/featherlib.bin
```

enter_dfu

Causes the WICED Feather to enter DFU mode. You will know if you are in the special DFU/Bootloader mode because the LED will blinky at a faster than normal rate.

```
python feather_dfu.py enter_dfu
```

info

Running this command will provide some basic information about your WICED Feather, and can be used when trying to debug issues in the support forums, etc.

When you run the '**info**' command you will see results resembling the following:

```
$ python feather_dfu.py info
Feather
ST32F205RGY
353231313533470E00430036
44:39:C4:EB:B9:64
0.1.0
3.5.2
0.5.0
0.5.0
Feb 26 2016
```

In order of appearance these values signify:

- The firmware family (normally 'Feather')
- The MCU version (normally 'STM32F205RG*')
- The unique serial number for this MCU
- The 48-bit HW MAC address for this chip
- The bootloader version
- The WICED SDK version
- The FeatherLib version
- The ArduinoCode version (may be user defined, or may mirror FeatherLib)
- The date the flashed FeatherLib was compiled

factory_reset

This command will perform a factory reset on the WICED Feather, erasing the Arduino user code as well as resetting the non-volatile config memory to factory defaults.

```
python feather_dfu.py factory_reset
```

nvm_reset

Resets to non-volatile config memory to factory default settings (but leaves the Arduino user code intact).

```
python feather_dfu.py nvm_reset
```

reboot

Causes the WICED Feather to perform a HW reset.

```
python feather_dfu.py reboot
```

SDEP Commands

SDEP commands allow the user code to communicate with the feather lib and vice versa. Normally you never need to use these commands directly (they are used by the higher level WICED Feather API), but they are documented below for advanced users and for debugging purposes.

```
// Generic Commands
SDEP_CMD_RESET                = 0x0001,    ///< HW reset
SDEP_CMD_FACTORYRESET         = 0x0002,    ///< Factory reset
SDEP_CMD_DFU                  = 0x0003,    ///< Enter DFU mode
SDEP_CMD_INFO                 = 0x0004,    ///< System information
SDEP_CMD_NVM_RESET            = 0x0005,    ///< Reset DCT
SDEP_CMD_ERROR_STRING         = 0x0006,    ///< Get descriptive error string
SDEP_CMD_COMMAND_STRING       = 0x0007,    ///< Get descriptive SDEP command
string

// Hardware Commands
SDEP_CMD_GPIO                 = 0x0100,    ///< Set GPIO
SDEP_CMD_RANDOMNUMBER         = 0x0101,    ///< Random number

// SPI Flash Commands
SDEP_CMD_SFLASHFORMAT         = 0x0200,    ///< Format SPI flash memory
SDEP_CMD_SFLASHLIST           = 0x0201,    ///< List SPI flash contents

// DEBUG Commands
SDEP_CMD_STACKDUMP            = 0x0300,    ///< Dump the stack
SDEP_CMD_STACKSIZE            = 0x0301,    ///< Get stack size
SDEP_CMD_HEAPDUMP             = 0x0302,    ///< Dump the heap
SDEP_CMD_HEAPSIZE             = 0x0303,    ///< Get heap size
SDEP_CMD_THREADLIST           = 0x0304,    ///< Get thread information

// WiFi Commands
SDEP_CMD_SCAN                 = 0x0400,    ///< AP scan
SDEP_CMD_CONNECT              = 0x0401,    ///< Connect to AP
SDEP_CMD_DISCONNECT           = 0x0402,    ///< Disconnect from AP
SDEP_CMD_APSTART              = 0x0403,    ///< Start AP
SDEP_CMD_APSTOP               = 0x0404,    ///< Stop AP
SDEP_CMD_WIFI_GET_RSSI        = 0x0405,    ///< Get RSSI of current connected
signal
SDEP_CMD_WIFI_PROFILE_ADD     = 0x0406,    ///< Add a network profile
SDEP_CMD_WIFI_PROFILE_DEL     = 0x0407,    ///< Remove a network profile
SDEP_CMD_WIFI_PROFILE_CLEAR   = 0x0408,    ///< Clear all network profiles
SDEP_CMD_WIFI_PROFILE_CHECK   = 0x0409,    ///< Check if a network profile exists
SDEP_CMD_WIFI_PROFILE_SAVE    = 0x040A,    ///< Save current connected profile to
NVM
SDEP_CMD_WIFI_PROFILE_GET     = 0x040B,    ///< Get AP's profile info
SDEP_CMD_TLS_DEFAULT_ROOT_CA = 0x040C,    ///< Enable the default Root CA list
SDEP_CMD_TLS_ADD_ROOT_CA     = 0x040D,    ///< Add an custom ROOT CA to current
Chain
SDEP_CMD_TLS_CLEAR_ROOT_CA    = 0x040E,    ///< Clear the whole ROOT CA chain

// Gateway Commands
SDEP_CMD_GET_IPV4_ADDRESS     = 0x0500,    ///< Get IPv4 address from an interface
```

```

SDEP_CMD_GET_IPV6_ADDRESS    = 0x0501,    ///< Get IPv6 address from an interface
SDEP_CMD_GET_GATEWAY_ADDRESS = 0x0502,    ///< Get IPv6 gateway address
SDEP_CMD_GET_NETMASK        = 0x0503,    ///< Get IPv4 DNS netmask
SDEP_CMD_GET_MAC_ADDRESS    = 0x0504,    ///< Get MAC Address

// Network Commands
SDEP_CMD_PING                = 0x0600,    ///< Ping
SDEP_CMD_DNSLOOKUP          = 0x0601,    ///< DNS lookup
SDEP_CMD_GET_ISO8601_TIME    = 0x0602,    ///< Get time
SDEP_CMD_GET_UTC_TIME        = 0x0603,    ///< Get UTC time in seconds

// TCP Commands
SDEP_CMD_TCP_CONNECT        = 0x0700,    ///< Create TCP stream socket and
connect
SDEP_CMD_TCP_WRITE          = 0x0701,    ///< Write to the TCP stream socket
SDEP_CMD_TCP_FLUSH          = 0x0702,    ///< Flush TCP stream socket
SDEP_CMD_TCP_READ           = 0x0703,    ///< Read from the TCP stream socket
SDEP_CMD_TCP_DISCONNECT     = 0x0704,    ///< Disconnect TCP stream socket
SDEP_CMD_TCP_AVAILABLE      = 0x0705,    ///< Check if there is data in TCP
stream socket
SDEP_CMD_TCP_PEEK           = 0x0706,    ///< Peek at byte data from TCP stream
socket
SDEP_CMD_TCP_STATUS         = 0x0707,    ///< Get status of TCP stream socket
SDEP_CMD_TCP_SET_CALLBACK   = 0x0708,    ///< Set callback function for TCP
connection
SDEP_CMD_TCP_LISTEN         = 0x0709,
SDEP_CMD_TCP_ACCEPT         = 0x070A,
SDEP_CMD_TCP_PEER_INFO      = 0x070B,

// UDP Commands
SDEP_CMD_UDP_CREATE         = 0x0800,    ///< Create UDP socket
SDEP_CMD_UDP_WRITE          = 0x0801,    ///< Write to the UDP socket
SDEP_CMD_UDP_FLUSH          = 0x0802,    ///< Flush UDP stream socket
SDEP_CMD_UDP_READ           = 0x0803,    ///< Read from the UDP stream socket
SDEP_CMD_UDP_CLOSE          = 0x0804,    ///< Close UDP stream socket
SDEP_CMD_UDP_AVAILABLE      = 0x0805,    ///< Check if there is data in UDP
stream socket
SDEP_CMD_UDP_PEEK           = 0x0806,    ///< Peek at byte data from UDP stream
socket
SDEP_CMD_UDP_PACKET_INFO    = 0x0807,    ///< Get packet info of UDP stream
socket

// MQTT Commands
SDEP_CMD_MQTTCONNECT        = 0x0900,    ///< Connect to a broker
SDEP_CMD_MQTTDISCONNECT     = 0x0901,    ///< Disconnect from a broker
SDEP_CMD_MQTTPUBLISH        = 0x0902,    ///< Publish a message to a topic
SDEP_CMD_MQTTSUBSCRIBE      = 0x0903,    ///< Subscribe to a topic
SDEP_CMD_MQTTUNSUBSCRIBE    = 0x0904,    ///< Unsubscribe from a topic

```

Generic

Reset (0x0001)

Causes a full system reset. An SDEP response message is sent before the system reset is performed.

- **Command Enum:** SDEP_CMD_RESET
- **Command ID:** 0x0001
- **Added:** Codebase 0.5.0

Parameters: None.

Return Code(s):

- **ERROR_NONE** if the command executed properly.

Factory Reset (0x0002)

Performs a factory reset of the device, resetting all config data in non-volatile memory to factory defaults, as well as erasing the Arduino user code area (leaving the bootloader and feather library intact). A system reset will take place once the config data has been set to the default values.

- **Command Enum:** SDEP_CMD_FACTORYRESET
- **Command ID:** 0x0002
- **Added:** Codebase 0.5.0

Parameters: None.

Return Code(s):

- **ERROR_NONE** if the command executed properly.

Enter DFU Mode (0x0003)

Causes the board to reset into USB DFU mode.

- **Command Enum:** SDEP_CMD_DFU
- **Command ID:** 0x0003
- **Added:** Codebase 0.5.0

Parameters: None.

Return Code(s):

- **ERROR_NONE** if the command executed properly.

System Information (0x0004)

Returns a string or set of comma-separated strings containing basic system information, such as the firmware version, the HW MAC address, compilation date, etc.

- **Command Enum:** SDEP_CMD_INFO
- **Command ID:** 0x0004
- **Added:** Codebase 0.5.0

Parameters:

Parameter ID

This optional parameter allows you to indicate the specific system information value to be returned.

- **Mandatory:** No
- **Size:** 1 byte
- **Type:** uint8_t

The parameter ID can be one of the following values:

- **1: Board Name:** The board family the firmware was built against
- **2: MCU Name:** The target MCU the firmware was built against
- **3: Serial:** The serial string that uniquely identifies this MCU
- **4: MAC Address:** The HW MAC address for the radio interface
- **5: Bootloader Version:** The bootloader version used
- **6: SDK Version:** The SDK version for the Broadcom WICED WiFi stack
- **7: Codebase Version:** The version for the Adafruit Featherlib
- **8: Firmware Version:** Currently the same as codebase version
- **9: Build Date:** The date when the Featherlib was compiled

Response Message

If no Parameter ID value is provided, the complete list of values will be returned as a comma-separated list of strings in incrementing order, starting with 1, 'Board Name'.

If a valid Parameter ID is provided, only the corresponding value will be returned.

Return Code(s)

- **ERROR_NONE** if the command executed properly.
- **ERROR_SDEP_INVALIDPARAMETER** if an invalid Parameter ID was provided, or an invalid number of parameters is provided.

NVM Reset (0x0005)

Resets all config data stored in non-volatile memory to it's default state.

- **Command Enum:** SDEP_CMD_NVM_RESET
- **Command ID:** 0x0005
- **Added:** Codebase 0.5.0

Parameters: None.

Return Code(s):

- **ERROR_NONE** if the command executed properly.

Error String (0x0006)

Returns a string containing the internal name associated with the supplied 32-bit error code.

- **Command Enum:** SDEP_CMD_ERROR_STRING
- **Command ID:** 0x0006
- **Added:** Codebase 0.5.0

Parameters: None.

Error ID

Indicates the specific error code to be converted to it's internal string representation.

- **Mandatory:** Yes
- **Size:** 4 bytes
- **Type:** uint32_t

Response Message:

If a valid error code is provided, a string representing the enum associated with that value will be returned.

Return Code(s):

- **ERROR_NONE** if the command executed properly.
- **ERROR_SDEP_INVALIDPARAMETER** if an invalid number of parameters is provided.

Generate Random Number (0x0101)

Generates a random 32-bit value using the hardware random number generator on the STM32F2 MCU.

- **Command Enum:** SDEP_CMD_RANDOMNUMBER
- **Command ID:** 0x0101
- **Added:** Codebase 0.5.0

Parameters: None.

Return Code(s):

A 32-bit number generated via the hardware random number generator.

Return Code(s)

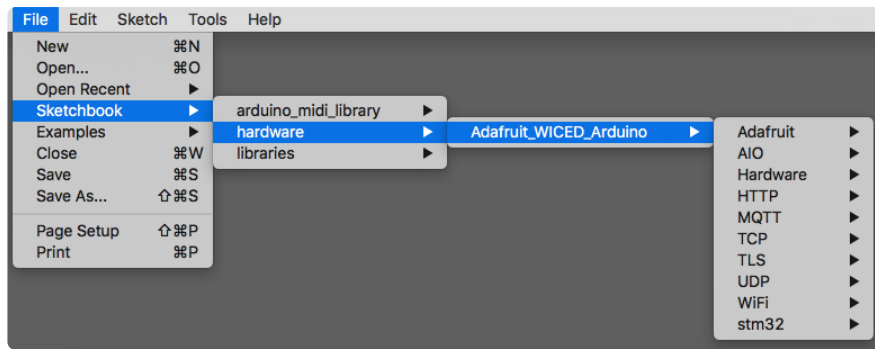
- **ERROR_NONE** if the command executed properly.
- **ERROR_SDEP_INVALIDPARAMETER** if an invalid number of parameters is provided.

Examples

The WICED Feather board support package includes a number of examples to help you get your project up and running with a minimum of effort.

Accessing the Examples (Arduino 1.6.5)

At present, the BSP installation is a manual process, as described in **Get the WICED BSP** earlier in this guide. To access to examples contained in this BSP, you will need to use a different menu path than you normally would:



Accessing the Examples (Arduino >= 1.6.8)

Recent versions of the Arduino IDE (after the 1.6.5 release used during development of this BSP) have changed the way examples sketches appear. On a newer version of the Arduino IDE (like 1.6.9), the WICED examples will no longer appear in the '**File > Sketchbook**' menu item.

To make the examples visible you must copy the contents of the ``hardware/Adafruit_WICED_Arduino/examples`` folder to your local sketchbook folder under a WICED subdirectory, so something like: ``sketchbook/WICED/examples``

Example Folders

The examples are broken up into sub-folders to try to keep things organized, although the exact folder structure is likely to evolve with time so it may not resemble exactly the image shown above.

As of the initial release, the following major folders are present:

- **Adafruit:** This folder contains test code that makes use of some specific Adafruit hardware
- **AIO:** Sketches making use of the [Adafruit IO \(https://adafru.it/fsU\)](https://adafru.it/fsU) servers
- **Hardware:** Examples showing how to use the peripherals on the STM32F205 MCU
- **HTTP:** Examples showing how to work with HTTP servers and data
- **MQTT:** Examples showing how to work with MQTT brokers
- **TCP:** Examples showing how to work with TCP sockets and connections
- **TLS:** Examples showing how to work with secure TLS/SSL/HTTPS TCP connections
- **UDP:** Examples related to UDP sockets and connections
- **WiFi:** General purpose wireless examples for the WICED Feather
- **stm32:** This folder contains libraries that are part of the WICED Feather BSP

Making Modifications to the Examples

One side effect of the examples being located outside of the normal examples structure is that **any changes you make to your sketch will be saved to the original example file.**

If you need to revert back to the original example, you may need to copy the code back from the original github repo. The examples code can always be seen here:

<https://adafru.it/B0u>

ScanNetworks

This example (found in the **Adafruit_WICED_Arduino/examples/WiFi** folder) will scan for access points in range of the WICED Feather.

Setup

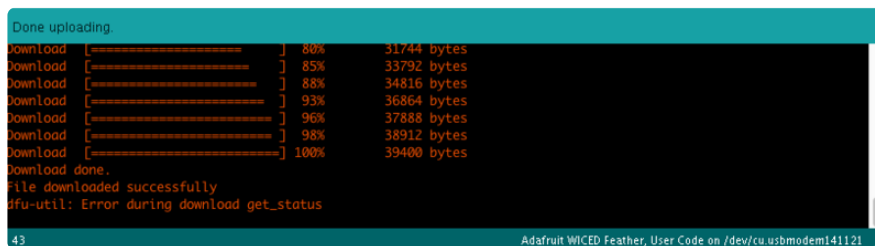
No particular setup is required for this sketch since it scans for available access points within range of the WICED Feather.

Compile and Flash

You can compile and flash your sketch to the WICED Feather using the 'Download' arrow icon at the top of the IDE:



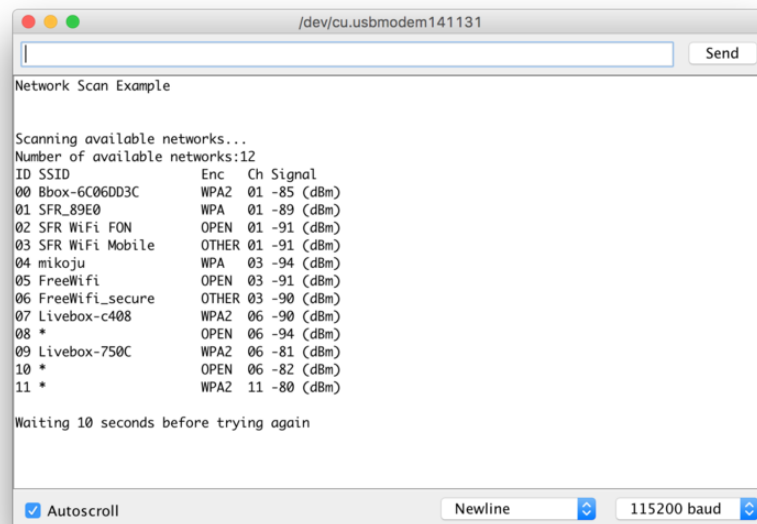
You should see the USB DFU progress as the update advances, and there will be a 'Done Uploading' message in the top left of the status bar when you are done:



Testing the Sketch

Wait a few seconds for the USB CDC serial interface to enumerate, and then open the **Serial Monitor** using either the Serial Monitor icon in the upper-right of the IDE or via **Tools > Serial Monitor**:

This will cause the WICED Feather to start scanning for access points in range:



Ping

This example (found in the **Adafruit_WICED_Arduino/examples/WiFi** folder) will ping the specified servers and display the ping response time(s).

Setup

Set your AP details using the **WLAN_SSID** and **WLAN_PASS** flags, setting them to the values used by your own access point:

```
#define WLAN_SSID          "YOURSSID"
#define WLAN_PASS          "YOURPASSWORD"
```

By default the sketch will ping **adafruit.com** and two Google domain name servers (8.8.8.8 and 8.8.4.4). If you wish to change the server(s) used, simply replace the values assigned in the variables below:

```
// Ping target by hostname
const char target_hostname[] = "adafruit.com";

// Ping target by IP String
const char target_ip_str[] = "8.8.8.8";
```

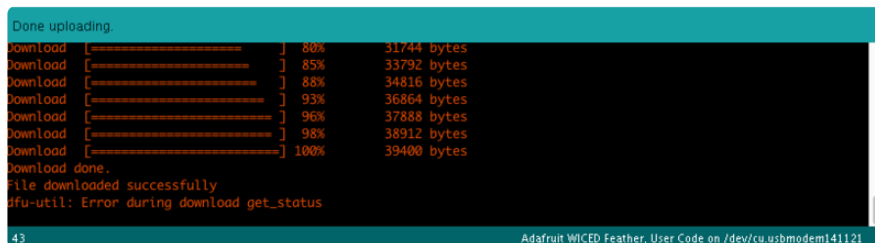
```
// Ping target by IPAddress object
IPAddress target_ip(8, 8, 4, 4);
```

Compile and Flash

You can then compile and flash your sketch to the WICED Feather using the 'Download' arrow icon at the top of the IDE:



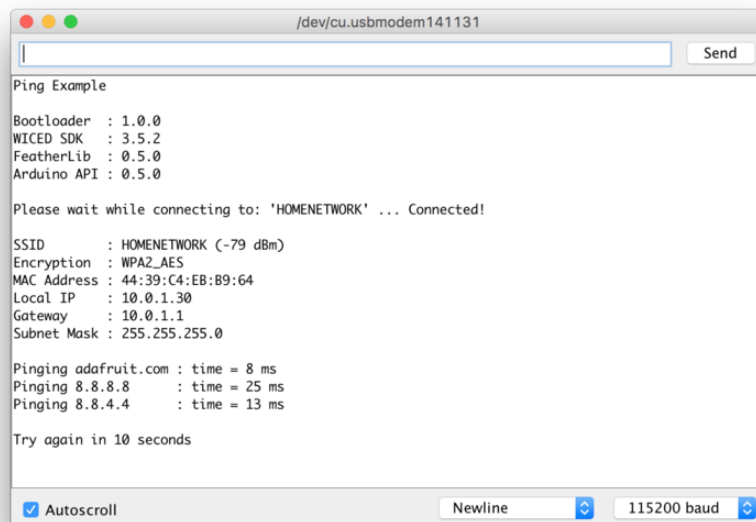
You should see the USB DFU progress as the update advances, and there will be a 'Done Uploading' message in the top left of the status bar when you are done:



Testing the Sketch

Wait a few seconds for the USB CDC serial interface to enumerate, and then open the **Serial Monitor** using either the Serial Monitor icon in the upper-right of the IDE or via **Tools > Serial Monitor**:

This will cause the WICED Feather to attempt to connect to the access point, and then it will attempt to ping the specified server(s):



GetHostByName

This example (located in **Adafruit_WICED_Arduino/examples/WiFi**) will perform a DNS lookup based on the specified domain name or IP address.

Setup

Set your AP details using the **WLAN_SSID** and **WLAN_PASS** flags, setting them to the values used by your own access point:

```
#define WLAN_SSID          "YOUR SSID HERE"
#define WLAN_PASS          "YOUR SSID KEY HERE"
```

Set the domain name or the IP address that you wish to resolve using the following variables:

```
// target by hostname
const char target_hostname[] = "adafruit.com";

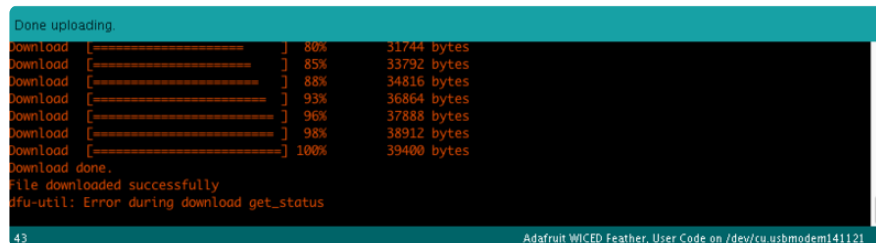
// target by IP String
const char target_ip_str[] = "8.8.8.8";
```

Compile and Flash

You can then compile and flash your sketch to the WICED Feather using the 'Download' arrow icon at the top of the IDE:



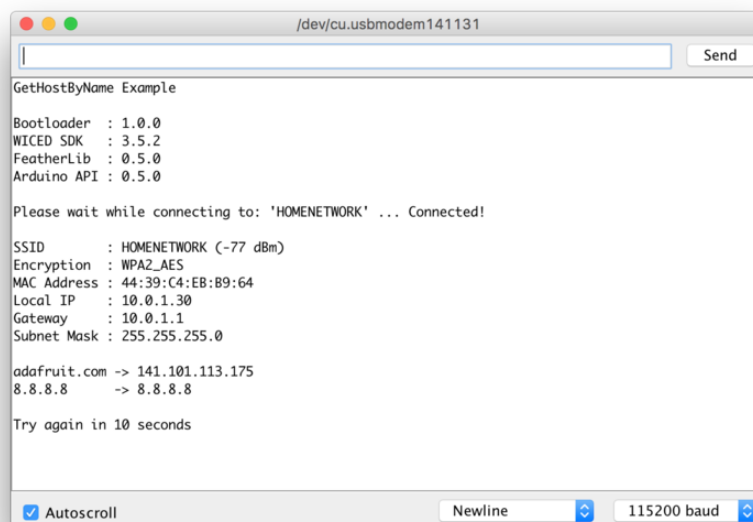
You should see the USB DFU progress as the update advances, and there will be a 'Done Uploading' message in the top left of the status bar when you are done:



Testing the Sketch

Wait a few seconds for the USB CDC serial interface to enumerate, and then open the **Serial Monitor** using either the Serial Monitor icon in the upper-right of the IDE or via **Tools > Serial Monitor**:

This will cause the WICED Feather to attempt to connect to the access point, and then it will attempt to look up the specified domain or IP address:



HttpGetPolling

This example (located in **Adafruit_WICED_Arduino/examples/HTTP**) will connect to an HTTP server and read the specified page using 'polling' (as opposed to using callbacks).

Setup

Set your AP details using the **WLAN_SSID** and **WLAN_PASS** flags, setting them to the values used by your own access point:

```
#define WLAN_SSID      "YOUR SSID HERE"
#define WLAN_PASS      "YOUR SSID KEY HERE"
```

Set the domain name or the IP address, the page and the port that you wish the resolve using the following variables:

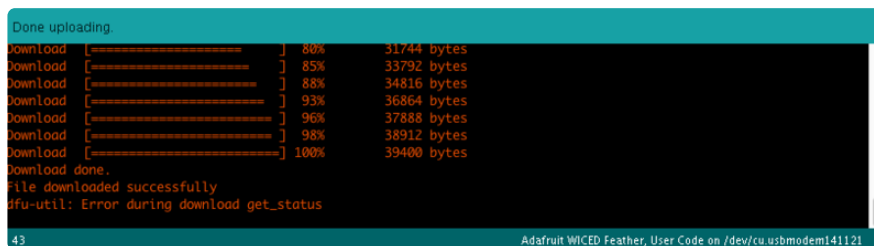
```
#define SERVER          "www.adafruit.com"    // The HTTP server to connect to
#define PAGE            "/testwifi/index.html" // The HTTP resource to request
#define PORT            80                   // The TCP port to use
```

Compile and Flash

You can then compile and flash your sketch to the WICED Feather using the 'Download' arrow icon at the top of the IDE:



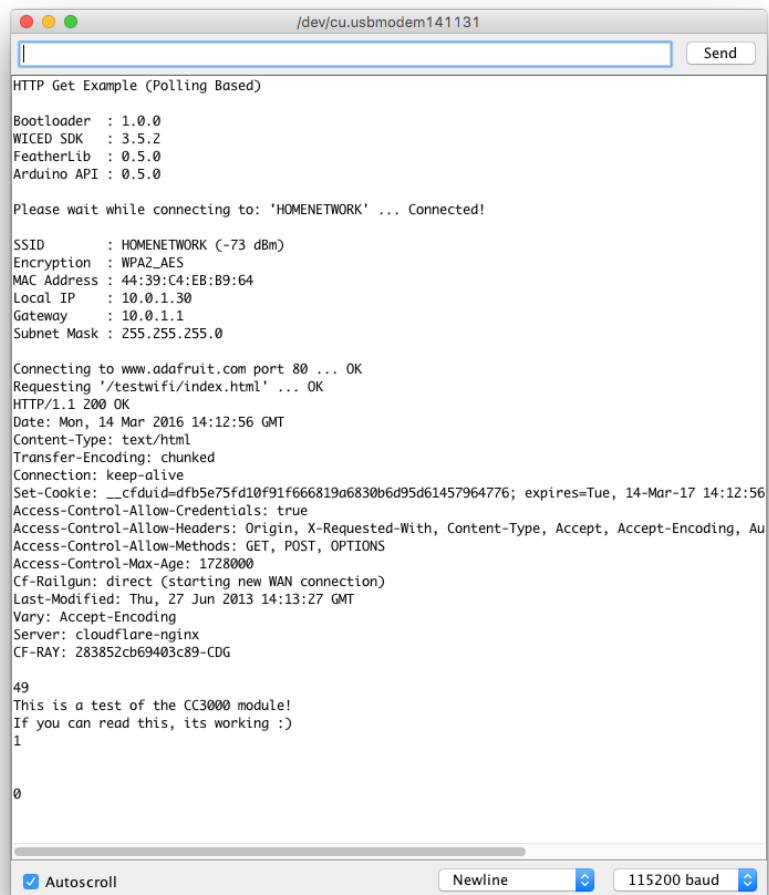
You should see the USB DFU progress as the update advances, and there will be a 'Done Uploading' message in the top left of the status bar when you are done:



Testing the Sketch

Wait a few seconds for the USB CDC serial interface to enumerate, and then open the **Serial Monitor** using either the Serial Monitor icon in the upper-right of the IDE or via **Tools > Serial Monitor**:

This will cause the WICED Feather to attempt to connect to the access point, and then it will attempt to retrieve the specified web page:



HttpGetCallback

This example (located in **Adafruit_WICED_Arduino/examples/HTTP**) will connect to an HTTP server and read the specified page using 'callbacks' (as opposed to using polling).

Setup

Set your AP details using the **WLAN_SSID** and **WLAN_PASS** flags, setting them to the values used by you own access point:

```
#define WLAN_SSID      "YOUR SSID HERE"
#define WLAN_PASS      "YOUR SSID KEY HERE"
```

Set the domain name or the IP address, the page and the port that you wish the resolve using the following variables:

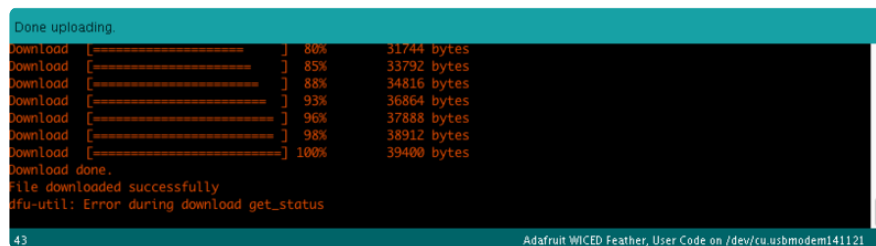
```
#define SERVER          "www.adafruit.com"    // The HTTP server to connect to
#define PAGE            "/testwifi/index.html" // The HTTP resource to request
#define PORT            80                    // The TCP port to use
```

Compile and Flash

You can then compile and flash your sketch to the WICED Feather using the 'Download' arrow icon at the top of the IDE:



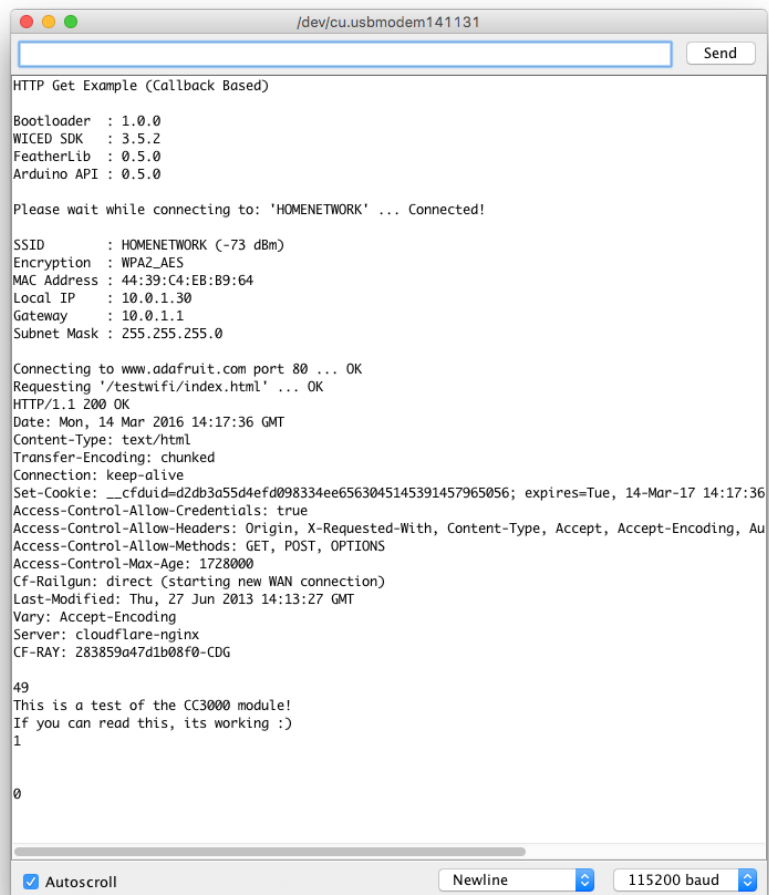
You should see the USB DFU progress as the update advances, and there will be a 'Done Uploading' message in the top left of the status bar when you are done:



Testing the Sketch

Wait a few seconds for the USB CDC serial interface to enumerate, and then open the **Serial Monitor** using either the Serial Monitor icon in the upper-right of the IDE or via **Tools > Serial Monitor**:

This will cause the WICED Feather to attempt to connect to the access point, and then it will attempt to retrieve the specified web page:



HTTPSLargeData

The example (located in the **Adafruit_WICED_Arduino/examples/TLS** folder) uses the AdafruitHTTP helper class and TLS to connect to a secure server and request a large file, which is then read using callbacks.

It tries to calculate the throughput for the specified file, which can be 10KB, 100KB or 1MB (indicate the file you wish to use before compiling the sketch).

Setup

Set your AP details using the **WLAN_SSID** and **WLAN_PASS** flags, setting them to the values used by you own access point:

```
#define WLAN_SSID      "YOUR SSID HERE"
#define WLAN_PASS      "YOUR SSID KEY HERE"
```

Next change the **FILE_ID** flag to indicate which file you want to load. Valid options are '0', '1', or '2':

```
#define FILE_ID    1

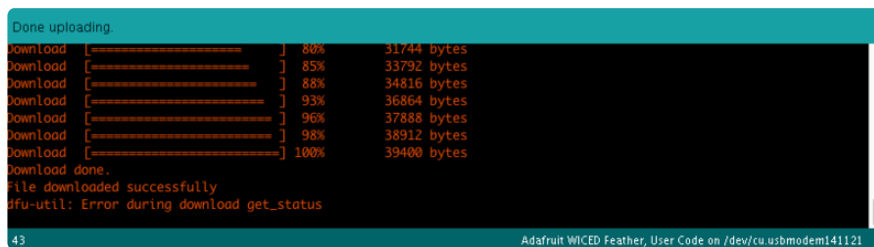
// S3 server to test large files,
const char * file_arr[] =
{
    [0] = "/text_10KB.txt" ,
    [1] = "/text_100KB.txt" ,
    [2] = "/text_1MB.txt" ,
};
```

Compile and Flash

You can then compile and flash your sketch to the WICED Feather using the 'Download' arrow icon at the top of the IDE:



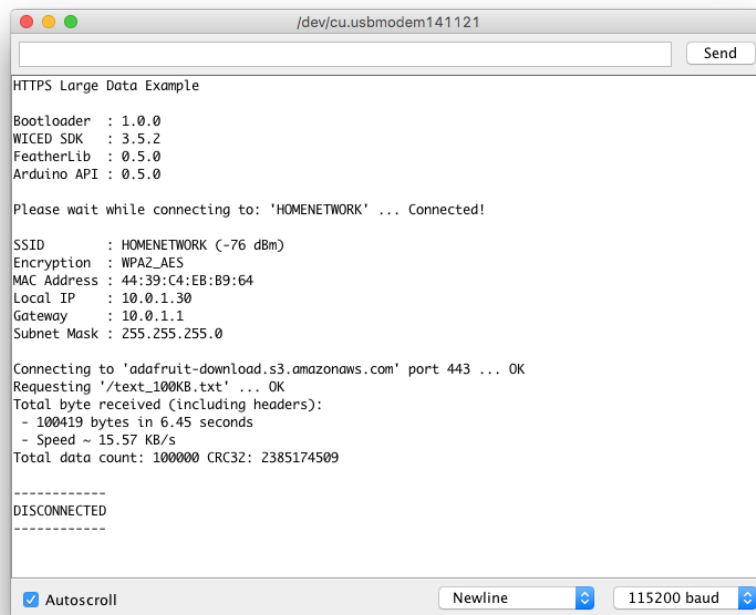
You should see the USB DFU progress as the update advances, and there will be a 'Done Uploading' message in the top left of the status bar when you are done:



Testing the Sketch

Wait a few seconds for the USB CDC serial interface to enumerate, and then open the **Serial Monitor** using either the Serial Monitor icon in the upper-right of the IDE or via **Tools > Serial Monitor**:

This will cause the WICED Feather to attempt to connect to the access point, and then make a secure (TLS based) connection and request to the Amazon S3 server for the specified file:



Throughput

The throughput example (located in the **Adafruit_WICED_Arduino/examples/WiFi** folder) uses AdafruitTCP to test the TCP throughput between the WICED Feather and another device running 'netcat', which simply listens for incoming TCP data on the specified port.

Setup

Set your AP details using the **WLAN_SSID** and **WLAN_PASS** flags, setting them to the values used by your own access point:

```
#define WLAN_SSID          "YOUR SSID HERE"
#define WLAN_PASS          "YOUR SSID KEY HERE"
```

You also need to set the IP address and port of the server you will be connecting to (the machine where you will be running netcat):

```
// your local PC's IP to test the throughput
// Run this command to create a server on your PC
// &gt; nc -l 8888

IPAddress server_ip(10, 0, 1, 27);
const uint16_t port = 8888;
```

Running Netcat

Before using this sketch you will need to start netcat and tell it to start listening on the pre-determined port, which can be done with the following command:

```
nc -l 8888
```

Depending on the version of netcat you are using, you may or may not see any feedback right away, but once netcat starts any incoming characters received will be echoed back to the command line, as shown in the example below:

[illegible]

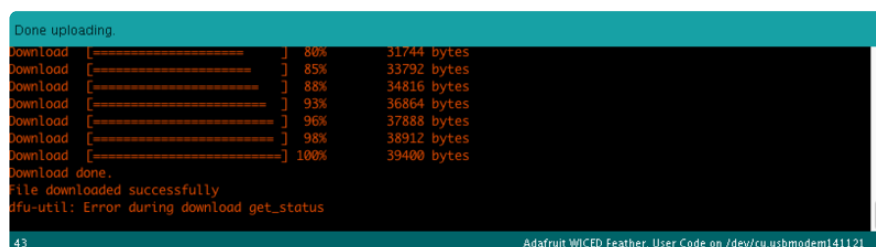
To stop netcat (once the test is complete) simply hit **CTRL+C**.

Compile and Flash

You can then compile and flash your sketch to the WICED Feather using the 'Download' arrow icon at the top of the IDE:



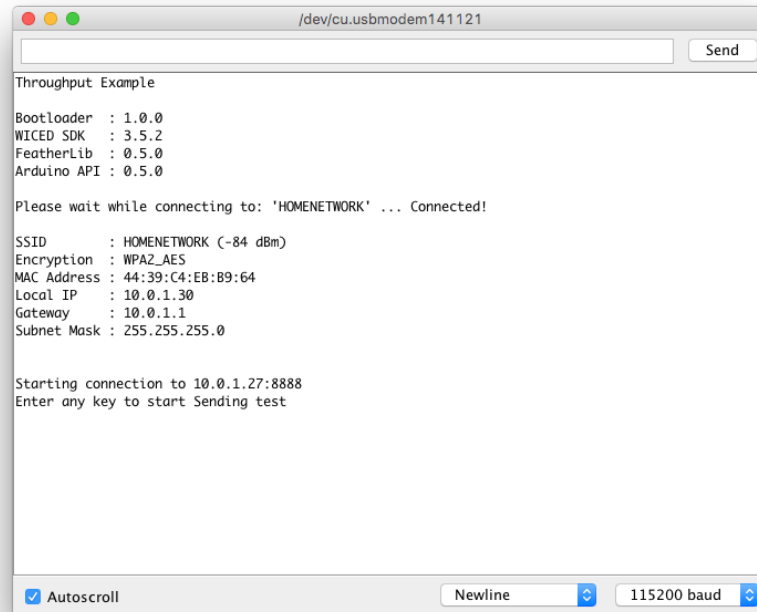
You should see the USB DFU progress as the update advances, and there will be a 'Done Uploading' message in the top left of the status bar when you are done:



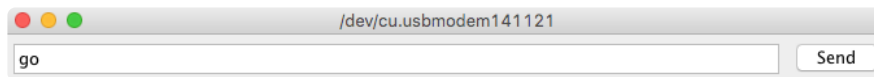
Testing the Sketch

Wait a few seconds for the USB CDC serial interface to enumerate, and then open the **Serial Monitor** using either the Serial Monitor icon in the upper-right of the IDE or via **Tools > Serial Monitor**:

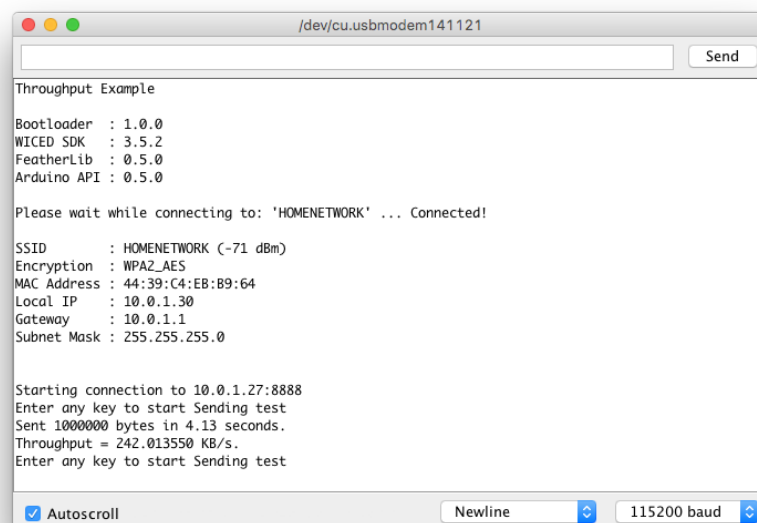
This will cause the WICED Feather to attempt to connect to the access point, and then it will attempt to connect to the netcat TCP Server:



At this point go to the top of the serial monitor and enter any character into the text box at the top and click the **SEND** button to start sending 1MB of data to netcat:



This will start the throughput test, which will display the calculated KB per second from the transfer:



FeatherOLED

The FeatherOLED example (located in the **Adafruit_WICED_Arduino/examples/Adafruit** folder) uses the [Adafruit_FeatherOLED](https://adafru.it/m3b) (<https://adafru.it/m3b>) library to display basic information about the WICED Feather on the [128x32 I2C OLED Feather Wing](http://adafru.it/2900) (<http://adafru.it/2900>).

This advanced example demonstrates several useful concepts and libraries for the WICED Feather:

- How to monitor the LIPO battery level
- How to work with an external OLED display for easy user feedback
- How to work with the [Adafruit Unified Sensor Library](https://adafru.it/dGB) (<https://adafru.it/dGB>) to retrieve sensor data
- How to work with [MQTT](https://adafru.it/m3c) (<https://adafru.it/m3c>) to push data to [Adafruit IO](https://adafru.it/m3d) (<https://adafru.it/m3d>)

This example optionally uses a [TSL2561 light sensor](http://adafru.it/439) (<http://adafru.it/439>) to generate real sensor data, but it should be relatively straight forward to use a different unified sensor driver, or you can disable the sensor entirely if you wish to simply use the OLED or send simulated sensor data.

Setup

Before you can use the FeatherOLED sketch you will have to install the [Adafruit_FeatherOLED](https://adafru.it/m3b) (<https://adafru.it/m3b>) library into your libraries folder. If you're new to Arduino our [Arduino Libraries Learning Guide](https://adafru.it/m3e) (<https://adafru.it/m3e>) explains everything you need to know to get Adafruit_FeatherOLED installed on your local system.

Setting the Access Point

Once you have Adafruit_FeatherOLED installed on your system, you need to set your AP details using the **WLAN_SSID** and **WLAN_PASS** flags in the example sketch, setting them to the values used by you own access point:

```
#define WLAN_SSID      "YOUR SSID HERE"  
#define WLAN_PASS      "YOUR SSID KEY HERE"
```

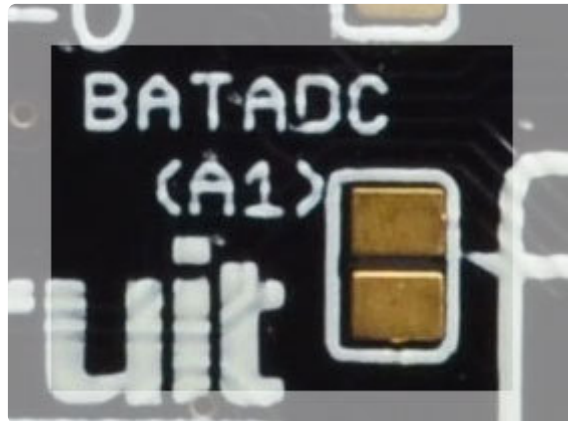
Enabling LIPO Battery Monitoring (Optional)

If you wish to monitor the LIPO cell voltage level, you will also need to enable the **VBAT_ENABLED** flag by setting its value to '1':

```
#define VBAT_ENABLED      1
#define VBAT_PIN          PA1
```

Important: Make sure that the **BATADC** solder jumper on the bottom of your WICED Feather is soldered shut as well, since this will run the LIPO cell through a voltage divider and into the ADC pin on PA1. See the Board Layout page for details, but the solder jumper can be seen below.

You have to solder these two metal leads together to form a 'bridge':



Enabling the TSL2561 Luminosity Sensor (Optional)

You can also enable the [TSL2561 light sensor](http://adafru.it/439) (<http://adafru.it/439>) to demonstrate how to work with the Adafruit_Sensor library to read sensor data on the WICED Feather.

To enable the TSL2561 in your sketch, simply set the `SENSOR_TSL2561_ENABLED` flag to '1':

```
#define SENSOR_TSL2561_ENABLED  1
```

This will cause the WICED Feather to read a new data sample from the TSL2561 every ten (10) seconds.

The TSL2561 should be connected to the WICED Feather as follows:

- TSL2561 **SCL** to WICED **SCL**
- TSL2561 **SDA** to WICED **SDA**
- TSL2561 **VIN** to WICED **3V**
- TSL2561 **GND** to WICED **GND**

Enabling MQTT to Adafruit IO (Optional)

You can optionally push the sensor data to Adafruit IO using the AdafruitAIO helper class.

To enable [MQTT](https://adafru.it/m3c) (<https://adafru.it/m3c>) to Adafruit IO support simply set the AIO_ENABLED flag to '1':

```
#define AIO_ENABLED      1
```

You also need to enter your AIO Username and your AIO key, as well as the target feeds that data should be published to:

```
#define AIO_USERNAME      "...your AIO username (see https://  
accounts.adafruit.com)..."  
#define AIO_KEY           "...your AIO key..."  
  
#define FEED_VBAT          "vbat"  
#define FEED_TSL2561_LUX  "lux"
```

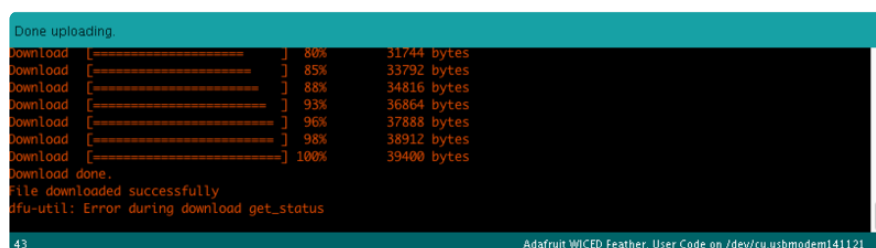
For more information on communication with Adafruit IO via MQTT see the [Adafruit IO MQTT API](https://adafru.it/m3d) (<https://adafru.it/m3d>).

Compile and Flash

You can compile and flash your sketch to the WICED Feather using the 'Download' arrow icon at the top of the IDE:



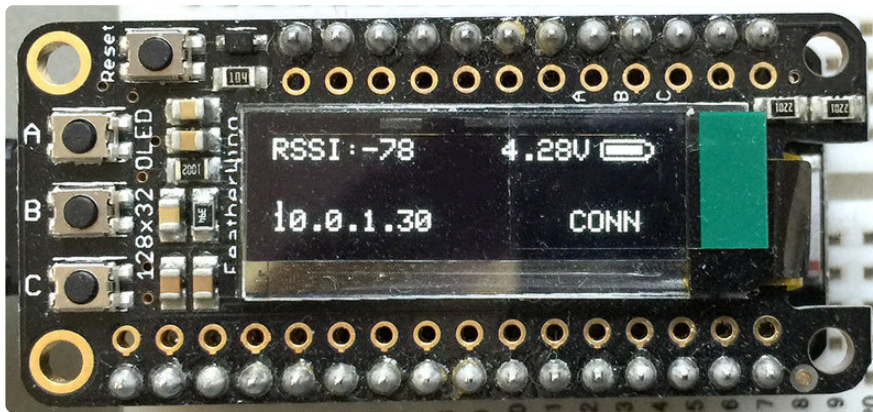
You should see the USB DFU progress as the update advances, and there will be a 'Done Uploading' message in the top left of the status bar when you are done:



Testing the Sketch

Unlike many of the example sketches, this example **will not wait for the USB CDC Serial Port to open before executing the code.**

If you have an OLED display properly connected, data should appear on it as soon as the USB DFU flash update process is completed:



The data rendered on the display will depend on the way that you configure the example sketch, but the top and bottom lines are reserved for WiFi and LIPO information, and the two middle lines (referred to as the 'Message Area' in Adafruit_FeatherOLED) can be used to render any text or messages.

FAQs

? I bricked my board. Can I force the device into DFU mode?

Yes. There are several ways to force the device into DFU mode if you somehow lock the board up with a faulty firmware image:

- Quickly double-click the **RESET** button on the board
- Set the **DFU Pin** to **GND** and reset the device (keeping DFU to GND during startup)
- Connect to the USB CDC interface at 1200 baud and disconnect. This magic baud rate signals to the module that we want to reset into DFU mode.
- Use the python script in 'tools/feather_dfu' to enter DFU mode:
`python feather_dfu.py enter_dfu`

Forcing the device into DFU mode should allow you to reflash the FeatherLib or user code and recover control of your hardware.

Note: You will know when you are in DFU mode since the on board status LED will start blinking at a rate of 5Hz.



What TLS Version does the WICED Feather support?

The WICED Feather supports the latest and greatest **TLS 1.2 standard**, which gives you access to the fastest and most secure encryption. It also supports TLS 1.1, TLS 1.0, and SSL 3.0. SSL 2.0 is **not** supported.

The WICED Feather supports the following cipher suites with TLS 1.2:

- TLS_DHE_RSA_WITH_AES_256_CBC_SHA256
- TLS_DHE_RSA_WITH_AES_256_CBC_SHA
- TLS_DHE_RSA_WITH_AES_128_CBC_SHA256
- TLS_DHE_RSA_WITH_AES_128_CBC_SHA
- TLS_RSA_WITH_AES_256_CBC_SHA256
- TLS_RSA_WITH_AES_256_CBC_SHA
- TLS_RSA_WITH_AES_128_CBC_SHA256
- TLS_RSA_WITH_AES_128_CBC_SHA
- TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA
- TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA
- TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256
- TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA

You can verify the TLS level yourself by pointing your WICED Feather to <https://www.howsmyssl.com> (<https://adafru.it/mgd>) or <https://www.ssllabs.com/ssltest/viewMyClient.html> (<https://adafru.it/mge>) and examining the HTML output. Note: You'll need to generate custom root certificates to access these domains, and you can read the output with the **TLS/HttpCustomRootCA** example.



When I try to build I'm getting:
Cannot run program
"{runtime.tools.arm-none-eabi-gcc.path}\bin\arm-none-eabi-g++" (in directory "."): CreateProcess error=2, The system cannot find the file specified

This is probably because you don't have the ARM Cortex M3 toolchain installed. Install the necessary GCC toolchain for ARM from the Arduino Board Manager via: **Tools->Board->Board Manager** then download **Arduino SAM Boards (32-bits ARM Cortex-M3)**

? When I try to flash using USB DFU I get the following error from featherdfu.py: Traceback (most recent call last): File "...\\hardware\\AdafruitWICEDArduino\\tools\\featherdfu.py", line 1, in import usb.backend.libusb1

This is probably caused by an old version of pyusb. Update your pyusb version to 1.0b or higher via the following command:

```
pip install --upgrade pyusb
```

You also need to make sure that you have the libusb runtime dll installed on your system, which you can do via this [libusb installer \(https://adafru.it/mba\)](https://adafru.it/mba). See the [Windows Setup \(https://adafru.it/mDg\)](https://adafru.it/mDg) page for details on using this installer though.

?

My board isn't enumerating as a USB device, or is stuck in DFU mode. How can I re-flash the FeatherLib firmware directly using dfu-util and restore my device?

You can reflash FeatherLib from the command line by forcing your device into DFU mode. See the first FAQ on this page for various ways to do this. Once in DFU mode (you'll know you're in DFU mode due to the constant blinky on the status LED), you can use dfu-util to flash a binary image to the WICED Feather using the following command syntax:

```
dfu-util -a 0 -s 0x08010000:leave -D featherlib.bin
```

0x08010000 is that start of the feather lib memory section (see the memory map in **System Architecture** in this learning guide for details).

To flash a user code binary you would change this value to 0x080E0000.

The 'featherlib.bin' image is available in the '[stm32/featherlib](https://adafru.it/19La) (<https://adafru.it/19La>)' folder. If you were running this from inside the `/tools/feather_dfu` folder you would execute this command as follows:

```
dfu-util -a 0 -s 0x08010000:leave -D ../../stm32/featherlib/featherlib.
```

If you have more than one DFU capable device on your system you can specify the exact USB VID and PID by adding the following flag:

```
-d 239a:0008
```

0x239A is the Vendor ID, and 0x0008 is the Product ID in DFU mode. You can verify the VID and PID values via ``dfu-util --list``.

This should result in output resembling the following;

```
dfu-util 0.8
```

```
Copyright 2005-2009 Weston Schmidt, Harald Welte and OpenMoko Inc.  
Copyright 2010-2014 Tormod Volden and Stefan Schmidt  
This program is Free Software and has ABSOLUTELY NO WARRANTY
```

Please report bugs to dfu-util@lists.gnumonks.org

dfu-util: Invalid DFU suffix signature

dfu-util: A valid DFU suffix will be required in a future dfu-util release

Opening DFU capable USB device...

ID 239a:0008

Run-time device DFU version 011a

Claiming USB DFU Interface...

Setting Alternate Setting #0 ...

Determining device status: state = dfuIDLE, status = 0

dfuIDLE, continuing

DFU mode device DFU version 011a

Device returned transfer size 1024

DfuSe interface name: "Internal Flash "

Downloading to address = 0x08010000, size = 464516

Download [=====] 100% 464516 bytes

Download done.

File downloaded successfully

Error during download get_status

At this point you have reflashed the FeatherLib section of code, and you should be able to flash your own code from the Arduino IDE in the 'User Code' section of flash memory.

NOTE: The WICED Feather also requires a valid user sketch (some Arduino code) to run, so after flashing the FeatherLib you will also need to compile and flash a sketch from the Arduino IDE for your board to start running. Powering up a board with only FeatherLib but no user sketch will end up in a 'dead-end' situation since it can't find any user code to execute.

Note++: Unlike many other Arduino compatible boards, you don't need a serial port to flash sketches from the Arduino IDE! The WICED Feather uses USB DFU, NOT the serial port for firmware updates! Don't worry if you don't see a serial port when you are trying to flash a sketch the first time.



How can I reflash the bootloader with a JLink or STLink/V2 from the Arduino IDE?

To reflash the bootloader on your WICED Feather using the Arduino IDE perform the following steps:

First install [AdaLink](https://adafru.it/fPq) (<https://adafru.it/fPq>) on your system, which is an abstraction layer that we provide to hide the details of different ARM hardware debuggers. If you have a choice, a Segger JLink is generally more reliable as a HW debugger and works across a larger variety of systems. The STLink with OpenOCD has issues with OS X El Capitan due to the new USB stack, for example.

To connect an [STLink/V2](http://adafru.it/2548) (<http://adafru.it/2548>) to the WICED Feather:

- Connect SWCLK on the STLink to SWCLK on the WICED Feather (which is a single 0.1" hole off the main header rail)
- Connect SWDIO on the STLink to SWDIO on the WICED Feather
- Connect GND on the STLink to GND on the WICED Feather
- Connect RST on the STLink to RST on the WICED Feather
- Power both the WICED Feather and STLink using USB

To connect a [Segger J-Link](http://adafru.it/1369) (<http://adafru.it/1369>) to the WICED Feather:

- Consult the [Segger JLink SWD and SWO Pinout](https://adafru.it/mEw) (<https://adafru.it/mEw>) for your JLink
- Connect SWCLK on the JLink to SWCLK on the WICED Feather (which is a single 0.1" hole off the main header rail)
- Connect SWDIO on the JLink to SWDIO on the WICED Feather
- Connect GND on the JLink to GND on the WICED Feather
- Connect VTRef on the JLink to 3V on the WICED Feather (**important!**)
- Connect RST on the JLink to RST on the WICED Feather
- Power both the WICED Feather and JLink using USB

From the Arduino IDE:

- Make sure 'Tools > Boards' is set to '**Adafruit WICED Feather**'
- In 'Tools > Programmer' select either '**STLinkV2 with AdaLink**' or '**JLink with AdaLink**'.

- Click the 'Tools > Burn Bootloader' menu entry, which should use AdaLink and either the STLink/V2 or JLink to flash the bootloader on your board.



How can I flash the bootloader using AdaLink directly?

You can also flash the bootloader from the command-line using [AdaLink \(https://adafru.it/fPq\)](https://adafru.it/fPq) directly.

- Make sure AdaLink is properly setup on your system (see the readme file in the Github repo).
- Find the bootloader.hex file in the [bootloader \(https://adafru.it/SGD\)](https://adafru.it/SGD) folder.
- Connect either a STLink/V2 or Segger JLink to your WICED Feather (see the FAQ entry above for connection details)
- With the debugger connected and both the debugger and WICED Feather powered, enter the following command (adjusting the path to bootloader.hex if required):

For an STLink/V2:

- `adalink stm32f2 -p stlink -h bootloader.hex`

For a Segger JLink:

- `adalink stm32f2 -p jlink -h bootloader.hex`

You can check if AdaLink is properly connected to the WICED Feather with the following commands:

For an STLink/V2:

- `adalink stm32f2 -p stlink -i`

For a Segger JLink:

- `adalink stm32f2 -p jlink -i`



I get 'OSError: [Errno 2] No such file or directory OSError: [Errno 2] No such file or directory' when trying to use feather_dfu.py in the Arduino IDE. What should I do?

If you get the following error in the Arduino IDE when trying to flash a sketch, you probably don't have **dfu-util** installed on your system:

OSError: [Errno 2] No such file or directory
OSError: [Errno 2] No such file or directory

Install dfu-util as detailed in this guide for your target OS.

Downloads

Related Documents

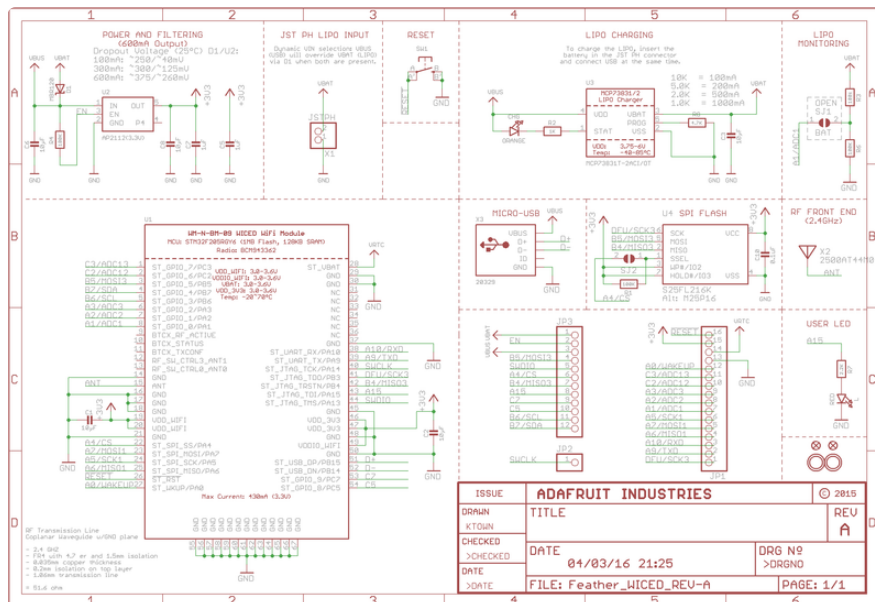
- [STM32F205RG Product Page \(https://adafru.it/m9A\)](https://adafru.it/m9A)
- [STM32F205 Datasheet \(https://adafru.it/m9B\)](https://adafru.it/m9B)
- [EagleCAD PCB files on GitHub \(https://adafru.it/oer\)](https://adafru.it/oer)
- [Fritzing object available in the Adafruit Fritzing Library \(https://adafru.it/aP3\)](https://adafru.it/aP3)

**WICED Feather WiFi Pinout
Diagram**

<https://adafru.it/z4f>

Schematic

The schematic for the latest WICED Feather board is shown below. Click the image for a higher resolution version.



Fabrication Print

Dimensions in Inches

