

---

# **ESP8266 Arduino Core Documentation**

*Versión 2.4.0*

**Ivan Grokhotkov**

**29 de enero de 2019**



---

## Contenido:

---

<b>1. Instalación</b>	<b>1</b>
1.1. Gestor de Tarjetas	1
1.2. Usando la versión git	2
<b>2. Referencia</b>	<b>5</b>
2.1. Digital IO	5
2.2. Entrada analógica	5
2.3. Salidas Analógicas	6
2.4. Tiempos y retrasos (delays)	6
2.5. Serial	7
2.6. Progmem	8
<b>3. Librerías</b>	<b>9</b>
3.1. WiFi(librería ESP8266WiFi)	9
3.2. Ticker	9
3.3. EEPROM	9
3.4. I2C (librería Wire)	10
3.5. SPI	10
3.6. SoftwareSerial	10
3.7. APIs específicas de ESP	10
3.8. Respondedor mDNS y DNS-SD (librería ESP8266mDNS)	11
3.9. Respondedor SSDP (ESP8266SSDP)	12
3.10. Servidor DNS (librería DNSServer)	12
3.11. Servo	12
3.12. Librería mejorada EEPROM para ESP (ESP_EEPROM)	12
3.13. Otras librerías (no incluidas con el IDE)	12
<b>4. Sistema de ficheros</b>	<b>15</b>
4.1. Esquema de la memoria flash	15
4.2. Limitaciones del sistema de ficheros	16
4.3. Subiendo ficheros al sistema de archivos	16
4.4. Objeto sistema de ficheros <i>SPIFFS</i>	17
4.5. Estructura de información del sistema de archivos	19
4.6. Objeto directorio <i>Dir</i>	19
4.7. Objeto fichero <i>File</i>	20
<b>5. Librería ESP8266WiFi</b>	<b>23</b>

5.1.	Introducción	23
5.2.	Descripción de la clase	25
5.3.	Diagnóstico	32
5.4.	¿Que hay dentro?	33
<b>6.</b>	<b>Actualizaciones OTA</b>	<b>39</b>
6.1.	Introducción	39
6.2.	Arduino IDE	40
6.3.	Buscador Web	50
6.4.	Servidor HTTP	53
6.5.	Interfaz de transmisión	56
6.6.	Clase Updater	56
<b>7.</b>	<b>Guía para PROGMEM sobre ESP8266 y Arduino IDE</b>	<b>59</b>
7.1.	Introducción	59
7.2.	Declarar una cadena flash en un bloque de código.	59
7.3.	Funciones para leer desde PROGMEM	60
7.4.	¿Como declaro una cadena global flash y la uso?	61
7.5.	Como uso cadenas alineadas flash?	62
7.6.	¿Como declaro y uso datos en PROGMEM?	62
7.7.	¿Como declaro algunos datos en PROGMEM y recupero un byte de él?	62
7.8.	En resumen	63
<b>8.</b>	<b>Utilizando GDB para depurar aplicaciones</b>	<b>65</b>
8.1.	Nota CLI e IDE	65
8.2.	Preparando tu aplicación para GDB	65
8.3.	Iniciando la sesión de depuración	66
8.4.	Ejemplo de sesión de depuración	67
8.5.	Limitaciones de depuración del Hardware ESP8266	70
<b>9.</b>	<b>Tarjetas</b>	<b>71</b>
9.1.	Módulo Genérico ESP8266	71
9.2.	Adaptador Serie	71
9.3.	Configuración Hardware mínima para Bootloading y ejecución	72
9.4.	ESP a Serie	72
9.5.	Mínimo	74
9.6.	Estabilidad mejorada	74
9.7.	Mensajes de arranque y modos	74
9.8.	Módulo Genérico ESP8285	76
9.9.	ESPDuino (Módulo ESP-13)	76
9.10.	Adafruit Feather HUZZAH ESP8266	76
9.11.	Invent One	77
9.12.	XinaBox CW01	77
9.13.	ESPRESSO Lite 1.0	77
9.14.	ESPRESSO Lite 2.0	77
9.15.	Phoenix 1.0	77
9.16.	Phoenix 2.0	77
9.17.	NodeMCU 0.9 (Módulo ESP-12)	78
9.18.	NodeMCU 1.0 (Módulo ESP-12E)	78
9.19.	Olimex MOD-WIFI-ESP8266(-DEV)	78
9.20.	SparkFun ESP8266 Thing	79
9.21.	SparkFun ESP8266 Thing Dev	79
9.22.	SweetPea ESP-210	79
9.23.	LOLIN(WEMOS) D1 R2 & mini	79
9.24.	LOLIN(WEMOS) D1 mini Pro	79

9.25. LOLIN(WEMOS) mini Lite . . . . .	79
9.26. WeMos D1 R1 . . . . .	80
9.27. ESPino (Módulo ESP-12) . . . . .	80
9.28. ThaiEasyElec's ESPino . . . . .	80
9.29. WifInfo . . . . .	80
9.30. Arduino . . . . .	80
9.31. 4D Systems gen4 IoD Range . . . . .	81
9.32. Digistump Oak . . . . .	81
9.33. WiFiduino . . . . .	81
9.34. Amperka WiFi Slot . . . . .	81
9.35. Seeed Wio Link . . . . .	81
9.36. ESPectro Core . . . . .	81
<b>10. Preguntas - FAQ</b>	<b>83</b>
10.1. Obtengo el error «espcmm_sync failed» cuando intento subir a mi ESP. ¿Como resuelvo este problema?	83
10.2. ¿Porqué no aparece esptool en el menú «Programador»? ¿Como subo al ESP sin él?	83
10.3. Mi ESP se bloquea al correr el programa. ¿Como lo resuelvo?	83
10.4. ¿Como puedo obtener algunos KBs extra en la flash?	84
10.5. Sobre WPS . . . . .	84
10.6. Esta librería de Arduino no funciona en ESP. ¿Como la hago funcionar?	84
10.7. En el IDE, para ESP-12E que tiene una flash de 4M, puedo seleccionar 4M (1M SPIFFS) o 4M (3M SPIFFS). No importa lo que seleccione, el IDE me dice que la capacidad máxima es de 1M. ¿Donde va mi flash?	84
10.8. He observado que ESP.restart() no funciona. Cual es la razón . . . . .	85
10.9. ¿Como solucionar el error «Board generic (platform esp8266, package esp8266) is unknown»?	85
10.10. ¿Cómo borrar PCBs TCP en estado de espera de tiempo?	85
10.11. ¿Por qué hay un generador de tarjetas y para que sirve?	86
<b>11. Exception Causes (EXCCAUSE)</b>	<b>87</b>
<b>12. Depuración</b>	<b>89</b>
12.1. Introducción . . . . .	89
12.2. Información . . . . .	91
<b>13. Volcados de pila</b>	<b>93</b>
13.1. Introducción . . . . .	93
<b>14. Utilizar Eclipse con Arduino ESP8266</b>	<b>97</b>
14.1. Que descargar . . . . .	97
14.2. Configurar Arduino . . . . .	97
14.3. Configurar Eclipse . . . . .	97
14.4. Eclipse no compila . . . . .	98



### 1.1 Gestor de Tarjetas

Este es el método elegido para los usuarios finales.

#### 1.1.1 Prerequisitos

- Arduino 1.6.8, descárgalo de la página [web de Arduino](#) :
- Conexión a internet

#### 1.1.2 Instrucciones

- Inicia Arduino y abre la ventana Preferencias.
- Introduce `http://arduino.esp8266.com/stable/package_esp8266com_index.json` en la casilla *Gestor de URLs Adicionales de Tarjetas*. Puedes añadir múltiples URLs separándolas con comas.
- Abre en gestor de tarjetas desde Herramientas > «Última tarjeta seleccionada» > Gestor de tarjetas y busca la plataforma *esp8266*.
- Seleccione la versión que desee de la lista.
- Click en el botón *Instalar*.
- No olvides seleccionar tu tarjeta ESP8266 desde Herramientas > Menú de tarjetas tras la instalación.

Opcionalmente puedes usar el paquete *staging* desde el siguiente link: `http://arduino.esp8266.com/staging/package_esp8266com_index.json`. Este contiene nuevas características, pero a la misma vez algo puede no funcionar.

Para mas información sobre el Gestor de Tarjetas, ver:

- <https://www.arduino.cc/es/guide/cores>

## 1.2 Usando la versión git

Esta es la instalación sugerida para contribuidores y desarrolladores de librerías.

### 1.2.1 Prerequisitos

- Arduino 1.6.8 (o mas moderno, versión actual funcionando 1.8.5)
- git
- **Python\_ 2.7** (<http://python.org>)
- terminal, console, o interprete de comandos (dependiendo de tu OS)
- Conexión a Internet

### 1.2.2 Instrucciones - Windows 10

- Primero, asegúrese de que no tiene la biblioteca ESP8266 instalada con el Gestor de Tarjetas (ver arriba)
- Instala git para Windows (si no lo tiene ya; ver <https://git-scm.com/download/win>)
- Abre un terminal y ve al directorio de Arduino por defecto. Este suele ser tu directorio *sketchbook* (normalmente C:\users\{username}\Documents\Arduino), la variable de entorno %USERPROFILE% normalmente contiene C:\users\{username}).
- Clona este repositorio al directorio hardware/esp8266com/esp8266.

```
cd %USERPROFILE%\Documents\Arduino\  
if not exist hardware mkdir hardware  
cd hardware  
if not exist esp8266com mkdir esp8266com  
cd esp8266com  
git clone https://github.com/esp8266/Arduino.git esp8266
```

Deberías acabar con la siguiente estructura de directorios en C:\Users\{your username}\Documents\:

```
Arduino  
|  
--- hardware  
    |  
    --- esp8266com  
        |  
        --- esp8266  
            |  
            --- bootloaders  
            --- cores  
            --- doc  
            --- libraries  
            --- package  
            --- tests  
            --- tools  
            --- variants  
            --- platform.txt  
            --- programmers.txt  
            --- README.md
```

(continues on next page)



(proviene de la página anterior)

```
--- boards.txt
--- LICENSE
```

- Inicialice los submodulos

```
cd %USERPROFILE%\Documents\hardware\esp8266com\esp8266
git submodule update --init
```

Si tiene mensajes de error sobre archivos faltantes relacionados con `SoftwareSerial` durante el proceso de compilación, debe ser porque omitió este paso y es necesario.

- Descarga las herramientas binarias:

```
cd esp8266/tools
python get.py
```

- Reinicia Arduino.
- Si utiliza el IDE de Arduino para Visual Studio (<https://www.visualmicro.com/>), asegúrese de hacer clic en Herramientas - Visual Micro - Volver a analizar las cadenas de herramientas y las bibliotecas
- Cuando actualice más tarde su biblioteca local, vaya al directorio `esp8266` y haga un `git pull`:

```
cd %USERPROFILE%\Documents\hardware\esp8266com\esp8266
git status
git pull
```

Tenga en cuenta que podría, en teoría, instalar en `C:\Archivos de programa (x86)\Arduino\hardware` sin embargo, esto tiene implicaciones de seguridad, por no mencionar que el directorio a menudo se pierde al reinstalar el IDE de Arduino. Tiene la ventaja (o el inconveniente, según su perspectiva) de estar disponible para todos los usuarios en su PC que utilizan Arduino.

### 1.2.3 Instrucciones - Otros Sistemas Operativos

- Abra un terminal y ve al directorio de Arduino. Este puede ser su directorio *sketchbook* (normalmente `<Documents>/Arduino`), o el directorio de instalación de Arduino mismamente, es tu elección.
- Clona este repositorio al directorio `hardware/esp8266com/esp8266`. Alternativamente, clonalo en otro lugar y crea un enlace simbólico, si tu sistema operativo lo admite.

```
cd hardware
mkdir esp8266com
cd esp8266com
git clone https://github.com/esp8266/Arduino.git esp8266
```

Deberías acabar con la siguiente estructura de directorios:

```
Arduino
|
--- hardware
    |
    --- esp8266com
        |
        --- esp8266
            |
            --- bootloaders
```

(continues on next page)

(proviene de la página anterior)

```
--- cores
--- doc
--- libraries
--- package
--- tests
--- tools
--- variants
--- platform.txt
--- programmers.txt
--- README.md
--- boards.txt
--- LICENSE
```

- Inicialice los submodulos

```
cd %USERPROFILE%\Documents\hardware\esp8266com\esp8266
git submodule update --init
```

Si tiene mensajes de error sobre archivos faltantes relacionados con `SoftwareSerial` durante el proceso de compilación, debe ser porque omitió este paso y es necesario.

- Descarga las herramientas binarias:

```
cd esp8266/tools
python get.py
```

- Reinicia Arduino.

- Cuando actualice más tarde su biblioteca local, vaya al directorio `esp8266` y haga un `git pull`:

```
cd hardware\esp8266com\esp8266
git status
git pull
```

## 2.1 Digital IO

Los números de los pines en Arduino se corresponden directamente con los números de los pines GPIO del ESP8266. Las funciones `pinMode`, `digitalRead` y `digitalWrite` funciona como de costumbre, así para leer el GPIO2, llama `digitalRead(2)`.

Los pines digitales 0—15 pueden ser `INPUT`, `OUTPUT` o `INPUT_PULLUP`. El pin 16 puede ser `INPUT`, `OUTPUT` o `INPUT_PULLDOWN_16`. Al arranque los pines están configurados como `INPUT`.

Los pines pueden también realizar otras funciones como: `Serial`, `I2C`, `SPI`. Estas funciones se activan normalmente mediante su correspondiente librería. El siguiente diagrama muestra el mapeo de pines para el popular módulo ESP-12.

Los pines digitales 6—11 no se muestran en el diagrama porque se utilizan para conectar la memoria flash en la mayoría de módulos. Al intentar usar estos pines como E/S hará que el programa falle.

Nota: Algunas tarjetas y módulos (ESP-12ED, NodeMCU 1.0) también poseen los pines 9 y 11. Estos pueden usarse si el chip flash trabaja en modo DIO (en vez de QIO, que es el modo por defecto).

La interrupción de pines se realiza a través de las funciones `attachInterrupt`, `detachInterrupt`. Las interrupciones pueden asociarse a cualquier GPIO, excepto GPIO16. Los tipos de interrupciones estándar de Arduino soportados son: `CHANGE`, `RISING`, `FALLING`.

## 2.2 Entrada analógica

El ESP8266 tiene un solo canal ADC disponible para los usuarios. Puede utilizarse para leer el voltaje en el pin ADC o para leer el voltaje de alimentación del módulo (VCC).

Para leer el voltaje externo suministrado al pin ADC utiliza `analogRead(A0)`. El rango de voltaje de entrada es 0 — 1.0V.

Para leer el voltaje VCC, utiliza `ESP.getVcc()` y el pin ADC debe mantenerse sin conectar. Adicionalmente, la siguiente línea tiene que añadirse al sketch:

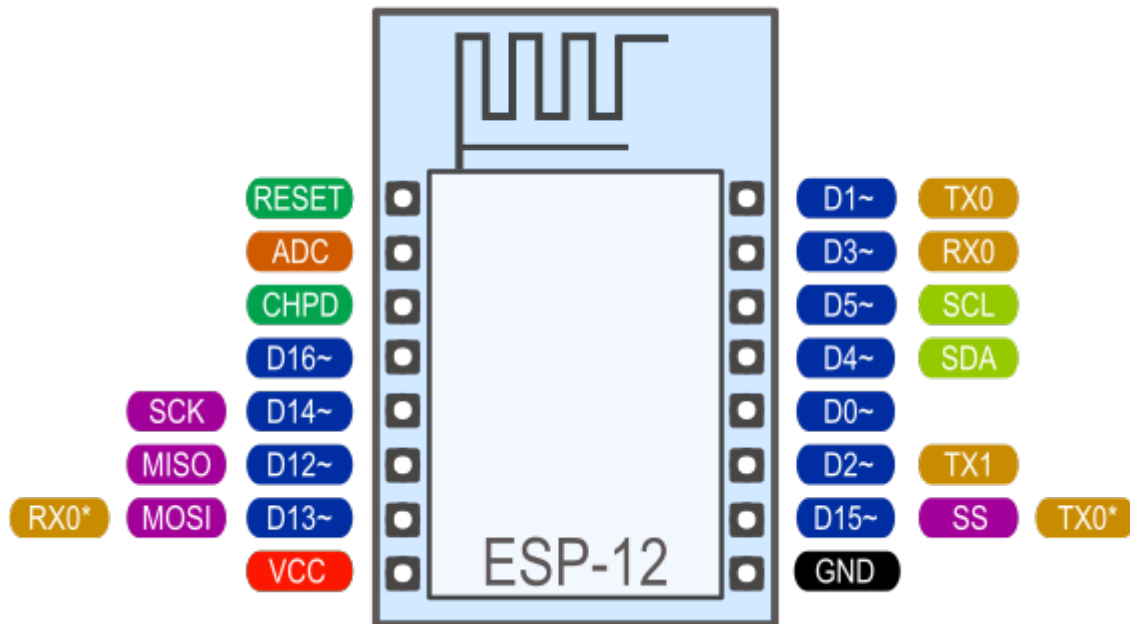


Figura 1: Funciones de los pines

```
ADC_MODE (ADC_VCC) ;
```

Esta línea tiene que estar fuera de cualquier función, por ejemplo, justo después de las líneas `#include` de tu sketch.

## 2.3 Salidas Analógicas

`analogWrite(pin, value)` activa el PWM software en el pin seleccionado. PWM puede usarse en los pines del 0 al 16. Llamar `analogWrite(pin, 0)` para desactivar PWM en el pin. `value` puede estar en un rango desde 0 a `PWMRANGE`, el cual es igual a 1023 por defecto. El rango PWM puede cambiarse llamando a `analogWriteRange(new_range)`.

La frecuencia PWM es de 1kHz por defecto. Llama a `analogWriteFreq(new_frequency)` para cambiar la frecuencia.

## 2.4 Tiempos y retrasos (delays)

`millis()` y `micros()` devuelven el número de milisegundos y microsegundos transcurridos desde el reset, respectivamente.

`delay(ms)` pausa el sketch por un número determinado de milisegundos y permite correr a las tareas WiFi y TCP/IP. `delayMicroseconds(us)` pausa por un número dado de microsegundos.

Recuerda que hay mucho código que necesita correr en el chip además del sketch cuando el WiFi está conectado. Las librerías WiFi y TCP/IP tienen una oportunidad de manejar cualquier evento pendiente cada vez que la función `loop()` se completa o cuando se llama a `delay`. Si en algún sitio de `loop` tu sketch toma mucho tiempo (>50ms)

sin llamar a `delay`, deberías considerar añadir una llamada a la función `delay` para mantener la pila WiFi corriendo adecuadamente.

Existe también la función `yield()`, la cual es equivalente a `delay(0)`. La función `delayMicroseconds`, por otro lado, no libera el control para otras tareas, por lo tanto no se recomienda realizar retrasos de más de 20 milisegundos.

## 2.5 Serial

El objeto `Serial` funciona del mismo modo que en un Arduino normal. Aparte el hardware FIFO (128 bytes para TX y RX) `Serial` tiene un buffer adicional de 256-byte para TX y RX. Ambos transmiten y reciben por medio de una interrupción. Las funciones de lectura y escritura solo bloquean la ejecución del sketch cuando el respectivo FIFO/buffer está lleno/vacío. Nota: la longitud del buffer adicional de 256-bit puede ser personalizado.

`Serial` utiliza UART0, el cual está mapeado a los pines GPIO1 (TX) y GPIO3 (RX). El `Serial` puede ser redirigido a GPIO15 (TX) y GPIO13 (RX) llamando a `Serial.swap()` después de `Serial.begin`. Llamando a `swap` otra vez se mapea UART0 de nuevo a GPIO1 y GPIO3.

`Serial1` utiliza UART1, el pin de TX es GPIO2. UART1 no puede utilizarse para recibir datos porque normalmente el pin de RX está ocupado en el flaseo del chip. Para utilizar `Serial1`, llama `Serial1.begin(baudrate)`.

Si `Serial1` no se utiliza y `Serial` no se intercambia, TX para UART0 puede ser mapeado a GPIO2 en su lugar llamando `Serial.set_tx(2)` después de `Serial.begin` o directamente con `Serial.begin(baud, config, mode, 2)`.

Por defecto la salida de diagnóstico de las librerías WiFi están desactivadas cuando llamas a `Serial.begin`. Para activar la salida de debug, llama `Serial.setDebugOutput(true)`. Para redirigir la salida de debug a `Serial1`, llama `Serial1.setDebugOutput(true)`.

También puedes utilizar `Serial.setDebugOutput(true)` para activar la salida de la función `printf()`.

El método `Serial.setRxBufferSize(size_t size)` permite definir el tamaño del buffer de recepción. El valor por defecto es 256.

Ambos objetos `Serial` y `Serial1` soportan 5, 6, 7, 8 data bits, odd (O), even (E), y no (N) parity, y 1 o 2 stop bits. Para definir el modo deseado, llama `Serial.begin(baudrate, SERIAL_8N1)`, `Serial.begin(baudrate, SERIAL_6E2)`, etc.

Se ha implementado un nuevo método en ambos `Serial` y `Serial1` para obtener la configuración actual de velocidad. Para obtener la velocidad actual, llama `Serial.baudRate()`, `Serial1.baudRate()`. Obtendrá un `int` con la velocidad actual. Por ejemplo:

```
// Establece la velocidad a 57600
Serial.begin(57600);

// Obtiene la velocidad actual
int br = Serial.baudRate();

// Imprimirá "Serial is 57600 bps"
Serial.printf("Serial is %d bps", br);
```

Ambos objetos `Serial` y `Serial1` son instancias de la clase `HardwareSerial`.

Se ha realizado también una librería oficial de ESP8266 de la librería *Software Serial*, ver este [pull request](#).

Nota: es una implementación **solo para tarjetas basadas en ESP8266** y no funcionará con otras tarjetas Arduino.

Para detectar la velocidad en baudios desconocida de los datos que entran en el puerto serie, use `Serial.detectBaudrate(time_t timeoutMillis)`. Este método intenta detectar la velocidad de transmisión en

baudios para un máximo de tiempo `timeoutMillis` en ms. Devuelve cero si no se detectó la velocidad en baudios, o de lo contrario la velocidad en baudios detectada. La función `detectBaudrate()` se puede invocar antes de llamar a `Serial.begin()`, ya que no necesita el búfer de recepción ni los parámetros de `SerialConfig`.

El UART no puede detectar otros parámetros como el `start-` o `stopbits`, número de Data bits o paridad.

La detección no cambia a si mismo la velocidad, tras la detección debes establecerla como normalmente utilizando `Serial.begin(detectedBaudrate)`.

La detección es muy rápida, solo requiere unos pocos bytes entrantes.

`SerialDetectBaudrate.ino` es un ejemplo completo de uso.

## 2.6 Progmem

Las características de la memoria de programa funcionan de la misma forma que en un Arduino normal; colocando datos de solo lectura y cadenas en la memoria de solo lectura libera la pila de su aplicación. La diferencia mas importante es que en el ESP8266 las cadenas literales no se agrupan. Esto significa que la misma cadena literal definida dentro de un `F(«»)` y / o `PSTR(«»)` tomará espacio para cada instancia en el código. Por lo tanto, tendrá que administrar el duplicado de strings usted mismo.

Hay una macro de ayuda adicional para que sea más fácil pasar cadenas `const PROGMEM` a métodos que toman un `__FlashStringHelper` llamada `FPSTR()`. Su uso ayudará a que sea más fácil juntar cadenas. No agrupando cadenas...

```
String response1;
response1 += F("http:");
...
String response2;
response2 += F("http:");
```

utilizando `FPSTR` sería...

```
const char HTTP[] PROGMEM = "http:";
...
{
  String response1;
  response1 += FPSTR(HTTP);
  ...
  String response2;
  response2 += FPSTR(HTTP);
}
```

### 3.1 WiFi(librería ESP8266WiFi)

La librería ESP8266WiFi ha sido desarrollada basándose en el SDK ESP8266, utilizando la nomenclatura convencional y la filosofía de la funcionalidad de la [Librería de la Shield Arduino WiFi](#). Con el tiempo, la riqueza de las funciones WiFi portadas del desde el SDK ESP8266 a esta librería superaron a las APIs de la librería de la Shield WiFi y se hizo evidente que tenemos que proporcionar documentación por separado sobre lo que es nuevo y extra.

*Documentación de la librería ESP8266WiFi*

### 3.2 Ticker

Es una librería para llamar a funciones repetidas cada cierto periodo de tiempo. [Dos ejemplos](#) incluidos.

Actualmente no se recomienda realizar operaciones de bloqueo de IO (network, serial, file) desde una llamada a la función Ticker. En su lugar, establece una bandera dentro de la llamada a ticker y chequea por esta bandera desde dentro de la función loop.

Aquí hay una librería que simplifica el uso de Ticker y evita el reset WDT: [TickerScheduler](#)

### 3.3 EEPROM

Este es un poco diferente de la clase estándar EEPROM. Necesitas llamar a `EEPROM.begin(size)` antes de iniciar a leer o escribir, size es el número de bytes que desea utilizar. Size puede estar entre 4 y 4096 bytes.

`EEPROM.write` no escriba a la flash inmediatamente, en su lugar llame `EEPROM.commit()` cada vez que desee guardar cambios en la flash. `EEPROM.end()` también guardará y liberará la copia RAM de contenido EEPROM.

La librería EEPROM utiliza un sector de la flash justo después de SPIFFS.

[Tres ejemplos](#) incluidos.

## 3.4 I2C (librería Wire)

La librería Wire actualmente soporta el modo maestro hasta aproximadamente 450 KHz. antes de utilizar I2C, los pines para SDA y SCL necesitan ser establecidos llamando `Wire.begin(int sda, int scl)`, p.ej. `Wire.begin(0, 2)` en ESP-01, de lo contrario, los pines por defecto son 4(SDA) y 5(SCL).

## 3.5 SPI

La librería SPI soporta por completo la API Arduino SPI incluyendo las transacciones, incluyendo la fase de ajuste (CPHA). El ajuste de Clock polarity (CPOL) no está soportado, todavía (SPI\_MODE2 y SPI\_MODE3 no funciona).

Los pines SPI usuales son:

- `MOSI = GPIO13`
- `MISO = GPIO12`
- `SCLK = GPIO14`

Hay un modo extendido donde puedes intercambiar los pines normales a los pines hardware SPI0. Esto se activa llamando `SPI.pins(6, 7, 8, 0)` antes de llamar a `SPI.begin()`. Los pines cambiarían a:

- `MOSI = SD1`
- `MISO = SD0`
- `SCLK = CLK`
- `HWCS = GPIO0`

De este modo se liberan los pines SPI con el controlador que lee el código del programa desde la flash y se controla por un árbitro de hardware (la flash tiene siempre alta prioridad). De este modo el CS será controlado por hardware y como ya no puede manejar la línea CS con un GPIO nunca sabrá realmente cuándo el árbitro le otorgará acceso al bus, por este motivo debe dejar que maneje el CS automáticamente.

## 3.6 SoftwareSerial

Una librería portada a ESP8266 de la librería SoftwareSerial hecha por Peter Lerup (@plerup) soporta velocidades hasta 115200 y múltiples instancias de SoftwareSerial. Ver <https://github.com/plerup/espsoftwareserial> si desea sugerir una mejora o abrir un issue relacionado con SoftwareSerial.

## 3.7 APIs específicas de ESP

Algunas APIs específicas de ESP relacionadas a deep sleep, RTC y memoria flash están disponibles en el objeto `ESP`.

`ESP.deepSleep(microseconds, mode)` pondrá el chip en deep sleep (sueño profundo - ahorro de energía). `mode` puede ser `WAKE_RF_DEFAULT`, `WAKE_RFCAL`, `WAKE_NO_RFCAL`, `WAKE_RF_DISABLED`. (Se necesita unir GPIO16 a RST para despertar del deep sleep). El chip puede dormir como mucho `ESP.deepSleepMax()` microsegundos.

`ESP.deepSleepInstant(microseconds, mode)` funciona similar a `ESP.deepSleep` pero duerme instantáneamente sin esperar a que el WiFi se apague.

`ESP.rtcUserMemoryWrite(offset, &data, sizeof(data))` y `ESP.rtcUserMemoryRead(offset, &data, sizeof(data))` permite a los datos ser almacenados y



recuperados de la memoria de usuario RTC del chip, respectivamente. `offset` se mide en bloques de 4 bytes y puede variar de 0 a 127 bloques (el tamaño total de la memoria RTC es de 512 bytes). `data` debe estar alineado con 4 bytes. Los datos almacenados pueden conservarse entre ciclos de sueño profundo, pero pueden perderse después de apagar y encender el chip. Los datos almacenados en los primeros 32 bloques se perderán después de realizar una actualización de OTA, ya que son utilizados por el núcleo interno.

`ESP.restart()` reinicia la CPU.

`ESP.getResetReason()` devuelve un String conteniendo la última razón de reset en un formato legible por un humano.

`ESP.getFreeHeap()` devuelve el tamaño libre de la pila.

`ESP.getHeapFragmentation()` devuelve la métrica de fragmentación (0% está limpio, más de ~50% no es inofensivo).

`ESP.getMaxFreeBlockSize()` devuelve el bloque de ram asignable máximo con respecto a la fragmentación de la pila.

`ESP.getChipId()` devuelve el ID del chip ESP8266 como un 32-bit integer.

`ESP.getCoreVersion()` devuelve un String con la versión del core.

`ESP.getSdkVersion()` devuelve la versión del SDK como un char.

`ESP.getCpuFreqMHz()` devuelve la frecuencia de la CPU en MHz como un unsigned 8-bit integer.

`ESP.getSketchSize()` devuelve el tamaño del actual sketch como un unsigned 32-bit integer.

`ESP.getFreeSketchSpace()` devuelve el espacio libre de sketch como un unsigned 32-bit integer.

`ESP.getSketchMD5()` devuelve una String con el MD5 (en minúscula) del actual sketch sketch.

`ESP.getFlashChipId()` devuelve el ID del chip flash como un 32-bit integer.

`ESP.getFlashChipSize()` devuelve el tamaño del chip flash, en bytes, como lo ve el SDK (puede ser menor que el tamaño real).

`ESP.getFlashChipRealSize()` devuelve el tamaño real del chip, en bytes, basado en el ID del chip flash.

`ESP.getFlashChipSpeed(void)` devuelve la frecuencia del chip flash, en Hz.

`ESP.getCycleCount()` devuelve la cuenta de ciclos de instrucciones de la CPU desde el arranque como un unsigned 32-bit. Esto es útil para tiempos precisos de acciones muy cortas, como bit banging.

`ESP.getVcc()` puede usarse para medir el voltaje suministrado. ESP necesita reconfigurar el ADC al inicio para poder tener esta característica disponible. Añade la siguiente línea en lo alto de tu sketch para utilizar `getVcc`:

```
ADC_MODE(ADC_VCC);
```

El pin TOUT debe estar desconectado en este modo.

Nota: por defecto ADC está configurado para leer del pin TOUT pin utilizando `analogRead(A0)`, y `ESP.getVcc()` no está disponible.

## 3.8 Respondedor mDNS y DNS-SD (librería ESP8266mDNS)

Permite al sketch responder a llamadas multicast DNS para nombres de dominios como «foo.local», y llamadas DNS-SD (descubrimiento de servicios). Ver el ejemplo incluido para mas detalle.

## 3.9 Respondedor SSDP (ESP8266SSDP)

SSDP es otro protocolo de servicio de descubrimiento, suportado en Windows. Ver ejemplo incluido para referencia.

## 3.10 Servidor DNS (librería DNSServer)

Implementa un servidor simple DNS que puede usarse en ambos modos STA y AP. Actualmente el servidor DNS soporta solo un dominio (para otros dominios responde con NXDOMAIN o un código de estatus personalizado). Con esto, los clientes pueden abrir un servidor web corriendo en el ESP8266 utilizando un nombre de dominio, en vez de una dirección IP.

## 3.11 Servo

Esta biblioteca permite la capacidad de controlar motores servo RC (hobby). Admite hasta 24 servos en cualquier pin de salida disponible. Por definición, los primeros 12 servos usarán Timer0 y actualmente esto no interferirá con ningún otro soporte. Los conteos de servos superiores a 12 utilizarán Timer1 y las funciones que lo utilizan se verán afectadas. Si bien muchos servomotores RC aceptarán el pin de datos IO de 3.3V de un ESP8266, la mayoría no podrá funcionar a 3.3v y requerirá otra fuente de alimentación que coincida con sus especificaciones. Asegúrese de conectar los cables entre el ESP8266 y la fuente de alimentación del servomotor.

## 3.12 Librería mejorada EEPROM para ESP (ESP\_EEPROM)

Una biblioteca mejorada para la EEPROM de ESPxxxx. Utiliza la memoria flash de acuerdo con la biblioteca estándar ESP EEPROM, pero reduce el reflash, por lo que reduce el desgaste y mejora el rendimiento de commit().

Como las acciones en el flash deben detener las interrupciones, un reflash de la EEPROM podría afectar notoriamente cualquier cosa usando PWM, etc.

## 3.13 Otras librerías (no incluidas con el IDE)

Las bibliotecas que no dependen del acceso a bajo nivel a los registros AVR deberían funcionar bien. Aquí hay algunas bibliotecas que se verificó que funcionan:

- [Adafruit\\_ILI9341](#) - Adafruit ILI9341 para el ESP8266
- [arduinoVNC](#) - Cliente VNC para Arduino
- [arduinoWebSockets](#) - Servidor y cliente WebSocket compatible con ESP8266 (RFC6455)
- [aREST](#) - Manejador de la librería REST API.
- [Blynk](#) - IoT framework sencillo para Makers (comprueba la [página de inicio rápido](#)).
- [DallasTemperature](#)
- [DHT-sensor-library](#) - Librería Arduino para el sensor DHT11/DHT22 de temperatura y humedad. Descarga la última librería v1.1.1 y no serán necesarios cambios. Las versiones antiguas deben inicializar el DHT como sigue: `DHT dht (DHTPIN, DHTTYPE, 15)`
- [DimSwitch](#) - Control electrónico regulable de balastos para luces de tubo fluorescentes remotamente como si se usara un interruptor de pared.

- [Encoder](#) - Librería Arduino para encoders rotatorios. Versión 1.4 soporta ESP8266.
- [esp8266\\_mdns](#) - Llamadas y respuestas mDNS en esp8266. O dicho de otro modo: Un cliente mDNS o librería de cliente Bonjour para el ESP8266.

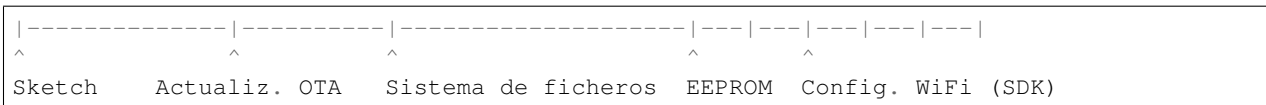
[ESP-NOW](#) - Wrapper lib para ESP-NOW (Ver #2227) - [ESPAsyncTCP](#) - Librería asíncrona TCP para ESP8266 y ESP32/31B - [ESPAsyncWebServer](#) - Librería de Servidor Web asíncrono para ESP8266 y ESP32/31B - [Homie for ESP8266](#) - Arduino framework para ESP8266 implementando Homie, una convención MQTT para IoT. - [NeoPixel](#) - Librería de Neopixel de Adafruit, ahora con soporte para el ESP8266 (utiliza la versión 1.0.2 o superior desde el Gestor de librerías de Arduino). - [NeoPixelBus](#) - Librería de Neopixel para Arduino compatible con ESP8266. Utiliza el «DmaDriven» o «UartDriven» branches para ESP8266. Incluye soporte de color HSL y mas. - [PubSubClient](#) - Librería MQTT por @Imroy. - [RTC](#) - Librería Arduino para DS1307 y DS3231 compatible con ESP8266. - [Soulliss, Smart Home](#) - Framework para Smart Home basado en Arduino, Android y openHAB. - [ST7735](#) - Librería de ST7735 de Adafruit modificada para ser compatible con ESP8266. Solo asegúrate de modificar los pines en el ejemplo por los todavía específicos de AVR. - [Task](#) - Librería no preventiva de multitarea de Arduino. Si bien es similar a la biblioteca Ticker incluida, esta librería fue diseñada para mantener la compatibilidad con Arduino. - [TickerScheduler](#) - Librería que provee un simple planificador para Ticker para prevenir el reset WDT. - [Teleinfo](#) - Librería del contador de energía genérico francés para leer los datos de monitorización de la energía Teleinfo como son consumo, contrato, potencia, periodo, ... Esta librería es de plataforma cruzada ESP8266, Arduino, Particle, y simple C++. [Post](#) dedicado francés en el blog del autor y toda la información [Teleinfo](#) también disponible. - [UTFT-ESP8266](#) - Librería para pantallas UTFT con soporte para ESP8266. Solo pantallas con soporte serial interface (SPI) por ahora (no 8-bit parallel mode, etc). También incluye soporte para el controlador hardware SPI de el ESP8266. - [WiFiManager](#) - Gestor de conexión WiFi con portal cautivo Web. Si no puede conectarse, se iniciará en modo AP y un portal de configuración donde podrás introducir tus credenciales WiFi. - [OneWire](#) - Librerías para chips Dallas/Maxim 1-Wire. - [Adafruit-PCD8544-Nokia-5110-LCD-Library](#) - Librería de PCD8544 de Adafruit para el ESP8266. - [PCF8574\\_ESP](#) - Una librería muy simple para utilizar el expansor de GPIOs PCF8574/PCF8574A I2C 8-pin. - [Dot Matrix Display Library 2](#) - Librería Freetronics DMD y pantalla Generic 16 x 32 P10 style Dot Matrix. - [SdFat-beta](#) - Librería para tarjetas SD con soporte para nombres largos, SPI basado en software y hardware y mucho mas. - [FastLED](#) - Una librería para controlar fácil y eficientemente una amplia variedad de chipsets LED, como el Neopixel (WS2812B), DotStar, LPD8806 y algunos mas. Incluye desvanecimiento, gradiente, funciones de conversión de color. - [OLED](#) - Una librería para controlar pantallas OLED conectadas con I2C. Testeado con pantallas OLED gráficas de 0.96 pulgadas. - [MFRC522](#) - Una librería para utilizar el lector/escritor de tags RFID Mifare RC522. - [Ping](#) - Permite al ESP8266 hacer ping a una máquina remota. - [AsyncPing](#) - Librería totalmente asíncrona de Ping (tiene estadísticas completas ping y direcciones hardware MAC).



### 4.1 Esquema de la memoria flash

Aunque el sistema de archivos está almacenado en el mismo chip flash que el sketch, la programación de un nuevo boceto no modificará el contenido del sistema de archivos. Esto permite utilizar el sistema de archivos para almacenar datos del sketch, archivos de configuración o contenido para el servidor web.

El siguiente diagrama ilustra el esquema o plantilla utilizado por el entorno Arduino:



El tamaño del sistema de ficheros depende del tamaño del chip flash. Dependiendo de que tarjeta se ha seleccionado en el IDE, tendrás las siguientes opciones de tamaño flash:

Tarjeta	tañano chip flash, bytes	Tamaño sistem. fich, bytes
Generic module	512k	64k, 128k
Generic module	1M	64k, 128k, 256k, 512k
Generic module	2M	1M
Generic module	4M	1M, 2M, 3M
Adafruit HUZZAH	4M	1M, 2M, 3M
ESPRESSO Lite 1.0	4M	1M, 2M, 3M
ESPRESSO Lite 2.0	4M	1M, 2M, 3M
NodeMCU 0.9	4M	1M, 2M, 3M
NodeMCU 1.0	4M	1M, 2M, 3M
Olimex MOD-WIFI-ESP8266(-DEV)	2M	1M
SparkFun Thing	512k	64k
SweetPea ESP-210	4M	1M, 2M, 3M
WeMos D1 & D1 mini	4M	1M, 2M, 3M
ESPduino	4M	1M, 2M, 3M
WiFiduino	4M	1M, 2M, 3M

**Nota** para utilizar funciones del sistema de ficheros en el sketch, añade la siguiente línea include al sketch:

```
#include "FS.h"
```

## 4.2 Limitaciones del sistema de ficheros

La implementación del sistema de archivos para ESP8266 tuvo que acomodarse a las restricciones del chip, entre las cuales está su RAM limitada. SPIFFS fue seleccionado porque está diseñado para sistemas pequeños, pero tiene como coste algunas simplificaciones y limitaciones.

Primero, por detrás, SPIFFS no soporta directorios, solo almacena una lista «plana» de ficheros. Pero en contra de un sistema de ficheros tradicional, el caracter «slash» '/' está permitido en los nombres de ficheros, por lo que las funciones que se ocupan de listar directorios (por ejemplo, `openDir("/website")`) básicamente solo filtra los nombres de archivo y conserva los que comienzan con el prefijo solicitado (`/website/`). En términos prácticos, eso hace poca diferencia sin embargo.

Segundo, existe una limitación a 32 caracteres en total en los nombres de ficheros. Un caracter '\0' está reservado para la cadena C de terminación, lo que nos deja 31 caracteres para utilizar.

Combinado, significa que se recomienda mantener los nombres de archivo cortos y no usar directorios profundamente anidados, como la ruta completa de cada archivo (incluido directorios, caracteres '/', nombre base, punto y extensión) tiene que ser 31 caracteres como máximo. Por ejemplo, el nombre de archivo `/website/images/bird_thumbnail.jpg` tiene 34 caracteres y causará problemas si se utiliza, por ejemplo en `exists()` o en caso de que otro archivo comience con los mismos primeros 31 caracteres.

**Peligro:** Ese límite se alcanza fácilmente y si se ignora, los problemas podrían pasar desapercibidos porque no aparecerá ningún mensaje de error en la compilación ni en el tiempo de ejecución.

Para mas detalles sobre la implementación interna de SPIFFS, ver el [fichero readme SPIFFS](#).

## 4.3 Subiendo ficheros al sistema de archivos

*ESP8266FS* es una herramienta que se integra en el IDE de Arduino. Añadiendo una nueva casilla al menú *Herramientas* para subir el contenido del directorio «data» del sketch al sistema de ficheros flash del ESP8266.

- Descarga la herramienta: <https://github.com/esp8266/arduino-esp8266fs-plugin/releases/download/0.3.0/ESP8266FS-0.3.0.zip>.
- En el directorio de sketches de Arduino, crea el directorio `tools` si no existe todavía.
- Descomprime la herramienta en el directorio `tools` (la ruta debe quedar `<home_dir>/Arduino/tools/ESP8266FS/tool/esp8266fs.jar`).
- Reinicia el IDE de Arduino.
- Abre el sketch (o crea uno nuevo y sálvalo)
- Ve al directorio del sketch (selecciona Programa > Mostrar carpeta de programa)
- Crea un directorio llamado `data` y algún fichero que quieras tener en el sistema de ficheros.
- Asegúrate de tener tu tarjeta seleccionada, el puerto (COM, tty, etc) y cierra el Monitor Serie.
- Selecciona Herramientas > ESP8266 Sketch Data Upload. Debería comenzar la subida de los ficheros a sistema de ficheros flash del ESP8266. Cuando acabe, la barra de estado del IDE mostrará el mensaje SPIFFS Image Uploaded.

## 4.4 Objeto sistema de ficheros *SPIFFS*

### 4.4.1 begin

```
SPIFFS.begin()
```

Este método monta el sistema de ficheros SPIFFS. Debe ser llamado antes de usar cualquier otro API del sistema de ficheros. Devuelve *true* si el sistema de archivos se ha montado satisfactoriamente, *false* en caso contrario.

### 4.4.2 end

```
SPIFFS.end()
```

Este método desmonta el sistema de ficheros SPIFFS. Utiliza este método antes de realizar una actualización OTA del SPIFFS.

### 4.4.3 format

```
SPIFFS.format()
```

Formatea el sistema de ficheros. Se puede llamar antes o después de llamar `begin`. Devuelve *verdadero* si el formateo tuvo éxito.

### 4.4.4 open

```
SPIFFS.open(path, mode)
```

Abre un fichero. `path` debe ser un camino absoluto comenzando con un slash (p.ej. `/dir/filename.txt`). `mode` es una palabra que especifica el modo de acceso. Puede ser una de las siguientes: «r», «w», «a», «r+», «w+», «a+». El significado de estos modos es el mismo que para la función `fopen` en C.

```
r      Abre un fichero de texto para leerlo. La secuencia se coloca en el comienzo
↳del archivo.

r+     Abre un fichero para lectura y escritura. La secuencia se coloca en el
↳comienzo del archivo.

w      Trunca en fichero con una longitud cero o crea un fichero de texto para
↳escritura.
      La secuencia se coloca en el comienzo del archivo.

w+     Abre para lectura y escritura. El fichero se crea si no existe, de lo
↳contrario se trunca.
      La secuencia se coloca en el comienzo del archivo.

a      Abre el fichero para añadir (escribiendo al final del fichero). El fichero se
↳crea si no existe.
      La secuencia se coloca al final del archivo.

a+     Abre el fichero para añadir (escribiendo al final del fichero). El fichero se
↳crea si no existe.
```

(continues on next page)

(proviene de la página anterior)

```
    La posición inicial para lectura es al comienzo del fichero, pero la salida es ↵  
↪ siempre añadida  
    al final del fichero
```

Devuelve el objeto *File*. Para comprobar si el archivo se abrió con éxito, utilice un operador booleano.

```
File f = SPIFFS.open("/f.txt", "w");  
if (!f) {  
    Serial.println("No se pudo abrir el fichero");  
}
```

### 4.4.5 exists

```
SPIFFS.exists(path)
```

Devuelve *true* si existe el archivo con la ruta indicada, *false* en caso contrario.

### 4.4.6 opendir

```
SPIFFS.opendir(path)
```

Abre un directorio en la ruta absoluta indicada. Devuelve un objeto *Dir*.

### 4.4.7 remove

```
SPIFFS.remove(path)
```

Elimina el fichero de la ruta absoluta indicada. Devuelve *true* si el fichero se borró satisfactoriamente.

### 4.4.8 rename

```
SPIFFS.rename(pathFrom, pathTo)
```

Renombra el fichero *pathFrom* a *pathTo*. La ruta debe ser absoluta. Devuelve *true* si el fichero se renombra satisfactoriamente.

### 4.4.9 info

```
FSInfo fs_info;  
SPIFFS.info(fs_info);
```

Rellena la estructura *FSInfo* con información sobre el sistema de ficheros. Devuelve *true* si tiene éxito, *false* en caso contrario.



## 4.5 Estructura de información del sistema de archivos

```
struct FSInfo {
  size_t totalBytes;
  size_t usedBytes;
  size_t blockSize;
  size_t pageSize;
  size_t maxOpenFiles;
  size_t maxPathLength;
};
```

Esta es la estructura que se rellena al usar el método `FS::info` .

- `totalBytes` — Tamaño total de datos útiles en el sistema de archivos.
- `usedBytes` — Número de bytes usado por los ficheros.
- `blockSize` — Tamaño del bloque SPIFFS.
- `pageSize` — Tamaño de la página lógica SPIFFS.
- `maxOpenFiles` — Número máximo de archivos que pueden estar abiertos simultáneamente.
- `maxPathLength` — Longitud máxima del nombre de archivo (incluido un byte cero de terminación).

## 4.6 Objeto directorio *Dir*

El propósito del objeto *Dir* es iterar sobre los ficheros dentro del directorio. Provee los métodos: `next()`, `fileName()`, y `openFile(mode)`.

El siguiente ejemplo muestra como debe utilizarse:

```
Dir dir = SPIFFS.openDir("/data");
while (dir.next()) {
  Serial.print(dir.fileName());
  File f = dir.openFile("r");
  Serial.println(f.size());
  if (dir.fileSize()) {
    File f = dir.openFile("r");
    Serial.println(f.size());
  }
}
```

### 4.6.1 next

Devuelve *true* mientras haya ficheros en el directorio para iterar. Debe llamarse antes de llamar a las funciones `fileName()`, `fileSize()` y `openFile()` .

### 4.6.2 fileName

Devuelve el nombre del archivo actual apuntado por el iterador interno.

### 4.6.3 fileSize

Devuelve el tamaño del archivo actual apuntado por el iterador interno

### 4.6.4 openFile

Este método toma el argumento *mode* que tiene el mismo significado que para la función `SPIFFS.open()`.

## 4.7 Objeto fichero *File*

Las funciones `SPIFFS.open()` y `dir.openFile()` devuelven un objeto *File*. Este objeto soporta todas las funciones de *Stream*, para que puedas usar `readBytes`, `findUntil`, `parseInt`, `println` y todos los otros métodos *Stream*.

Hay algunas funciones que son específicas del objeto *File*.

### 4.7.1 seek

```
file.seek(offset, mode)
```

Esta función se comporta como la función C `fseek`. Dependiendo de un valor de *mode*, se mueve a la posición actual en un fichero de la siguiente manera:

- Si *mode* es `SeekSet`, la posición se establece a *offset* bytes desde el comienzo del fichero.
- Si *mode* es `SeekCur`, la posición actual se mueve a *offset* bytes.
- Si *mode* es `SeekEnd`, la posición se establece a *offset* bytes desde el final del fichero.

Devuelve *true* si la posición se estableció satisfactoriamente.

### 4.7.2 position

```
file.position()
```

Devuelve la posición actual dentro del fichero, en bytes.

### 4.7.3 size

```
file.size()
```

Devuelve el tamaño del fichero, en bytes.

### 4.7.4 name

```
String name = file.name();
```

Devuelve el nombre del fichero, como `const char*`. Conviértelo a *String* para almacenarlo

### 4.7.5 close

```
file.close()
```

Cierra el fichero. Ninguna otra operación debe realizarse sobre el objeto *File* después de llamar a la función `close`.



---

## Librería ESP8266WiFi

---

Todo sobre el WiFi de ESP8266. Si está ansioso por conectar su nuevo módulo ESP8266 a la red WiFi para comenzar a enviar y recibir datos, este es un buen lugar para comenzar. Si está buscando detalles más en profundidad sobre cómo programar una funcionalidad de red WiFi específica, también se encuentra en el lugar correcto.

### 5.1 Introducción

La librería WiFi para ESP8266 ha sido desarrollada basándose en el SDK de ESP8266, usando nombres convencionales y la filosofía de funcionalidades generales de la librería WiFi de Arduino. Con el tiempo, la riqueza de las funciones WiFi del SDK de ESP8266 pasadas a ESP8266/Arduino superan a la librería WiFi de Arduino y se hizo evidente que tenemos que proporcionar documentación por separado sobre lo que es nuevo y extra.

Esta documentación lo guiará a través de varias clases, métodos y propiedades de la librería ESP8266WiFi. Si eres nuevo en C++ y Arduino, no te preocupes. Comenzaremos por conceptos generales y luego pasaremos a la descripción detallada de los miembros de cada clase en particular, incluidos los ejemplos de uso.

El alcance de la funcionalidad que ofrece la biblioteca ESP8266WiFi es bastante extensa y por lo tanto, esta descripción se ha dividido en documentos separados marcados con :arrow\_right:.

#### 5.1.1 Comienzo rápido

Esperamos que ya esté familiarizado con la carga del sketch [Blink.ino](#) en el módulo ESP8266 y obtenga el LED parpadeando. De lo contrario, compruebe [este tutorial](#) de Adafruit o [este otro gran tutorial](#) desarrollado por Sparkfun.

Para conectar el módulo ESP al WiFi (como conectar un teléfono móvil a un punto caliente), solo necesita un par de líneas de código:

```
#include <ESP8266WiFi.h>

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

(continues on next page)

(proviene de la página anterior)

```

Serial.println();

WiFi.begin("nombre-red", "contraseña-red");

Serial.print("Conectando");
while (WiFi.status() != WL_CONNECTED)
{
  delay(500);
  Serial.print(".");
}
Serial.println();

Serial.print("Conectado, dirección IP: ");
Serial.println(WiFi.localIP());
}

void loop() {}

```

En la línea `WiFi.begin("nombre-red", "contraseña-red")` reemplace `nombre-red` y `contraseña-red` con el nombre y contraseña a la red WiFi que quiere conectarse. Entonces suba el sketch al módulo ESP y abra el Monitor Serie. Deberías ver algo como:

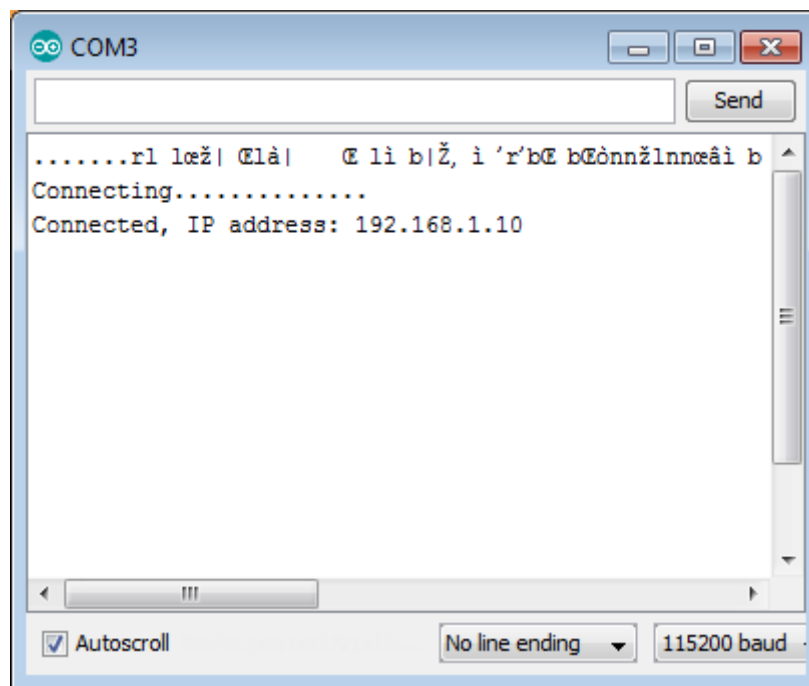


Figura 1: Registro de conexión en el Monitor Serie del IDE Arduino

¿Como funciona? En la primera línea del boceto `#include <ESP8266WiFi.h>` estamos incluyendo la librería `ESP8266WiFi`. Esta librería proporciona las rutinas específicas WiFi de ESP8266 a las que llamamos para conectarse a la red.

La conexión real a WiFi se inicia llamando al:

```
WiFi.begin("nombre-red", "contraseña-red");
```

El proceso de conexión puede demorar unos segundos, así que comprobamos que esto se complete en el siguiente

ciclo:

```
while (WiFi.status() != WL_CONNECTED)
{
  delay(500);
  Serial.print(".");
}
```

El bucle `while()` seguirá en bucle mientras `WiFi.status()` no es `WL_CONNECTED`. El ciclo saldrá solo si el estado cambia a `WL_CONNECTED`.

La última línea imprimirá la dirección IP asignada al módulo ESP por DHCP:

```
Serial.println(WiFi.localIP());
```

Si no ve la última línea sino solo más y más puntos “.....”, entonces probablemente el nombre o la contraseña de la red WiFi en el sketch son incorrectos. Verifique el nombre y la contraseña conectando desde cero a esta WiFi una PC o un teléfono móvil.

*Nota:* si la conexión se establece y luego se pierde por algún motivo, ESP se reconectará automáticamente al último punto de acceso utilizado una vez que vuelva a estar en línea. Esto se hará automáticamente mediante la librería WiFi, sin intervención del usuario.

Eso es todo lo que necesita para conectar su ESP8266 al WiFi. En los siguientes capítulos, explicaremos qué cosas interesantes se pueden hacer con ESP una vez conectados.

### 5.1.2 Quien es quien

Los dispositivos que se conectan a la red WiFi se llaman estaciones (STA). La conexión a WiFi es proporcionada por un punto de acceso (AP), que actúa como un centro para una o más estaciones. El punto de acceso en el otro extremo está conectado a una red cableada. Un punto de acceso generalmente se integra con un router para proporcionar acceso desde la red WiFi a Internet. Cada punto de acceso es reconocido por un SSID (Service Set Identifier), que esencialmente es el nombre de la red que usted selecciona cuando conecta un dispositivo (estación) al WiFi.

El módulo ESP8266 puede funcionar como una estación, por lo que podemos conectarlo a la red WiFi. Y también puede funcionar como un punto de acceso wireless (SoftAP), para establecer su propia red WiFi. Por lo tanto, podemos conectar otras estaciones a dicho módulo ESP. ESP8266 también puede operar tanto en modo estación como en modo punto de acceso. Esto proporciona la posibilidad de construir, p. ej. [redes de malla](#).

La biblioteca `ESP8266WiFi` proporciona una amplia colección de [métodos C++](#) y [propiedades o atributos](#) para configurar y operar un módulo ESP8266 en modo estación y/o punto de acceso. Se describen en los siguientes capítulos.

## 5.2 Descripción de la clase

La librería `ESP8266WiFi` se divide en varias clases. En la mayoría de los casos, al escribir el código, el usuario no está interesado en esta clasificación. Lo usamos para dividir la descripción de esta librería en piezas más manejables.

Los siguientes capítulos describen todas las llamadas a funciones ( [métodos](#) y [propiedades](#) en términos C++) enumerados en clases particulares de `ESP8266WiFi`. La descripción se ilustra con ejemplos de aplicaciones y fragmentos de código para mostrar cómo usar las funciones en la práctica. La mayoría de esta información se divide en documentos separados. Por favor, sigue para acceder a ellos.

### 5.2.1 Station

El modo estación (STA) se utiliza para conectar el módulo ESP a una red WiFi establecida por un punto de acceso.

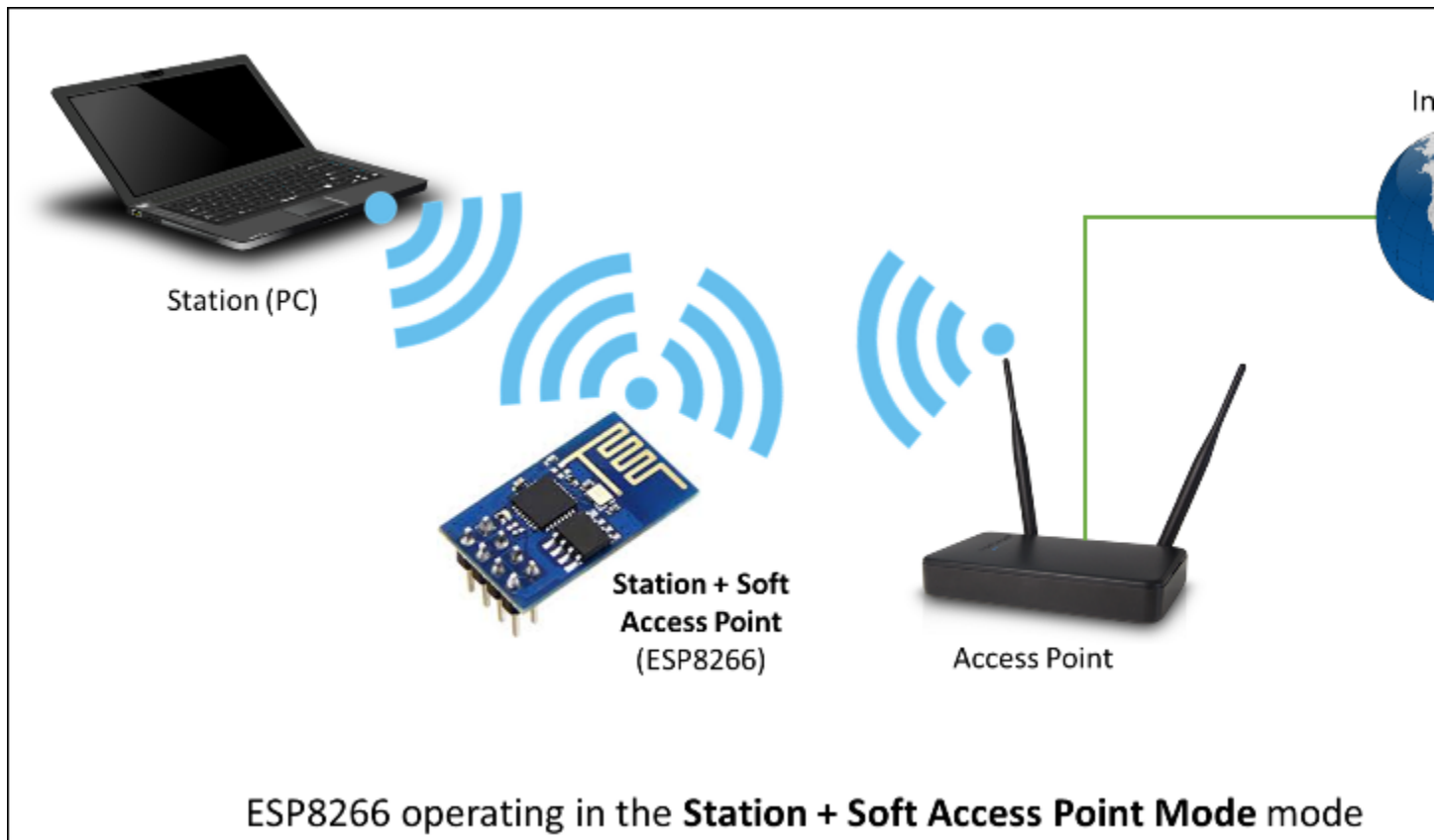


Figura 2: ESP8266 operando en modo Estación + Punto de Acceso

<b>B</b>	<b>E</b>	<b>S</b>	WiFiClient
BufferDataSource	ESP8266WiFiAPClass	SList	WiFiClientSecure
BufferedStreamDataSource	ESP8266WiFiClass	SSLContext	WiFiEventHandlerOpaque
<b>C</b>	ESP8266WiFiGenericClass	<b>U</b>	WiFiEventModeChange
ClientContext	ESP8266WiFiMulti	UdpContext	WiFiEventSoftAPModeProbeRequest
<b>D</b>	ESP8266WiFiScanClass	<b>W</b>	WiFiEventSoftAPModeStationConnect
DataSource	ESP8266WiFiSTAClass	WifiAPList_t	WiFiEventSoftAPModeStationDisconnect
	<b>P</b>		WiFiEventStationModeAuthModeChange
	ProgmemStream		WiFiEventStationModeConnected

Figura 3: Índice de clases de la librería ESP8266WiFi



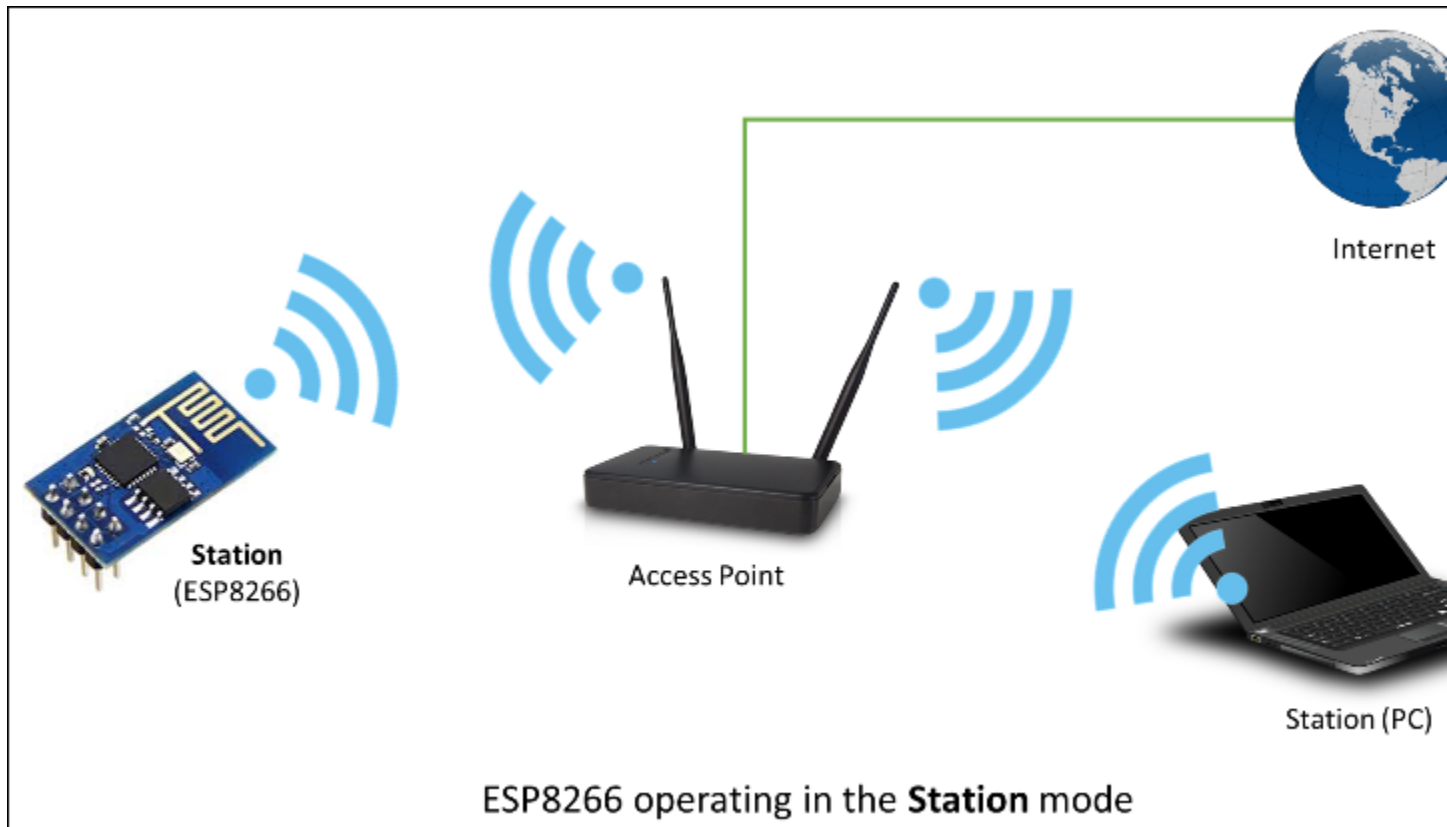


Figura 4: ESP8266 operando en modo estación

La clase de estación tiene varias características para facilitar la administración de la conexión WiFi. En caso de que se pierda la conexión, el ESP8266 se volverá a conectar automáticamente al último punto de acceso utilizado, una vez que esté nuevamente disponible. Lo mismo ocurre en el reinicio del módulo. Esto es posible ya que ESP guarda las credenciales al último punto de acceso utilizado en la memoria flash (no volátil). Usando los datos guardados, ESP también se volverá a conectar si se modificó el sketch, si el código no altera el modo WiFi o las credenciales.

Documentación clase Station

Echa un vistazo a la sección separada con ejemplos.

## 5.2.2 Punto de Acceso Wireless

Un [punto de acceso inalámbrico \(AP\)](#) es un dispositivo que proporciona acceso a la red WiFi a otros dispositivos (estaciones) y los conecta a una red cableada. ESP8266 puede proporcionar una funcionalidad similar, excepto que no tiene interfaz para una red cableada. Tal modo de operación se llama punto de acceso SoftAP. La cantidad máxima de estaciones que pueden estar simultáneamente conectadas al SoftAP puede establecerse de 0 a 8, pero por defecto es 4.

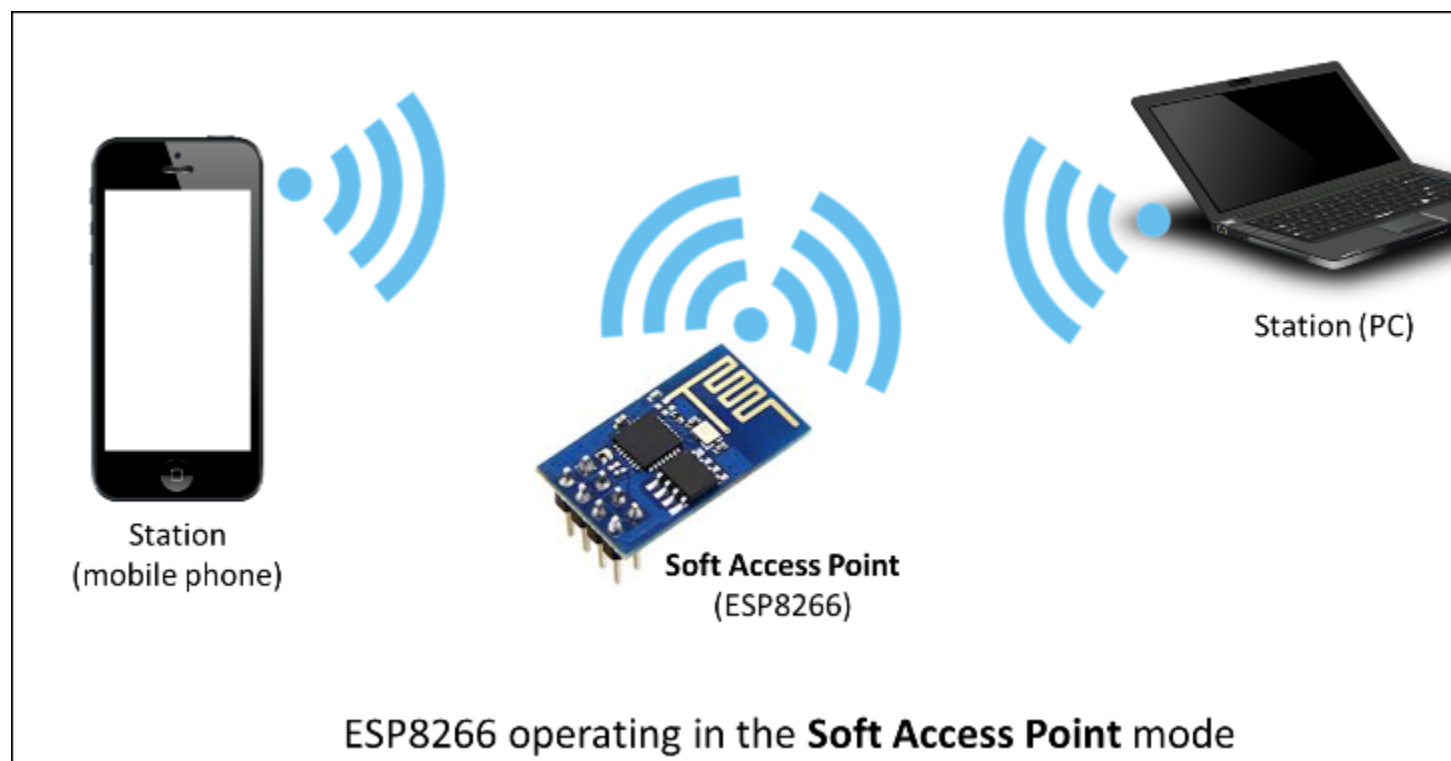


Figura 5: ESP8266 operando en modo Punto de acceso SoftAP

El modo SoftAP se usa a menudo y es un paso intermedio antes de conectar ESP a una red WiFi en modo estación. Esto es cuando el SSID y la contraseña de dicha red no se conocen por adelantado. ESP primero arranca en modo SoftAP, para que podamos conectarnos a él usando un ordenador portátil o un teléfono móvil. Luego, podemos proporcionar credenciales a la red objetivo. Una vez hecho esto, ESP se cambia al modo estación y se puede conectar al WiFi objetivo.

Otra aplicación práctica del modo SoftAP es configurar una [red mallada](#). ESP puede funcionar tanto en modo SoftAP como en modo Estación para que pueda actuar como un nodo de una red mallada.

Documentación clase Soft Access Point

Echa un vistazo a la sección separada con ejemplos.

### 5.2.3 Scan

Para conectar un teléfono móvil a un punto de acceso público, normalmente abre la aplicación de configuración de Wi-Fi, enumera las redes disponibles y elige el punto de acceso que necesita. Luego ingresa una contraseña (o no) y estás dentro. Puedes hacer lo mismo con ESP. La clase de escaneo implementa la funcionalidad del escaneo y la lista de redes disponibles en el rango.

Documentación clase Scan

Echa un vistazo a la sección separada con ejemplos.

### 5.2.4 Client

La clase Client crea `clientes` que puede acceder a servicios proporcionados por `servidores` para enviar, recibir y procesar datos.

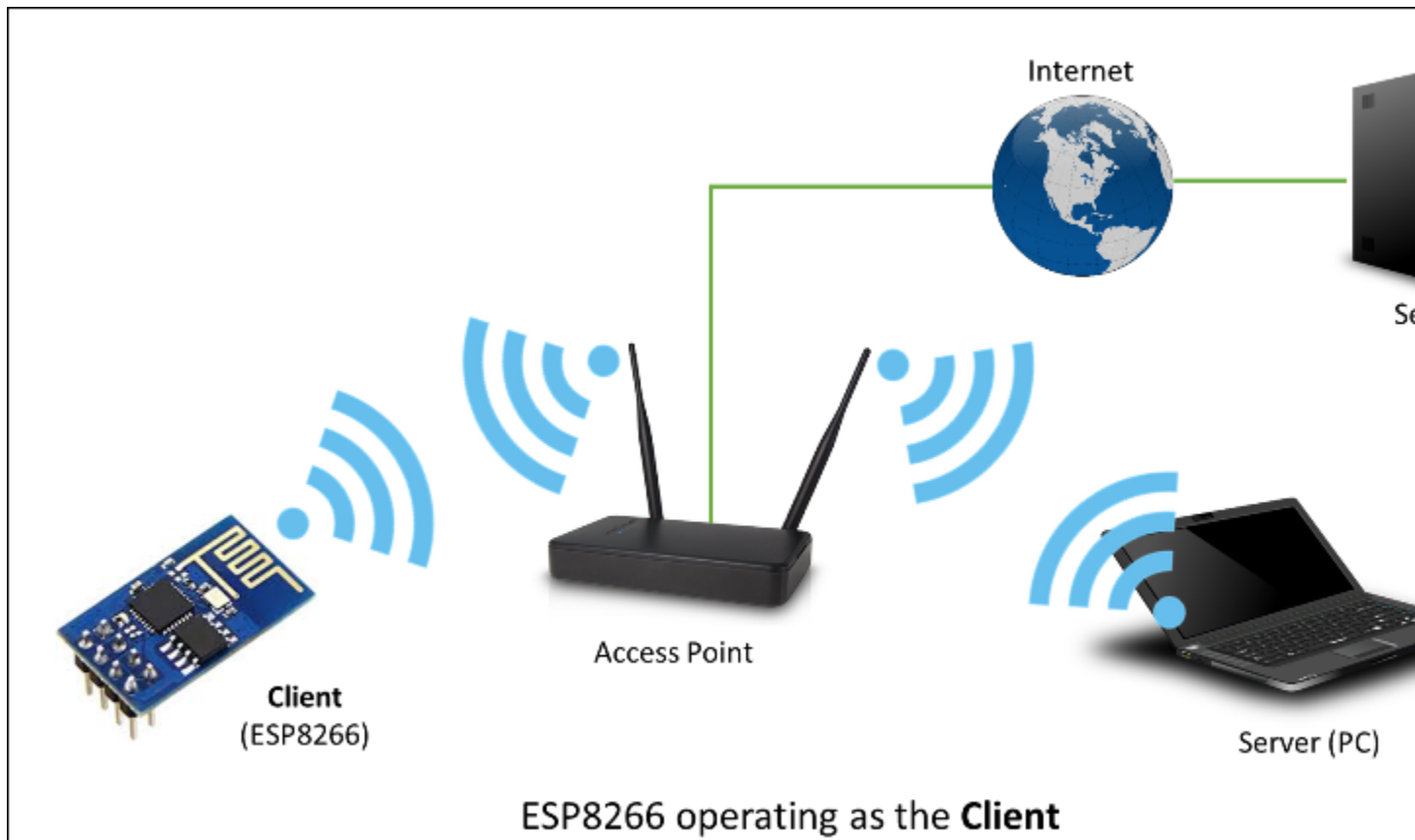


Figura 6: ESP8266 operando como Cliente

Echa un vistazo a la sección separada con ejemplos / Lista de funciones

### 5.2.5 Cliente Seguro axTLS - OBSOLETO

La siguiente sección detalla axTLS, la biblioteca TLS más antigua utilizada por el proyecto. Todavía se admite, pero generalmente no se realizarán correcciones y documentación adicionales. Consulte la siguiente sección para el objeto cliente TLS actualizado.

El cliente seguro axTLS es una extensión de la *clase Client* donde la conexión y el intercambio de datos con los servidores se hace usando un *protocolo seguro*. Es compatible con TLS 1.1. El TLS 1.2 no es compatible.

Las aplicaciones seguras tienen una sobrecarga adicional de memoria (y procesamiento) debido a la necesidad de ejecutar algoritmos de criptografía. Cuanto más fuerte sea la clave del certificado, más gastos generales se necesitan. En la práctica, no es posible ejecutar más de un único cliente seguro a la vez. El problema se refiere a la memoria RAM que no podemos agregar, el tamaño de la memoria flash por lo general no es el problema. Si desea aprender cómo se ha desarrollado la librería de *Client Secure*, qué servidores se han probado y cómo se han superado las limitaciones de la memoria, lea el fascinante informe de problemas #43.

Echa un vistazo a la sección separada con ejemplos / lista de funciones

### 5.2.6 Cliente seguro BearSSL y servidor seguro

*BearSSL::WiFiClientSecure* y *BearSSL::WiFiServerSecure* son extensiones de las clases estándar *Client* y *Server* donde la conexión y el intercambio de datos con servidores y clientes utilizan un *protocolo seguro*. Soporta TLS 1.2 utilizando una amplia variedad de cifrados modernos, hashes y tipos de clave.

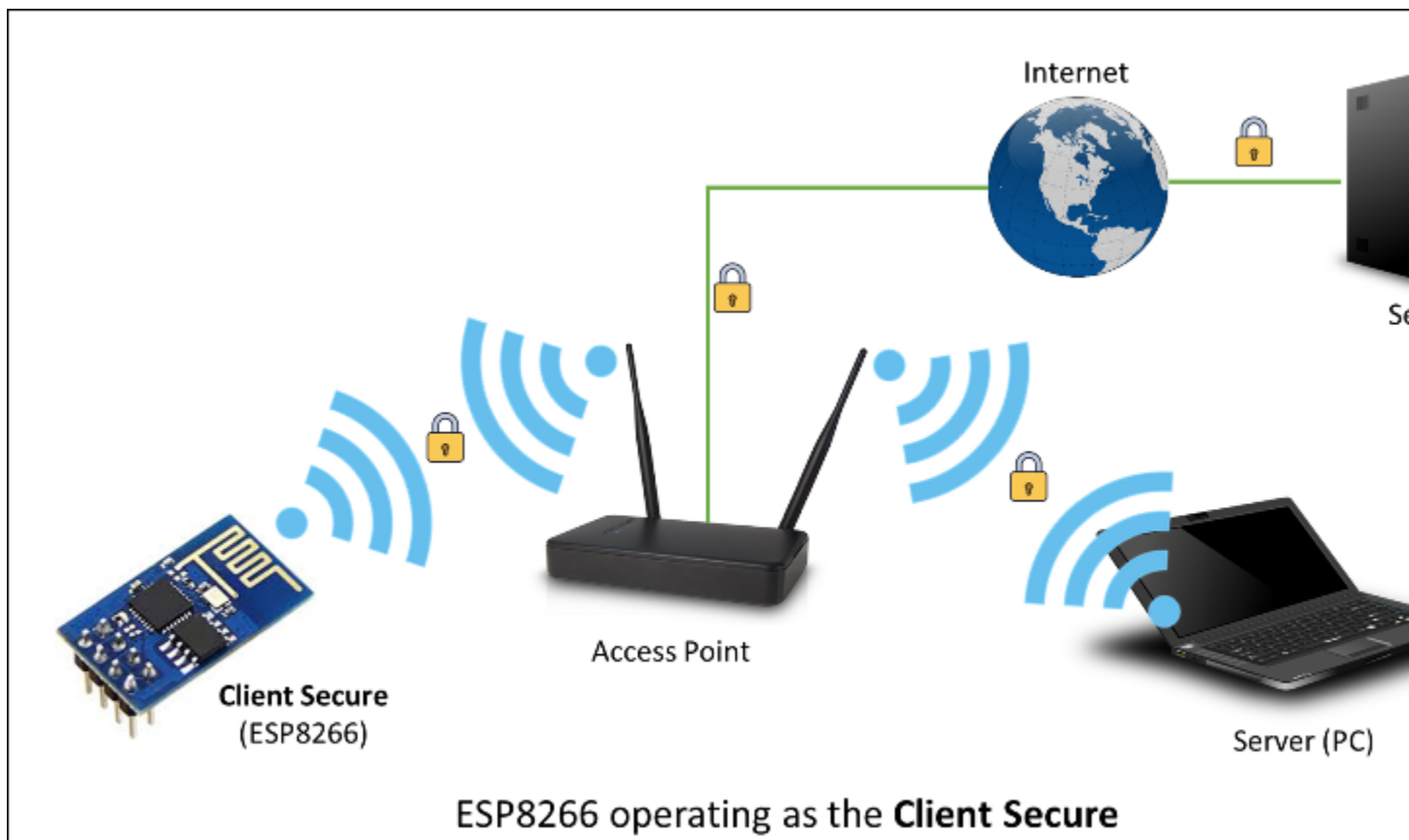


Figura 7: ESP8266 operando como Cliente seguro

Los clientes y servidores seguros requieren cantidades significativas de memoria y procesamiento adicionales para habilitar sus algoritmos criptográficos. En general, solo se puede procesar una única conexión segura de servidor o cliente a la vez, dada la poca memoria RAM presente en el ESP8266, pero existen métodos para reducir este requisito de memoria RAM que se detalla en las secciones correspondientes.

*BearSSL::WiFiClientSecure* contiene más información sobre el uso y configuración de conexiones TLS.

BearSSL::WiFiServerSecure discute el modo servidor TLS disponible. Por favor lea y entienda primero BearSSL::WiFiClientSecure ya que el servidor usa la mayoría de los mismos conceptos.

## 5.2.7 Server

La clase de Server crea [Servidores](#) que proporcionan funcionalidad a otros programas o dispositivos, llamados [Clientes](#).

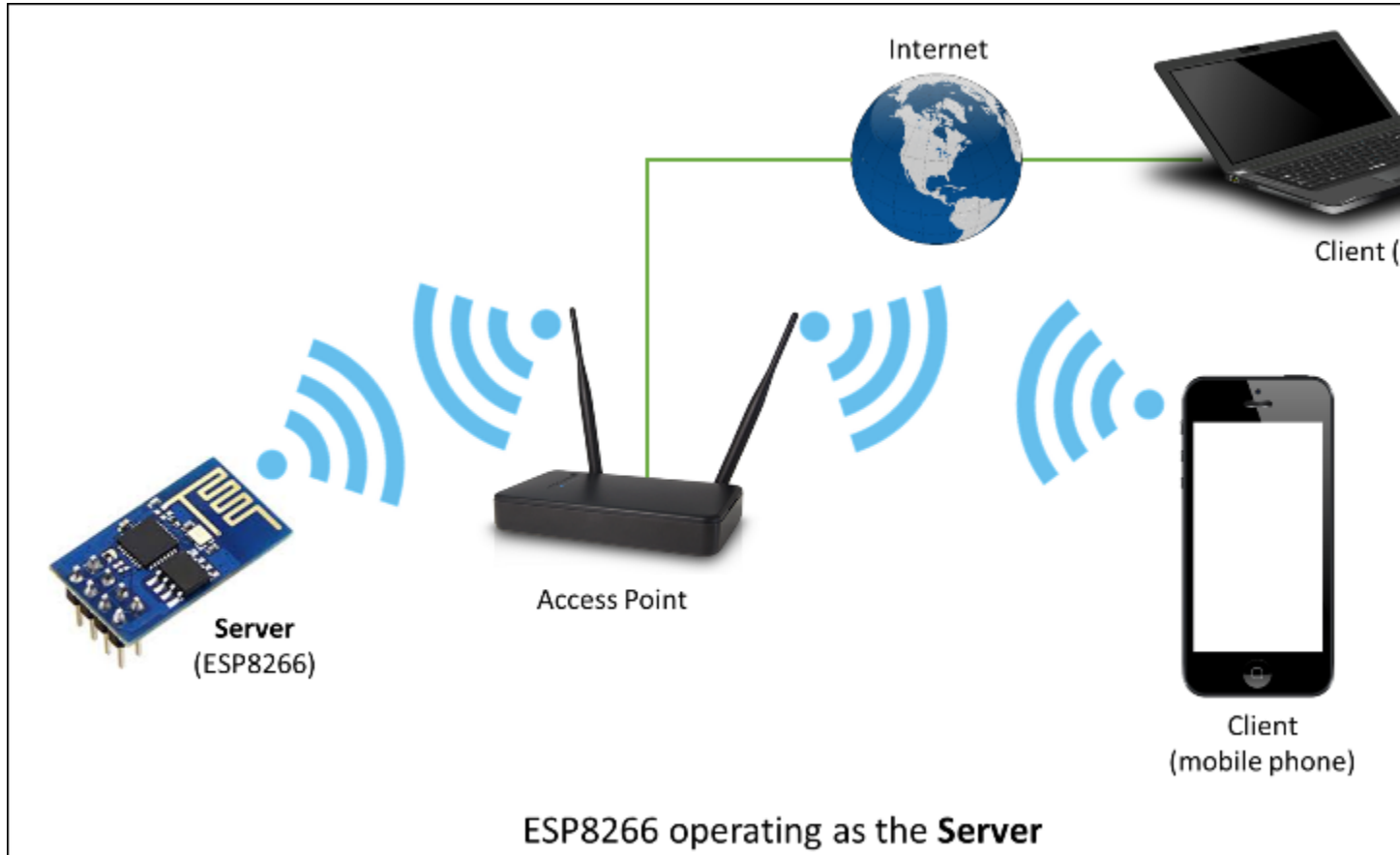


Figura 8: ESP8266 operando como Servidor

Los clientes se conectan para enviar y recibir datos y acceder a la funcionalidad provista.

Echa un vistazo a la sección separada con ejemplos / lista de funciones.

## 5.2.8 UDP

La clase UDP permite el envío y recepción de mensajes [User Datagram Protocol \(UDP\)](#). El UDP usa un modelo de transmisión simple de «disparar y olvidar» sin garantía de entrega, pedido o protección duplicada. UDP proporciona sumas de comprobación para la integridad de datos y números de puertos para direccionar diferentes funciones a la fuente y el destino del datagrama.

Echa un vistazo a la sección separada con ejemplos / lista de funciones.

## 5.2.9 Generic

Hay varias funciones ofrecidas por el SDK de ESP8266 y no están presentes en la biblioteca [Arduino WiFi](#). Si una función no encaja en una de las clases discutidas anteriormente, probablemente estará en la Clase Genérica. Entre ellas se encuentra el controlador para gestionar eventos WiFi como conexión, desconexión u obtención de una IP, cambios en el modo WiFi, funciones para gestionar el modo de suspensión del módulo, nombre de host para una resolución de dirección IP, etc.

Echa un vistazo a la sección separada con ejemplos / lista de funciones.

## 5.3 Diagnóstico

Hay varias técnicas disponibles para diagnosticar y solucionar problemas al conectarse a WiFi y mantener la conexión activa.

### 5.3.1 Comprobar los códigos devueltos

Casi todas las funciones descritas en los capítulos anteriores devuelven información de diagnóstico.

Tal diagnóstico se puede proporcionar como un simple `booleano`, `true` o `false`, para indicar el resultado de la operación. Puede verificar este resultado como se describe en los ejemplos, por ejemplo:

```
Serial.printf("Modo WiFi establecido a WIFI_STA %s\n", WiFi.mode(WIFI_STA) ? "" :  
↪ "Falló!");
```

Algunas funciones proporcionan más que solo una información binaria. Un buen ejemplo es `WiFi.status()`.

```
Serial.printf("Estado de la conexión: %d\n", WiFi.status());
```

Esta función devuelve los siguientes códigos para describir lo que está sucediendo con la conexión WiFi:

- 0 : `WL_IDLE_STATUS` cuando WiFi está en proceso de cambio de estado
- 1 : `WL_NO_SSID_AVAIL` en caso de que no se pueda alcanzar el SSID configurado
- 3 : `WL_CONNECTED` después de establecida una conexión exitosa
- 4 : `WL_CONNECT_FAILED` si la contraseña es incorrecta
- 6 : `WL_DISCONNECTED` si el módulo no está configurado en modo estación

Es una buena práctica mostrar y verificar la información devuelta por las funciones. El desarrollo de aplicaciones y la resolución de problemas serán así más fáciles.

### 5.3.2 Usar `printDiag`

Hay una función específica disponible para imprimir la información clave de diagnóstico del WiFi:

```
WiFi.printDiag(Serial);
```

Una salida de muestra de esta función se ve de la siguiente manera:

```

Mode: STA+AP
PHY mode: N
Channel: 11
AP id: 0
Status: 5
Auto connect: 1
SSID (10): sensor-net
Passphrase (12): 123!$#0&*esP
BSSID set: 0

```

Utilice esta función para proporcionar una instantánea del estado de Wi-Fi en partes del código de la aplicación, que sospecha que puede estar fallando.

### 5.3.3 Activar el diagnóstico WiFi

Por defecto, la salida de diagnóstico de las librerías WiFi están desactivadas cuando se invoca `Serial.begin`. Para habilitar nuevamente la salida de depuración, llame a `Serial.setDebugOutput(true)`. Para redirigir la salida de depuración a `Serial1`, llame a `Serial1.setDebugOutput(true)`. Para obtener más detalles sobre el diagnóstico con puertos serie, consulte la [documentación](#).

A continuación se muestra un ejemplo de salida para el sketch de muestra discutido mas arriba en *Inicio rápido* con `Serial.setDebugOutput(true)`:

```

Conectandoscandone
state: 0 -> 2 (b0)
state: 2 -> 3 (0)
state: 3 -> 5 (10)
add 0
aid 1
cnt

connected with sensor-net, channel 6
dhcp client start...
chg_B1:-40
...ip:192.168.1.10,mask:255.255.255.0,gw:192.168.1.9
.
Conectado, dirección IP: 192.168.1.10

```

El mismo sketch sin `Serial.setDebugOutput(true)` imprimirá lo siguiente:

```

Conectando...
Conectado, dirección IP: 192.168.1.10

```

### 5.3.4 Activar Debug en el IDE

El IDE Arduino provee métodos para *activar la depuración* para librerías específicas.

## 5.4 ¿Que hay dentro?

Si desea analizar en detalle qué hay dentro de la librería ESP8266WiFi, vaya directamente a la carpeta `ESP8266WiFi` del repositorio ESP8266/Arduino en GitHub.

Para facilitar el análisis, en lugar de buscar en el encabezado individual o en los archivos fuente, use una de las herramientas gratuitas para generar documentación automáticamente. El índice de clase en el capítulo de mas arriba *Descripción de clase* ha sido preparado en muy poco tiempo usando el gran [Doxygen](#), que es el herramienta estándar de facto para generar documentación a partir de fuentes anotadas de C++.

La herramienta rastrea todos los archivos de encabezado y fuente, recopilando información de los bloques de comentarios formateados. Si el desarrollador de una clase particular anotó el código, lo verá como en los ejemplos a continuación.

Si el código no está anotado, aún verá el prototipo de la función, incluidos los tipos de argumentos y puede usar los enlaces proporcionados para ir directamente al código fuente para verificarlo por su cuenta. Doxygen proporciona una navegación realmente excelente entre los miembros de la librería.

Varias clases de [ESP8266WiFi](#) no están anotadas. Al preparar este documento, [Doxygen](#) ha sido de gran ayuda para navegar rápidamente a través de casi 30 archivos que componen esta librería.



ESP8266WiFi 1

ESP8266WiFi Library Documentation

Main Page Modules **Classes** Files

Class List Class Index Class Hierarchy Class Members

ESP8266WiFi

- axTLS API
- Modules
- Classes
  - Class List
    - BufferDataSource
    - BufferedStreamDataSource
    - ClientContext
    - DataSource
    - ESP8266WiFiAPClass
    - ESP8266WiFiClass**
    - ESP8266WiFiGenericClass
    - ESP8266WiFiMulti
    - ESP8266WiFiScanClass
    - ESP8266WiFiSTAClass
    - ProgmemStream
    - SList
    - SSLContext
    - UdpContext

ESP8266WiFiClass

## ESP8266WiFiClass Class Reference

```
#include <ESP8266WiFi.h>
```

Inheritance diagram for ESP8266WiFiClass:

```

graph TD
    ESP8266WiFiClass[ESP8266WiFiClass]
    ESP8266WiFiGenericClass[ESP8266WiFiGenericClass]
    ESP8266WiFiSTAClass[ESP8266WiFiSTAClass]
    ESP8266WiFiClass --> ESP8266WiFiGenericClass
    ESP8266WiFiClass --> ESP8266WiFiSTAClass
  
```

## Public Member Functions

void printDiag (Print &dest)

- Public Member Functions inherited from ESP8266WiFiGenericClass
- Public Member Functions inherited from ESP8266WiFiSTAClass
- Public Member Functions inherited from ESP8266WiFiScanClass
- Public Member Functions inherited from ESP8266WiFiAPClass

Friends

Figura 9: Ejemplo de documentación preparada con Doxygen

```

wl_status_t ESP8266WiFiSTAClass::begin ( const char *  ssid,
                                           const char *  passphrase = NULL,
                                           int32_t      channel = 0,
                                           const uint8_t * bssid = NULL,
                                           bool         connect = true
                                           )

```

Start Wifi connection if passphrase is set the most secure supported mode will be automatically selected

#### Parameters

**ssid** const char\* Pointer to the SSID string.  
**passphrase** const char \* Optional. Passphrase. Valid characters in a passphrase must be between ASCII 32-126  
**bssid** uint8\_t[6] Optional. BSSID / MAC of AP  
**channel** Optional. Channel of AP  
**connect** Optional. call connect

#### Returns

Definition at line 97 of file [ESP8266WiFiSTA.cpp](#).

Figura 10: Ejemplo de documentación para el método begin STA por Doxygen

```

bool ESP8266WiFiSTAClass::hostname ( char * aHostname )

```

Set ESP8266 station DHCP hostname

#### Parameters

**aHostname** max length:32

#### Returns

ok

Definition at line 422 of file [ESP8266WiFiSTA.cpp](#).

Figura 11: Ejemplo de documentación para la propiedad hostname por Doxygen

```
uint8_t WiFiUDP::begin ( uint16_t port )
```

Definition at line 77 of file `WiFiUdp.cpp`.

Figura 12: Ejemplo de documentación para el método begin UDP (no anotado en el código) por Doxygen



### 6.1 Introducción

La actualización OTA (Over the Air - Por el aire) es el proceso de carga del firmware en el módulo ESP mediante una conexión Wi-Fi en lugar de un puerto serie. Dicha funcionalidad se vuelve extremadamente útil en caso de acceso físico limitado o nulo al módulo.

OTA puede usarse mediante:

- *Arduino IDE*
- *Buscador Web*
- *Servidor HTTP*

La opción IDE de Arduino está destinada principalmente para la fase de desarrollo de software. Las otras dos opciones serían más útiles después de la implementación para proporcionar al módulo actualizaciones de la aplicación manualmente con un navegador web o automáticamente utilizando un servidor http.

En cualquier caso, la primera carga de firmware debe realizarse a través de un puerto serie. Si las rutinas OTA se implementan correctamente en un sketch, entonces todas las subidas siguientes se pueden realizar por el aire.

No hay seguridad impuesta en el proceso OTA de ser hackeado. Es responsabilidad del desarrollador garantizar que las actualizaciones solo se permitan desde fuentes legítimas/confiables. Una vez que se completa la actualización, el módulo se reinicia y se ejecuta el nuevo código. El desarrollador debe asegurarse de que la aplicación que se ejecuta en el módulo se cierre y reinicie de manera segura. Los capítulos a continuación proporcionan información adicional sobre la seguridad y protección del proceso OTA.

#### 6.1.1 Seguridad

El módulo debe exponerse de forma inalámbrica para actualizarse con un nuevo sketch. Eso plantea posibilidades de que el módulo sea hackeado y se cargue otro código. Para reducir la posibilidad de ser hackeado, considere proteger sus subidas con una contraseña, seleccionando cierto puerto OTA, etc.

Compruebe las funcionalidades proporcionadas con la librería [ArduinoOTA](#) para mejorar la seguridad:

```
void setPort(uint16_t port);  
void setHostname(const char* hostname);  
void setPassword(const char* password);
```

Cierta funcionalidad de protección ya está incorporada y no requiere ninguna codificación adicional por parte del desarrollador. `ArduinoOTA` y `espota.py` utiliza `Digest-MD5` para autenticar la subida. La integridad de los datos transferidos se verifica en el lado ESP utilizando un checksum `MD5`.

Haga su propio análisis de riesgos y dependiendo de la aplicación, decida qué funciones de la librería implementa. Si es necesario, considere la implementación de otros medios de protección contra la piratería, por ejemplo: exponer el módulo para cargar solo según el programa específico, desencadenar OTA solo si el usuario presiona el botón dedicado «Actualizar» conectado al ESP, etc.

## 6.1.2 Protección

El proceso OTA toma los recursos y el ancho de banda del ESP durante la carga. Luego se reinicia el módulo y se ejecuta un nuevo sketch. Analiza y prueba cómo afecta la funcionalidad del sketch existente y nuevo.

Si el ESP se coloca en una ubicación remota y controla algún equipo, debe poner atención adicional sobre lo que sucede si el proceso de actualización interrumpe repentinamente la operación de este equipo. Por lo tanto, decida cómo poner este equipo en un estado seguro antes de iniciar la actualización. Por ejemplo, su módulo puede estar controlando el sistema de riego de un jardín en una secuencia. Si esta secuencia no se cierra correctamente y se deja abierta una válvula de agua, su jardín puede quedar inundado.

Las siguientes funciones son proporcionadas en la librería `ArduinoOTA`, destinadas a manejar la funcionalidad de su aplicación durante etapas específicas de OTA o ante un error de OTA:

```
void onStart(OTA_CALLBACK(fn));  
void onEnd(OTA_CALLBACK(fn));  
void onProgress(OTA_CALLBACK_PROGRESS(fn));  
void onError(OTA_CALLBACK_ERROR(fn));
```

## 6.1.3 Requerimientos Básicos

El tamaño del chip flash debe poder contener el boceto anterior (actualmente en ejecución) y el nuevo boceto (OTA) al mismo tiempo.

Tenga en cuenta que el sistema de archivos y la EEPROM, por ejemplo, también necesitan espacio (una vez), consulte el Esquema de la memoria flash.

```
ESP.getFreeSketchSpace();
```

puede usarse para comprobar el espacio libre para el nuevo sketch.

Para obtener una descripción general del diseño de la memoria, dónde se almacena el nuevo boceto y cómo se copia durante el proceso OTA, consulte *Proceso de actualización - Vista de la memoria*.

Los siguientes capítulos proporcionan más detalles y métodos específicos para hacer OTA..

## 6.2 Arduino IDE

La carga inalámbrica de módulos desde Arduino IDE está diseñada para los siguientes escenarios típicos:

- durante el desarrollo del firmware como una alternativa más rápida a la carga en serie.

- para actualizar una pequeña cantidad de módulos.
- solo si los módulos están disponibles en la misma red que la computadora con Arduino IDE.

## 6.2.1 Requerimientos

- El ESP y el ordenador deben estar conectados a la misma red.

## 6.2.2 Ejemplo de Aplicación

Las siguientes instrucciones muestran la configuración de OTA en la placa NodeMCU 1.0 (módulo ESP-12E). Puedes usar cualquier otra placa asumiendo que cumple *Requerimientos Básicos* descritos anteriormente. Esta instrucción es válida para todos los sistemas operativos compatibles con Arduino IDE. Se han realizado capturas de pantalla en Windows 7 y es posible que vea pequeñas diferencias (como el nombre del puerto serie), si está usando Linux y MacOS.

1. Antes de continuar, asegúrese de tener el siguiente software instalado:

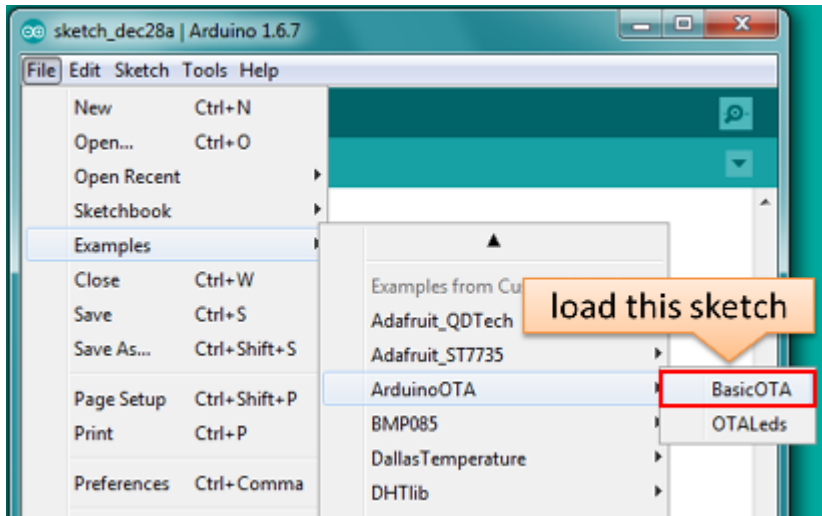
- Arduino IDE 1.6.7 o posterior - <https://www.arduino.cc/en/Main/Software>
- Paquete de la plataforma esp8266/Arduino 2.0.0 o posterior - para instrucciones vaya a: <https://github.com/Irmoreno007/ESP8266-Arduino-Spanish#instalando-mediante-el-gestor-de-tarjetas>
- Python 2.7 - <https://www.python.org/>

**Nota:** Los usuarios de Windows deben seleccionar «Agregar python.exe a la ruta» (ver más abajo, esta opción no está seleccionada de manera predeterminada).

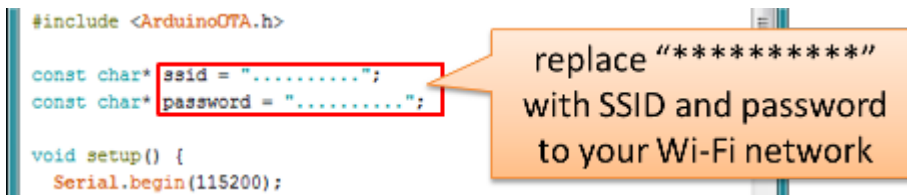


2. Ahora prepare el sketch y la configuración para la primera subida a través del puerto serie.

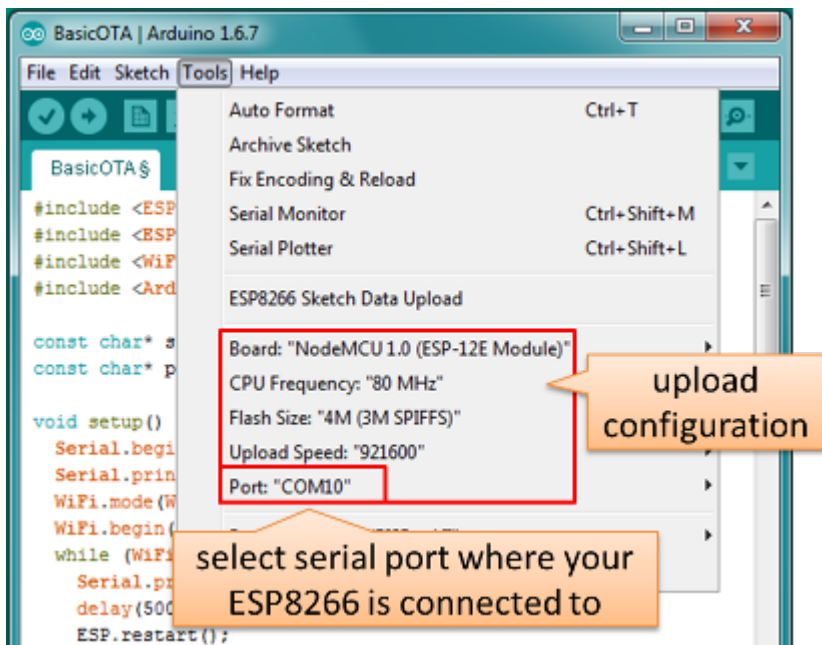
- Inicie Arduino IDE y cargue el sketch BasicOTA.ino disponible en Archivo > Ejemplos > ArduinoOTA



- Actualice el SSID y la contraseña en el sketch, para que el módulo pueda unirse a su red Wi-Fi



- Configure los parámetros de carga como se muestra a continuación (es posible que deba ajustar la configuración si está utilizando un módulo diferente):



**Nota:** Dependiendo de la versión del paquete de plataforma y la placa que tenga, puede ver `Upload using:` en el menú de arriba. Esta opción está inactiva y no importa lo que seleccione. Se dejó para compatibilidad con la implementación anterior de OTA y finalmente se eliminó en la versión 2.2.0 del paquete de plataforma.

3. Suba el sketch (Ctrl+U). Una vez hecho, abra el Monitor Serie (Ctrl+Shift+M) y compruebe si el módulo se ha



unido a su red Wi-Fi:

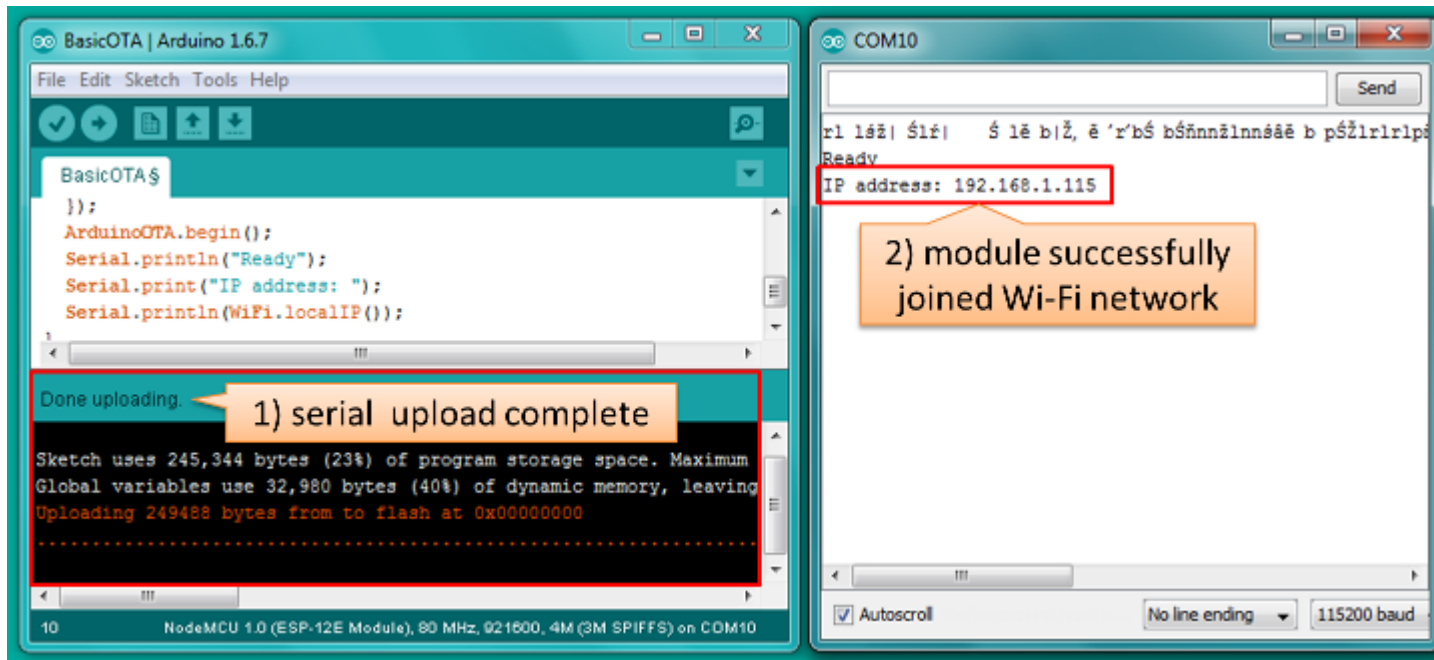


Figura 1: **Nota:** El módulo ESP debe reiniciarse después de la carga por el puerto serie. De lo contrario, los siguientes pasos no funcionarán. El reinicio se puede hacer automáticamente después de abrir el monitor serie como se muestra en la captura de pantalla anterior. Depende de cómo tengas conectado DTR y RTS desde el convertidor USB-serie al ESP. Si el restablecimiento no se realiza automáticamente, hágalo presionando el botón de reset o reiniciando manualmente la alimentación. Para obtener más información sobre por qué debería hacerse esto, consulte [Preguntas frecuentes](#) con respecto a `ESP.restart()`.

4. Solo si el módulo está conectado a la red, después de un par de segundos, el puerto `esp8266-ota` aparecerá en el IDE de Arduino. Seleccione el puerto con la dirección IP que se muestra en la ventana del Monitor serie en el paso anterior:

**Nota:** Si el puerto OTA no se muestra, salga del IDE de Arduino, ábralo nuevamente y verifique si el puerto está allí. Si no funciona, verifique la configuración de su firewall y del router. El puerto OTA se anuncia mediante el servicio mDNS. Para verificar si su PC puede ver el puerto, puede usar una aplicación como Bonjour Browser.

5. Ahora prepárate para tu primera carga OTA seleccionando el puerto OTA:

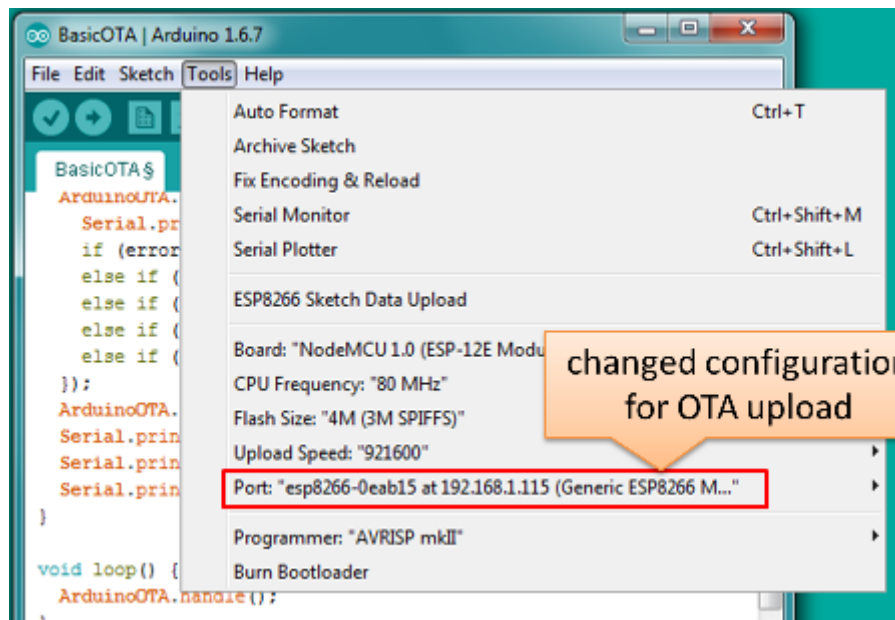
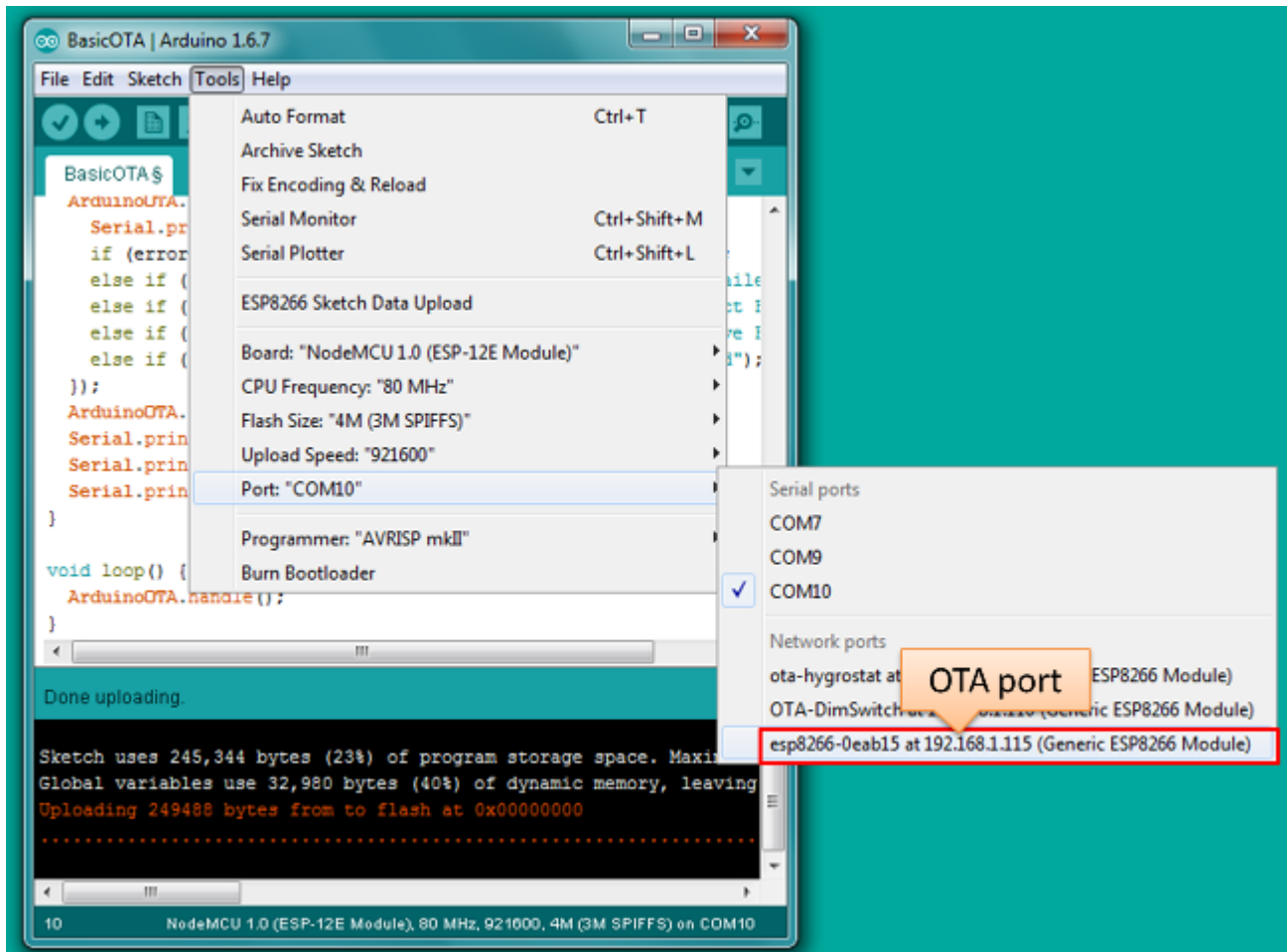
**Nota:** La entrada del menú `Upload Speed`: no importa en este punto ya que se refiere al puerto serie. Deje lo sin cambiar.

6. Si ha completado con éxito todos los pasos anteriores, puede cargar (`Ctrl+U`) el mismo (o cualquier otro) sketch sobre OTA:

**Nota:** Para poder cargar su sketch una y otra vez utilizando OTA, debe insertar rutinas OTA en su interior. Por favor use `BasicOTA.ino` como ejemplo.

## Protección con contraseña

Proteger sus cargas OTA con contraseña es realmente sencillo. Todo lo que necesita hacer es incluir la siguiente declaración en su código:





```
ArduinoOTA.setPassword((const char *) "123");
```

Donde 123 es una contraseña de ejemplo que debe reemplazar con la suya.

Antes de implementarlo en su sketch, es una buena idea verificar cómo funciona el sketch *BasicOTA.ino* disponible en *Archivo > Ejemplos > ArduinoOTA*. Adelante, abra *BasicOTA.ino*, descomente la declaración anterior y cargue el sketch. Para facilitar la resolución de problemas, no modifique el boceto de ejemplo, solo lo absolutamente necesario. Se incluye una contraseña OTA 123 simple y original. A continuación, intente cargar el sketch de nuevo (utilizando OTA). Una vez finalizada la compilación, una vez que la carga está a punto de comenzar, debería ver la solicitud de contraseña de la siguiente manera:

Ingrese la contraseña y la carga debe iniciarse como de costumbre, con la única diferencia del mensaje Autenticando ... OK visible en el registro de subida.

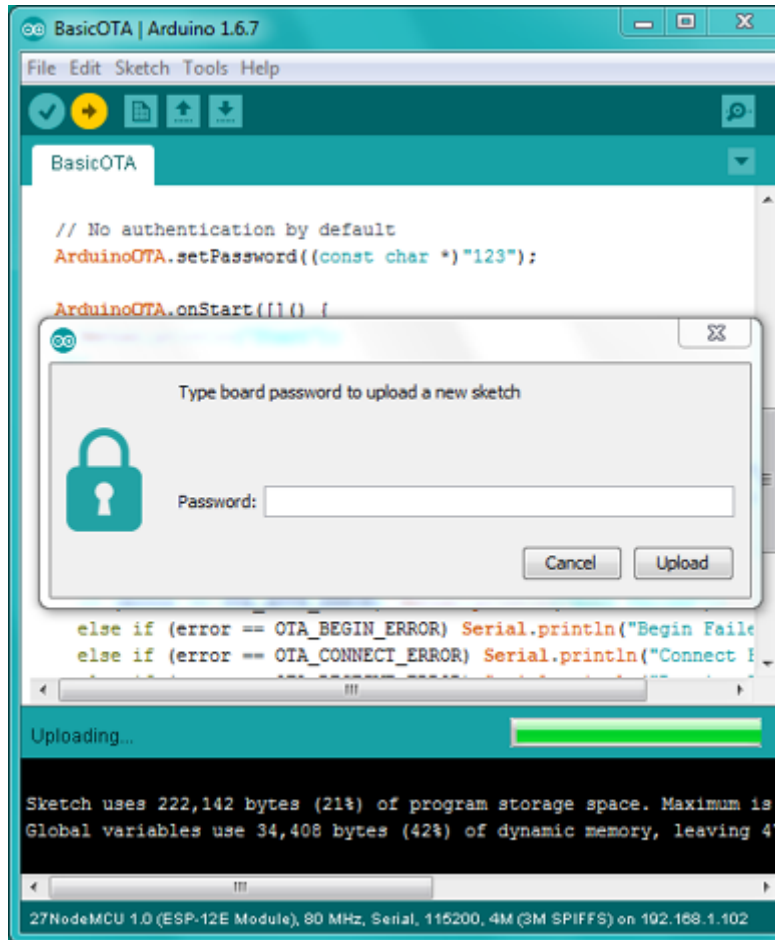
No se le solicitará que vuelva a ingresar la misma contraseña la próxima vez. Arduino IDE lo recordará por ti. Verá una solicitud de contraseña solo después de volver a abrir el IDE o si la cambia en su sketch, cargue el sketch y luego intente cargarlo nuevamente.

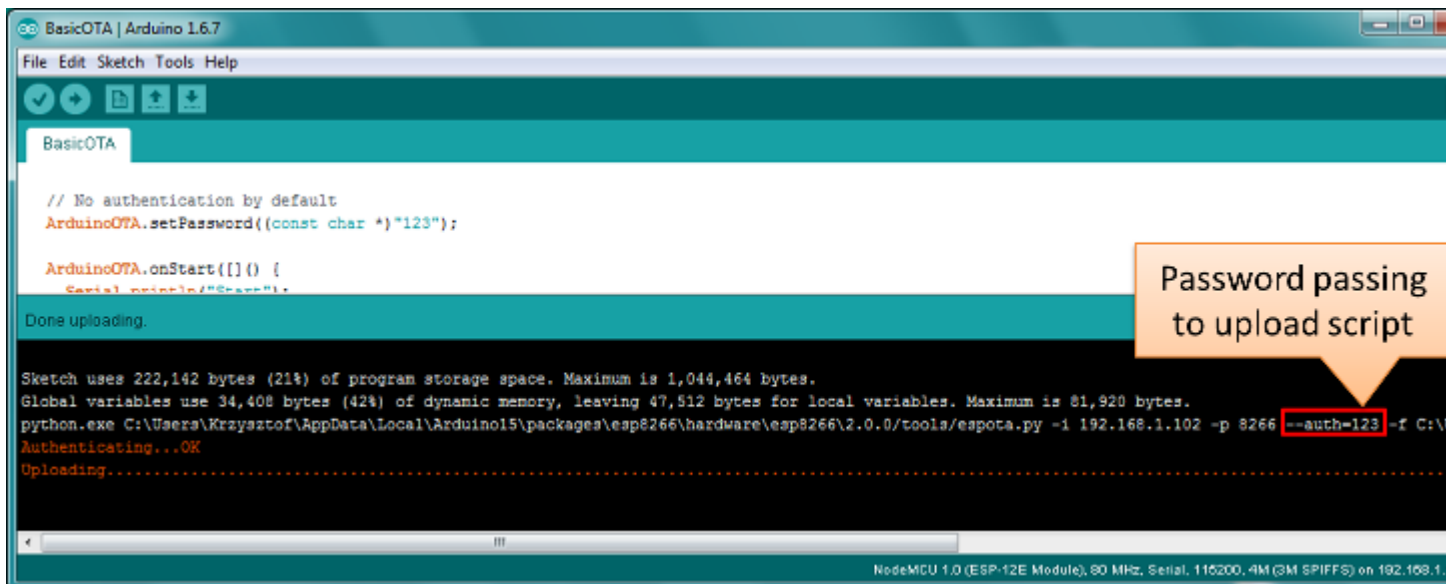
Tenga en cuenta que es posible revelar la contraseña ingresada previamente en el IDE de Arduino, si el IDE no se ha cerrado desde la última carga. Esto se puede hacer habilitando *Mostrar salida detallada mientras: Subir* en *Archivo > Preferencias* y intentando subir el modulo.

La imagen de arriba muestra que la contraseña es visible en el registro, ya que se pasa al script de subida *espot.py*.

Otro ejemplo a continuación muestra la situación cuando la contraseña se cambia entre subidas.

En la imagen puede verse que al subir, el IDE de Arduino utilizó la contraseña ingresada previamente, por lo que la carga falló y eso fue claramente reportado por el IDE. Solo entonces el IDE solicitó una nueva contraseña. Se ingresó correctamente y el segundo intento de carga fue exitoso.





BasicOTA | Arduino 1.6.7

File Edit Sketch Tools Help

BasicOTA

```
// No authentication by default
ArduinoOTA.setPassword((const char *)"123");

ArduinoOTA.onStart([]() {
  Serial.println("Start");
});
```

Done uploading.

Sketch uses 222,142 bytes (21%) of program storage space. Maximum is 1,044,464 bytes.  
Global variables use 34,408 bytes (42%) of dynamic memory, leaving 47,512 bytes for local variables. Maximum is 81,920 bytes.

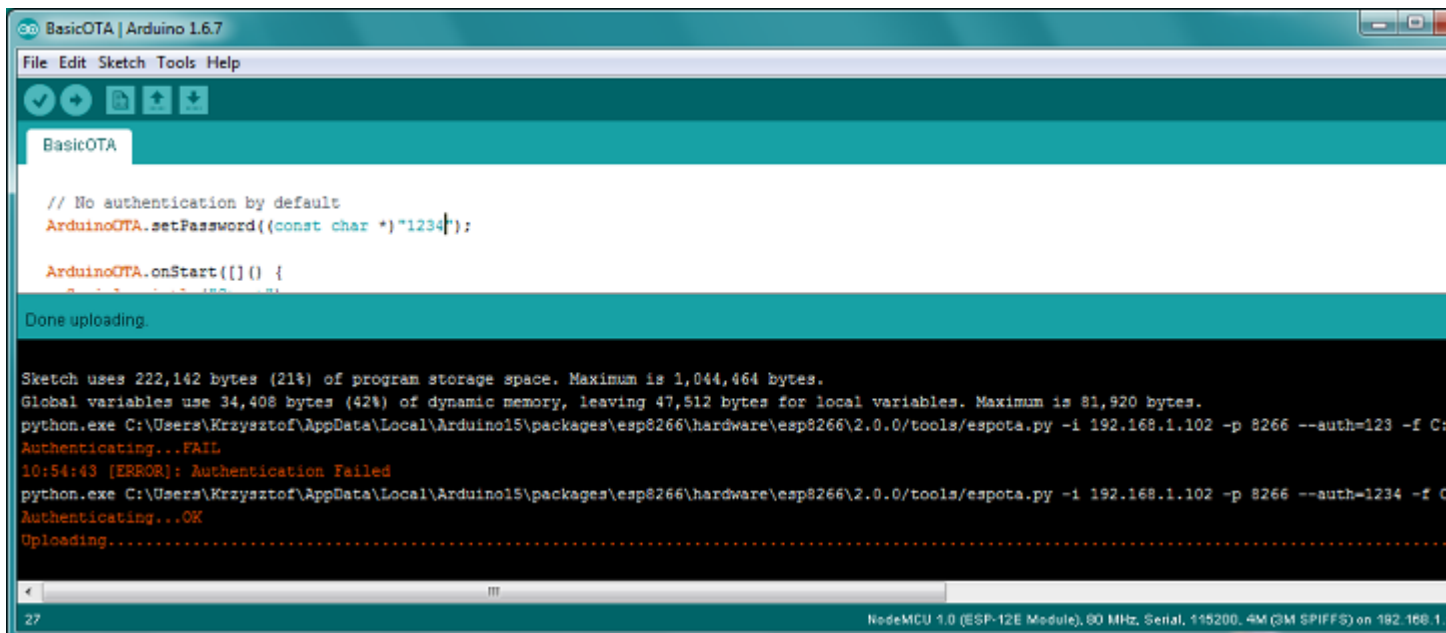
python.exe C:\Users\Krzysztof\AppData\Local\Arduino15\packages\esp8266\hardware\esp8266\2.0.0/tools/espota.py -i 192.168.1.102 -p 8266 --auth=123 -f C:\Users\Krzysztof\AppData\Local\Arduino15\packages\esp8266\hardware\esp8266\2.0.0/bin/espota.py

Authenticating...OK

Uploading.....

NodeMCU 1.0 (ESP-12E Module), 80 MHz, Serial, 115200, 4M (3M SPIFFS) on 192.168.1.102

Password passing  
to upload script



BasicOTA | Arduino 1.6.7

File Edit Sketch Tools Help

BasicOTA

```
// No authentication by default
ArduinoOTA.setPassword((const char *)"123");

ArduinoOTA.onStart([]() {
  Serial.println("Start");
});
```

Done uploading.

Sketch uses 222,142 bytes (21%) of program storage space. Maximum is 1,044,464 bytes.  
Global variables use 34,408 bytes (42%) of dynamic memory, leaving 47,512 bytes for local variables. Maximum is 81,920 bytes.

python.exe C:\Users\Krzysztof\AppData\Local\Arduino15\packages\esp8266\hardware\esp8266\2.0.0/tools/espota.py -i 192.168.1.102 -p 8266 --auth=123 -f C:\Users\Krzysztof\AppData\Local\Arduino15\packages\esp8266\hardware\esp8266\2.0.0/bin/espota.py

Authenticating...FAIL

10:54:43 [ERROR]: Authentication Failed

python.exe C:\Users\Krzysztof\AppData\Local\Arduino15\packages\esp8266\hardware\esp8266\2.0.0/tools/espota.py -i 192.168.1.102 -p 8266 --auth=1234 -f C:\Users\Krzysztof\AppData\Local\Arduino15\packages\esp8266\hardware\esp8266\2.0.0/bin/espota.py

Authenticating...OK

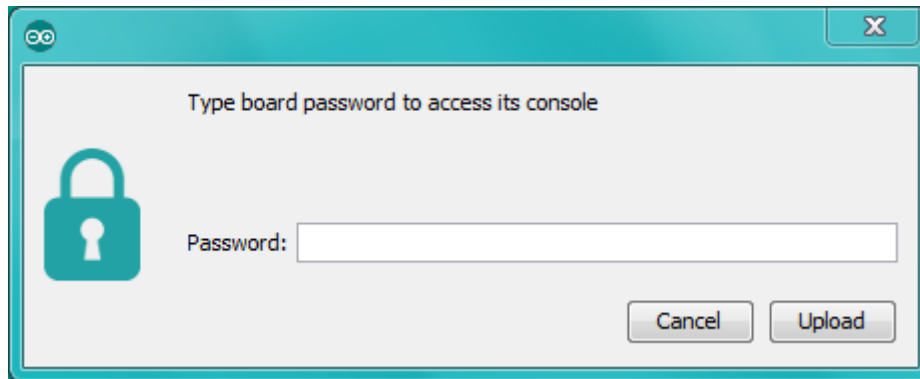
Uploading.....

27

NodeMCU 1.0 (ESP-12E Module), 80 MHz, Serial, 115200, 4M (3M SPIFFS) on 192.168.1.102

## Solución de problemas

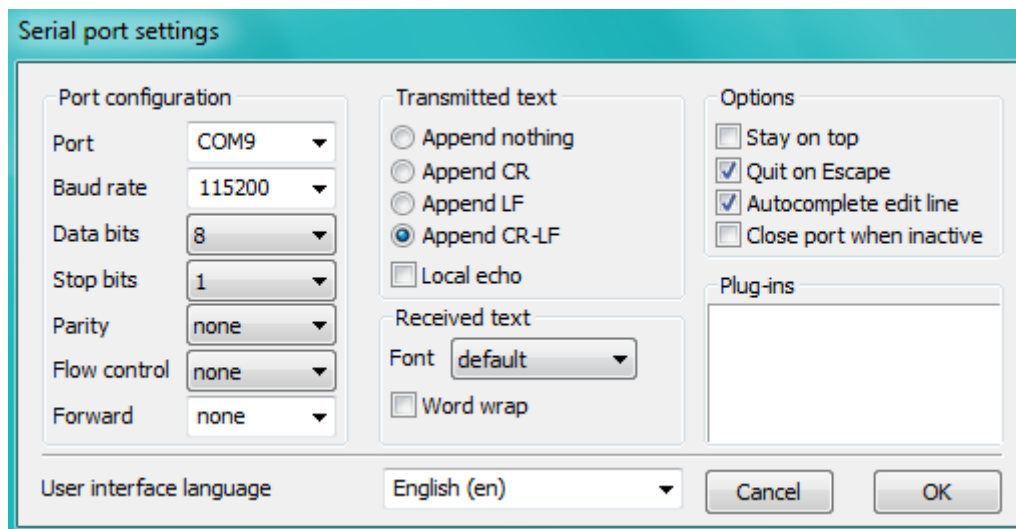
Si la actualización de OTA falla, el primer paso es verificar los mensajes de error que pueden aparecer en la ventana de carga del IDE de Arduino. Si esto no proporciona sugerencias útiles, intente cargar de nuevo mientras verifica lo que muestra ESP en el puerto serie. Serial Monitor de IDE no será útil en ese caso. Al intentar abrirlo, es probable que veas lo siguiente:



Esta ventana es para Arduino Yún y aún no está implementada para esp8266/Arduino. Aparece porque el IDE está intentando abrir Serial Monitor utilizando el puerto de red que ha seleccionado para la carga OTA.

En su lugar, necesita un monitor serie externo. Si es un usuario de Windows, consulte [Termite](#). Este es un terminal RS232 práctico, elegante y simple que no impone el control de flujo RTS o DTR. Dicho control de flujo puede causar problemas si está utilizando líneas respectivas para alternar los pines GPIO0 y RESET en ESP para la carga.

Seleccione el puerto COM y la velocidad en baudios en el programa terminal externo como si estuviera usando Arduino Serial Monitor. Consulte la configuración típica de [Termite](#) a continuación:

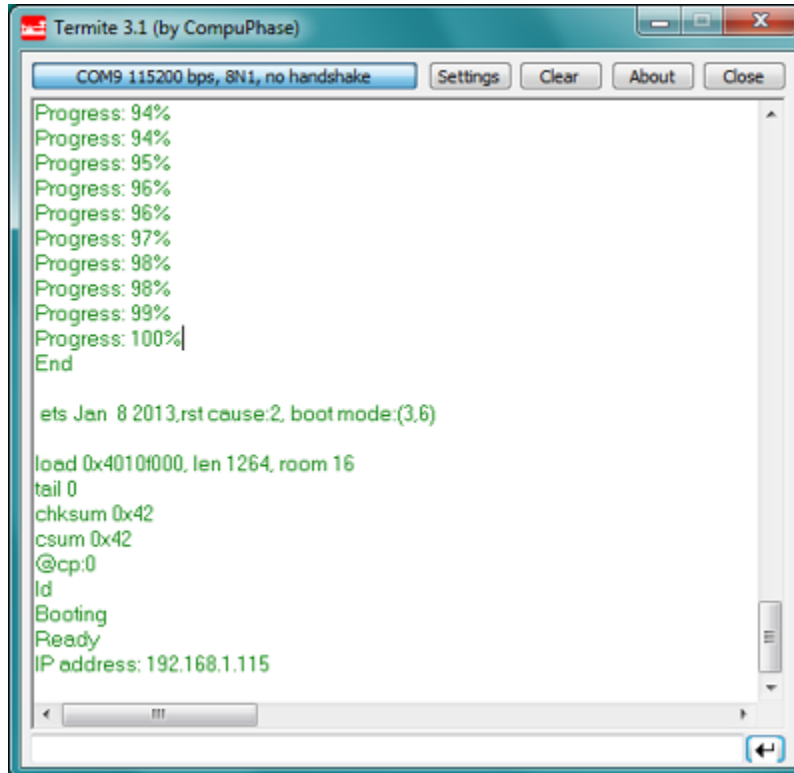


Luego ejecute OTA desde el IDE y observe lo que se muestra en el terminal. El proceso *ArduinoOTA* exitoso usando el sketch *BasicOTA.ino* se ve a continuación (la dirección IP depende de la configuración de su red):

Si la carga falla, es probable que vea los errores detectados por el cargador, la excepción y el seguimiento de la pila, o ambos.

En lugar del registro como en la pantalla anterior, puede ver lo siguiente:

Si este es el caso, lo más probable es que el módulo ESP no se haya reiniciado después de la primera subida utilizando el puerto serie.



Termite 3.1 (by CompuPhase)

COM9 115200 bps, 8N1, no handshake

Settings Clear About Close

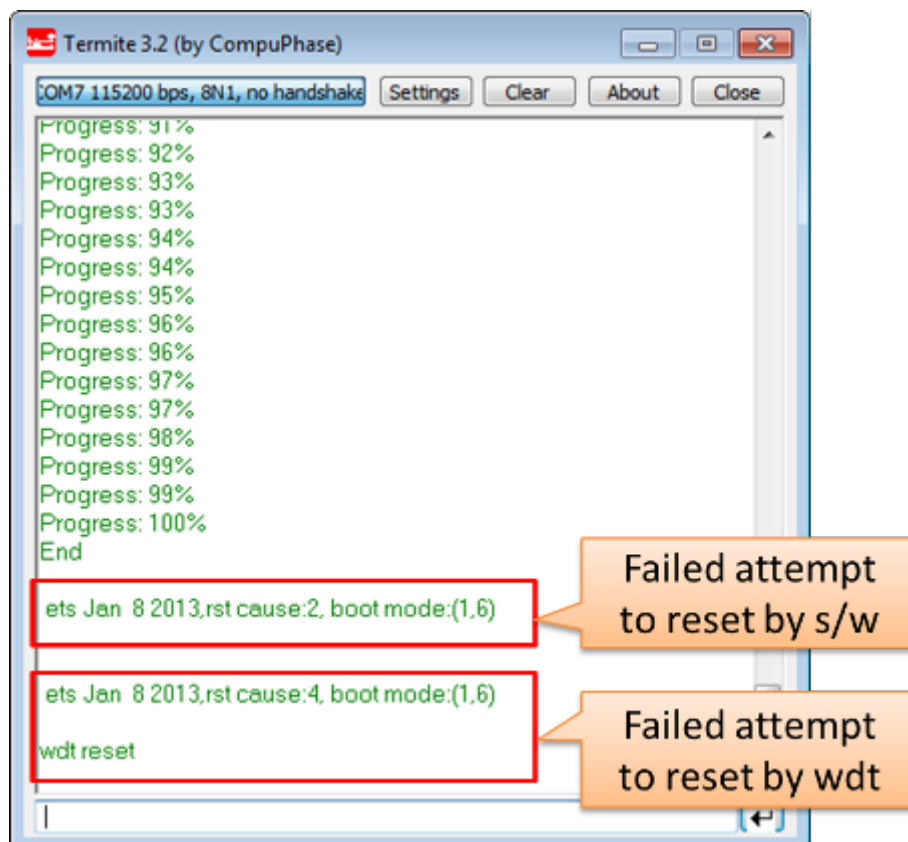
```

Progress: 94%
Progress: 94%
Progress: 95%
Progress: 96%
Progress: 96%
Progress: 97%
Progress: 98%
Progress: 98%
Progress: 99%
Progress: 100%
End

ets Jan 8 2013,rst cause:2, boot mode:(3,6)

load 0x4010f000, len 1264, room 16
tail 0
chksum 0x42
csum 0x42
@cp:0
ld
Booting
Ready
IP address: 192.168.1.115

```



Termite 3.2 (by CompuPhase)

COM7 115200 bps, 8N1, no handshake

Settings Clear About Close

```

Progress: 91%
Progress: 92%
Progress: 93%
Progress: 93%
Progress: 94%
Progress: 94%
Progress: 95%
Progress: 96%
Progress: 96%
Progress: 97%
Progress: 97%
Progress: 98%
Progress: 99%
Progress: 99%
Progress: 100%
End

ets Jan 8 2013,rst cause:2, boot mode:(1,6)

ets Jan 8 2013,rst cause:4, boot mode:(1,6)

wdt reset

```

Failed attempt to reset by s/w

Failed attempt to reset by wdt



Las causas más comunes de fallo OTA son las siguientes:

- no hay suficiente memoria física en el chip (por ejemplo, ESP01 con 512K de memoria flash no es suficiente para OTA).
- demasiada memoria declarada para SPIFFS, por lo que el nuevo sketch no se ajustará entre el boceto existente y SPIFFS - vea *Proceso de actualización - vista de la memoria*.
- muy poca memoria declarada en Arduino IDE para su placa seleccionada (es decir, menor que el tamaño físico).
- no reiniciar el módulo ESP después de la primera subida utilizando el puerto serie.

Para obtener más información sobre el diseño de la memoria flash, consulte *Sistema de ficheros*. Para obtener información general sobre dónde se almacena el nuevo boceto, cómo se copia y cómo se organiza la memoria para el propósito de OTA, consulte *Proceso de actualización - vista de la memoria*.

## 6.3 Buscador Web

Las actualizaciones descritas en este capítulo se realizan con un navegador web que puede ser útil en los siguientes escenarios típicos:

- después de la implementación de la aplicación si la carga directa desde Arduino IDE es inconveniente o no es posible.
- después de la implementación si el usuario no puede exponer el módulo para OTA desde un servidor de actualización externo.
- para proporcionar actualizaciones después de la implementación a una pequeña cantidad de módulos al configurar un servidor de actualización no es factible.

### 6.3.1 Requerimientos

- El ESP y el ordenador deben estar conectados a la misma red.

### 6.3.2 Descripción general de la implementación

Las actualizaciones con un navegador web se implementan utilizando la clase `ESP8266HTTPUpdateServer` junto con las clases `ESP8266WebServer` y `ESP8266mDNS`. Se requiere el siguiente código para que funcione:

`setup()`

```
MDNS.begin(host);

httpUpdater.setup(&httpServer);
httpServer.begin();

MDNS.addService("http", "tcp", 80);
```

`loop()`

```
httpServer.handleClient();
```



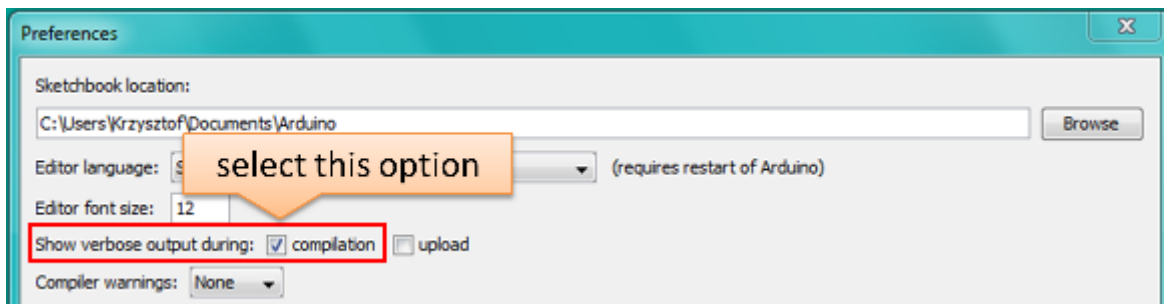
### 6.3.3 Ejemplo de aplicación

La implementación de ejemplo proporcionada a continuación se ha realizado utilizando:

- Sketch de ejemplo WebUpdater.ino disponible en la librería ESP8266HTTPUpdateServer library.
- NodeMCU 1.0 (Módulo ESP-12E).

Puede utilizar otro módulo si cumple con los requisitos descritos anteriormente. *Requerimientos Básicos.*

1. Before you begin, please make sure that you have the following software installed:
  - Arduino IDE y la versión 2.0.0-rc1 (del 17 de noviembre de 2015) de la plataforma del paquete como se describe en <https://github.com/lrmoreno007/ESP8266-Arduino-Spanish#instalando-mediante-el-gestor-de-tarjetas>
  - Software de host en función del sistema operativo que utilice:
    - a) Avahi <http://avahi.org/> para Linux
    - b) Bonjour <http://www.apple.com/support/bonjour/> para Windows
    - c) Mac OSX y iOS - ya soportado interiormente, no se requiere ningún software extra
2. Prepare el boceto y la configuración para la primera subida mediante puerto serie.
  - Inicie Arduino IDE y cargue el sketch WebUpdater.ino disponible en Archivo > Ejemplos > ESP8266HTTPUpdateServer.
  - Actualice su SSID y contraseña en el sketch, para que el módulo pueda unirse a su red Wi-Fi.
  - Abra Archivo > Preferencias, busque “Mostrar salida detallada mientras:” y active la opción “Compilación”.



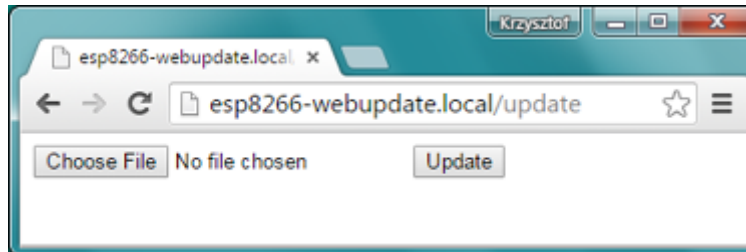
**Nota:** Esta configuración será necesaria en el paso 5 a continuación. Puedes desmarcar esta configuración después.

3. Suba el sketch (Ctrl+U). Una vez hecho esto, abra el Monitor Serie (Ctrl+Shift+M) y verifique si aparece el siguiente mensaje, que contiene la url para la actualización OTA.



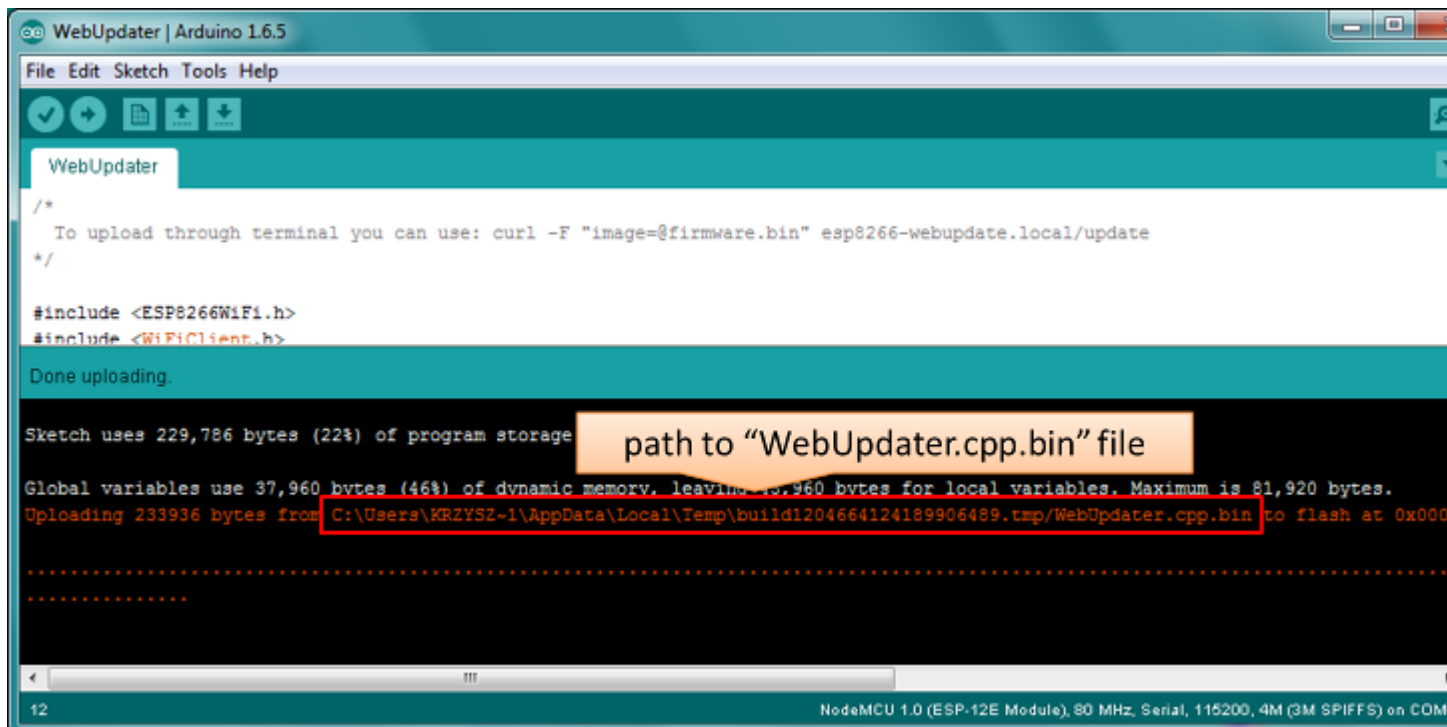
**Nota:** Dicho mensaje se mostrará solo después de que el módulo se una con éxito a la red y esté listo para una carga OTA. Recuerde lo hablado acerca de reiniciar el módulo después de la primera subida mediante serial como se explica en el capítulo *Arduino IDE*, paso 3.

- Ahora abra el navegador web e ingrese la url proporcionada por el Monitor Serie, es decir, `http://esp8266-webupdate.local/update`. Una vez ingresado, el navegador debe mostrar un formulario como el que se encuentra en su módulo. El formulario te invita a elegir un archivo para actualizar.

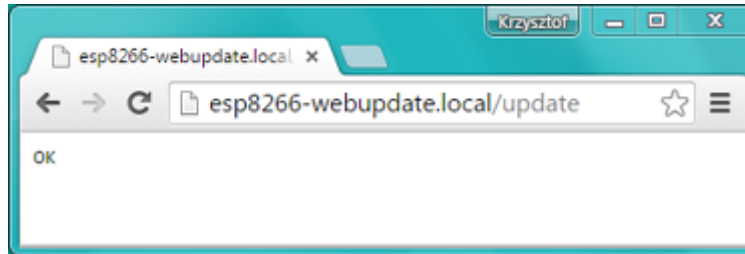


**Nota:** Si mediante `http://esp8266-webupdate.local/update` no funciona, intente reemplazar `esp8266-webupdate` con la dirección IP del módulo. Por ejemplo, si la IP de su módulo es `192.168.1.100`, entonces la url debería ser `http://192.168.1.100/update`. Esta solución es útil en caso de que el software host instalado en el paso 1 no funcione. Si todavía nada funciona y no hay pistas en el Monitor Serie, intente diagnosticar el problema abriendo la URL proporcionada en Google Chrome, presionando F12 y verificando el contenido de las pestañas «Consola» y «Red». Chrome proporciona un registro avanzado en estas pestañas.

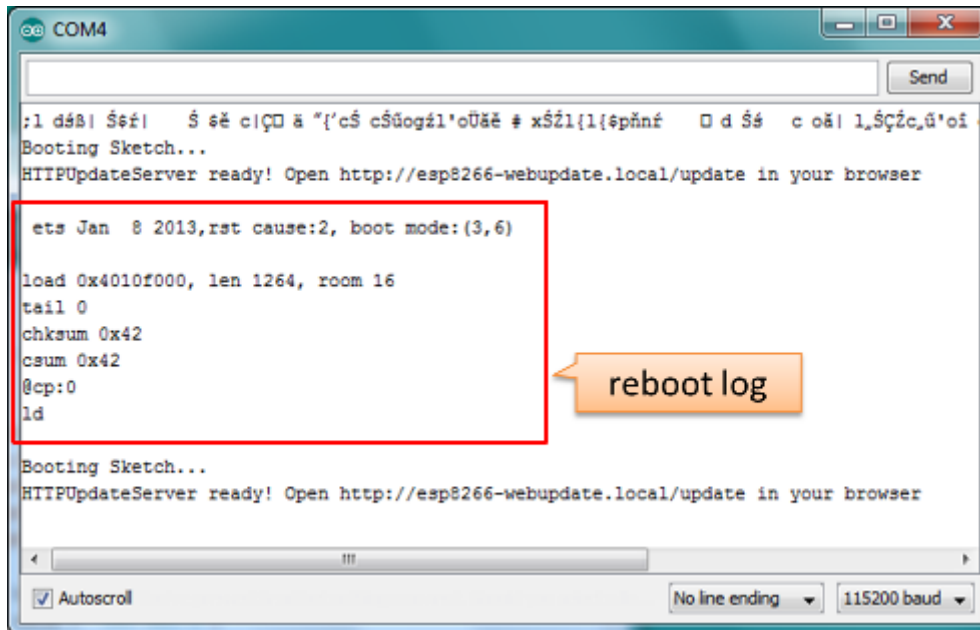
- Para obtener el archivo, navegue al directorio utilizado por Arduino IDE para almacenar los resultados de la compilación. Puede verificar la ruta de acceso a este archivo en el registro de compilación que se muestra en la ventana de depuración del IDE como se indica a continuación.



- Ahora presione “Choose File” en el navegador web, vaya al directorio identificado en el paso 5 anterior, busque el archivo «WebUpdater.cpp.bin» y cárguelo. Si la carga se realiza correctamente, verá «OK» en el navegador web como se muestra a continuación.



Se reiniciará el módulo que debería estar visible en el Monitor Serie:



Justo después de reiniciar, debería ver exactamente el mismo mensaje HTTPUpdateServer ready! Open `http://esp8266-webupdate.local/update` in your browser como en el paso 3. Esto se debe a que el módulo se ha cargado nuevamente con el mismo código: primero utilizando el puerto serie y luego usando OTA.

Una vez que se sienta cómodo con este procedimiento, siga adelante y modifique el boceto de `WebUpdater.ino` para imprimir algunos mensajes adicionales, compile, localice un nuevo archivo binario y cárguelo utilizando el navegador web para ver los cambios introducidos en el Monitor Serie.

También puede agregar rutinas OTA a su propio sketch siguiendo las pautas en *Descripción general de la implementación*. Si esto se hace correctamente, siempre debe poder cargar un nuevo boceto sobre el anterior utilizando un navegador web.

En caso de que la actualización de OTA falle después de introducir modificaciones en su boceto, siempre puede recuperar el módulo cargándolo en un puerto serie. Luego, diagnostique el problema con el boceto utilizando el Monitor Serie. Una vez que se solucione el problema intente OTA otra vez.

## 6.4 Servidor HTTP

La clase `ESPhttpUpdate` puede buscar actualizaciones y descargar un archivo binario desde el servidor web HTTP. Es posible descargar actualizaciones de cada dirección IP o de dominio en la red o en Internet.

## 6.4.1 Requerimientos

- Servidor Web

## 6.4.2 Código Arduino

### Actualizador Sencillo

El actualizador sencillo descarga el archivo cada vez que se llama a la función.

```
ESPhttpUpdate.update("192.168.0.2", 80, "/arduino.bin");
```

### Actualizador Avanzado

Es posible apuntar la función de actualización a un script en el servidor. Si se proporciona un argumento de cadena de versión, se enviará al servidor. El script del lado del servidor puede usar esto para verificar si se debe realizar una actualización.

El script del lado del servidor puede responder de la siguiente manera:

- código de respuesta 200, y enviar la imagen del firmware
- o código de respuesta 304 para notificar a ESP que no se requiere ninguna actualización.

```
t_httpUpdate_return ret = ESPhttpUpdate.update("192.168.0.2", 80, "/esp/update/  
↪arduino.php", "optional current version string here");  
switch (ret) {  
  case HTTP_UPDATE_FAILED:  
    Serial.println("[update] Update failed.");  
    break;  
  case HTTP_UPDATE_NO_UPDATES:  
    Serial.println("[update] Update no Update.");  
    break;  
  case HTTP_UPDATE_OK:  
    Serial.println("[update] Update ok."); // may not called we reboot the ESP  
    break;  
}
```

## 6.4.3 Manejo de solicitudes del servidor

### Actualizador Sencillo

Para el actualizador sencillo, el servidor solo necesita entregar el archivo binario para su actualización.

### Advanced updater

Para la administración avanzada de actualizaciones, un script debe ejecutarse en el lado del servidor, por ejemplo, un script PHP. En cada solicitud de actualización, el ESP envía alguna información en los encabezados HTTP al servidor.

Ejemplo de datos de encabezado:

```
[HTTP_USER_AGENT] => ESP8266-http-Update
[HTTP_X_ESP8266_STA_MAC] => 18:FE:AA:AA:AA:AA
[HTTP_X_ESP8266_AP_MAC] => 1A:FE:AA:AA:AA:AA
[HTTP_X_ESP8266_FREE_SPACE] => 671744
[HTTP_X_ESP8266_SKETCH_SIZE] => 373940
[HTTP_X_ESP8266_SKETCH_MD5] => a56f8ef78a0bebd812f62067daf1408a
[HTTP_X_ESP8266_CHIP_SIZE] => 4194304
[HTTP_X_ESP8266_SDK_VERSION] => 1.3.0
[HTTP_X_ESP8266_VERSION] => DOOR-7-g14f53a19
```

Con esta información, el script ahora puede verificar si se necesita una actualización. También es posible entregar diferentes binarios basados en la dirección MAC, por ejemplo.

Ejemplo de guión:

```
<?PHP

header('Content-type: text/plain; charset=utf8', true);

function check_header($name, $value = false) {
    if(!isset($_SERVER[$name])) {
        return false;
    }
    if($value && $_SERVER[$name] != $value) {
        return false;
    }
    return true;
}

function sendFile($path) {
    header($_SERVER["SERVER_PROTOCOL"].' 200 OK', true, 200);
    header('Content-Type: application/octet-stream', true);
    header('Content-Disposition: attachment; filename='.basename($path));
    header('Content-Length: '.filesize($path), true);
    header('x-MD5: '.md5_file($path), true);
    readfile($path);
}

if(!check_header('HTTP_USER_AGENT', 'ESP8266-http-Update')) {
    header($_SERVER["SERVER_PROTOCOL"].' 403 Forbidden', true, 403);
    echo "only for ESP8266 updater!\n";
    exit();
}

if(
    !check_header('HTTP_X_ESP8266_STA_MAC') ||
    !check_header('HTTP_X_ESP8266_AP_MAC') ||
    !check_header('HTTP_X_ESP8266_FREE_SPACE') ||
    !check_header('HTTP_X_ESP8266_SKETCH_SIZE') ||
    !check_header('HTTP_X_ESP8266_SKETCH_MD5') ||
    !check_header('HTTP_X_ESP8266_CHIP_SIZE') ||
    !check_header('HTTP_X_ESP8266_SDK_VERSION')
) {
    header($_SERVER["SERVER_PROTOCOL"].' 403 Forbidden', true, 403);
    echo "only for ESP8266 updater! (header)\n";
    exit();
}
```

(continues on next page)

(proviene de la página anterior)

```

$db = array(
  "18:FE:AA:AA:AA:AA" => "DOOR-7-g14f53a19",
  "18:FE:AA:AA:AA:BB" => "TEMP-1.0.0"
);

if(!isset($db[$_SERVER['HTTP_X_ESP8266_STA_MAC']])) {
  header($_SERVER["SERVER_PROTOCOL"].' 500 ESP MAC not configured for updates',
  ↪true, 500);
}

$localBinary = "./bin/".$db[$_SERVER['HTTP_X_ESP8266_STA_MAC']].".bin";

// Check if version has been set and does not match, if not, check if
// MD5 hash between local binary and ESP8266 binary do not match if not.
// then no update has been found.
if(!check_header('HTTP_X_ESP8266_SDK_VERSION') && $db[$_SERVER['HTTP_X_ESP8266_STA_
↪MAC']] != $_SERVER['HTTP_X_ESP8266_VERSION'])
  || $_SERVER["HTTP_X_ESP8266_SKETCH_MD5"] != md5_file($localBinary)) {
  sendFile($localBinary);
} else {
  header($_SERVER["SERVER_PROTOCOL"].' 304 Not Modified', true, 304);
}

header($_SERVER["SERVER_PROTOCOL"].' 500 no version for ESP MAC', true, 500);

```

## 6.5 Interfaz de transmisión

POR HACER descripción del Interfaz de Transmisión

El Interfaz de Transmisión es la base para todos los demás modos de actualización como OTA, http Server / client.

## 6.6 Clase Updater

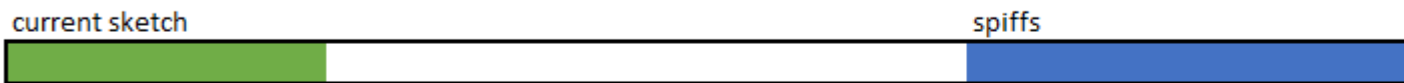
Updater está en el Core y se ocupa de escribir el firmware en la memoria flash, verificar su integridad y decirle al cargador de arranque que cargue el nuevo firmware en el siguiente arranque.

**Nota:** El comando del cargador de arranque se almacenará en los primeros 128 bytes de la memoria RTC del usuario, y luego será recuperado por eboot en el arranque. Eso significa que los datos de usuario presentes allí se perderán (discusión en #5330).

### 6.6.1 Proceso de actualización - Vista de la memoria

- El nuevo sketch se almacenará en el espacio entre el sketch anterior y el spiff.
- En el siguiente reinicio, el gestor de arranque «eboot» verifica los comandos.
- El nuevo sketch ahora se copia «sobre» el anterior.
- Se inicia el nuevo sketch.

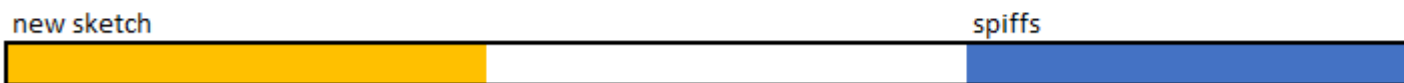
**start:**



**update:**



**reboot:**







---

## Guía para PROGMEM sobre ESP8266 y Arduino IDE

---

### 7.1 Introducción

PROGMEM es una característica Arduino AVR que ha sido portada a ESP8266 para asegurar la compatibilidad con las librerías existentes en Arduino, así como para ahorrar RAM. En ESP8266 al declarar una cadena como `const char * xyz = "this is a string"` colocará esta cadena en la RAM, no en la flash. Es posible colocar una String en la flash, y después cargarla a la RAM cuando sea necesario. En un AVR de 8bit AVR este proceso es muy simple. En el ESP8266 de 32bit hay condiciones que deben cumplirse para leer desde el flash.

En ESP8266, PROGMEM es una macro:

```
#define PROGMEM ICACHE_RODATA_ATTR
```

ICACHE\_RODATA\_ATTR está definida por:

```
#define ICACHE_RODATA_ATTR __attribute__((section(".irom.text")))
```

Coloca la variable en la sección `.irom.text` de la flash. Colocar cadenas en la flash requiere el uso de alguno de los siguientes métodos.

### Declarar una cadena global almacenada en la flash.

```
static const char xyz[] PROGMEM = "Esta es una cadena almacenada en la flash";
```

### 7.2 Declarar una cadena flash en un bloque de código.

Para esto puedes usar la macro `PSTR`. Que está completamente definida en `pgmspace.h`

```
#define PGM_P      const char *
#define PGM_VOID_P const void *
#define PSTR(s)  (__extension__({static const char __c[] PROGMEM = (s); &__c[0];}))
```

En la práctica:

```
void myfunction(void) {
PGM_P xyz = PSTR("Almacena esta cadena en la flash");
const char * abc = PSTR("También almacena esta cadena en la flash");
}
```

Los dos ejemplos anteriores almacenarán cadenas en la flash. Para recuperar y manipular cadenas de flash debe leerse desde la flash en palabras de 4 bytes. En el IDE Arduino para ESP8266 hay varias funciones que pueden ayudar a recuperar cadenas desde la flash que han sido almacenadas utilizando PROGMEM. Ambos ejemplos anteriores devuelven `const char *`. Sin embargo el uso de estos punteros, sin corregir el alineamiento de 32bit provocará un error de segmentación y el ESP8266 se caerá. Debes leer desde la flash de forma alineada a 32-bits.

## 7.3 Funciones para leer desde PROGMEM

Está completamente definido en `pgmspace.h`

```
int memcmp_P(const void* buf1, PGM_VOID_P buf2P, size_t size);
void* memcpy_P(void* dest, PGM_VOID_P src, int c, size_t count);
void* memmem_P(const void* buf, size_t bufSize, PGM_VOID_P findP, size_t findPSize);
void* memcpy_P(void* dest, PGM_VOID_P src, size_t count);
char* strncpy_P(char* dest, PGM_P src, size_t size);
char* strcpy_P(dest, src)
char* strncat_P(char* dest, PGM_P src, size_t size);
char* strcat_P(dest, src)
int strncmp_P(const char* str1, PGM_P str2P, size_t size);
int strcmp_P(str1, str2P)
int strcasecmp_P(const char* str1, PGM_P str2P, size_t size);
int strcasecmp_P(str1, str2P)
size_t strlen_P(PGM_P s, size_t size);
size_t strlen_P(strP)
char* strstr_P(const char* haystack, PGM_P needle);
int printf_P(PGM_P formatP, ...);
int sprintf_P(char *str, PGM_P formatP, ...);
int snprintf_P(char *str, size_t strSize, PGM_P formatP, ...);
int vsnprintf_P(char *str, size_t strSize, PGM_P formatP, va_list ap);
```

Hay muchas funciones pero en realidad son versiones `_P` de las funciones estándar C que han sido adaptadas para leer con el alineamiento de la flash de ESP8266 de 32bit. Todas ellas toman un `PGM_P` que es esencialmente un `const char *`. Debajo de la capa se usan todas estas funciones, un proceso para garantizar que se lean 4 bytes y se devuelva el byte de la solicitud.

Esto funciona bien cuando se ha diseñado una función como la anterior, que está especializada para tratar con los punteros de PROGMEM, pero no hay verificación de tipos, excepto contra `const char *`. Esto significa que es totalmente legítimo, en lo que respecta al compilador, pasar cualquier cadena `const char *`, lo que obviamente no es cierto y conducirá a un comportamiento indefinido. Esto hace que sea imposible crear cualquier función sobrecargada que pueda usar cadenas de flash cuando están definidas como `PGM_P`. Si lo intenta, obtendrá un error de sobrecarga ambiguo como `PGM_P == const char *`.

Introduzca `__FlashStringHelper`. .. Esta es una clase contenedora que permite que las cadenas flash se usen como una clase, esto significa que la verificación de tipos y la sobrecarga de funciones se pueden usar con cadenas flash. La

mayoría de las personas estarán familiarizadas con la macro `F()` y posiblemente con la macro `FPSTR()`. Estos se definen en `WString.h`:

```
#define FPSTR(pstr_pointer) (reinterpret_cast<const __FlashStringHelper *>(pstr_
↪pointer))
#define F(string_literal) (FPSTR(PSTR(string_literal)))
```

Así `FPSTR()` toma un puntero `PROGMEM` a una cadena y lo arroja a esta clase `__FlashStringHelper`. Por lo tanto, si ha definido una cadena como la anterior `xyz`, puede usar `FPSTR()` para convertirla en `__FlashStringHelper` para pasarla a las funciones que la toman.

```
static const char xyz[] PROGMEM = "Esta es una cadena almacenada en la flash";
Serial.println(FPSTR(xyz));
```

El `F()` combina ambos métodos para crear una forma fácil y rápida de almacenar una cadena alineada en flash, y devolver el tipo `__FlashStringHelper`. Por ejemplo:

```
Serial.println(F("Esta es una cadena almacenada en la flash"));
```

Aunque estas dos funciones proporcionan una función similar, cumplen funciones diferentes. `FPSTR()` te permite definir una cadena flash global y luego usarla en cualquier función que tome `__FlashStringHelper`. `F()` le permite definir estas cadenas de flash en un lugar determinado, pero no puede usarlas en ningún otro lado. La consecuencia de esto es que puede compartir cadenas comunes usando `FPSTR()` pero no `F()`. `__FlashStringHelper` es lo que la clase `String` usa para sobrecargar su constructor:

```
String(const char *cstr = ""); // constructor from const char *
String(const String &str); // copy constructor
String(const __FlashStringHelper *str); // constructor for flash strings
```

Esto te permite escribir:

```
String mystring(F("Esta cadena está almacenada en la flash"));
```

¿Cómo escribo una función para usar `__FlashStringHelper`? Sencillo: vuelva a colocar el puntero en un `PGM_P` y use las funciones `_P` que se muestran arriba. Esto es un ejemplo de implementación para `String` para la función `concat`.

```
unsigned char String::concat(const __FlashStringHelper * str) {
    if (!str) return 0; // return if the pointer is void
    int length = strlen_P((PGM_P)str); // Lo echa a PGM_P, que es básicamente const_
↪char *, y mide usando la versión _P de strlen.
    if (length == 0) return 1;
    unsigned int newlen = len + length;
    if (!reserve(newlen)) return 0; // Crea un buffer de la longitud correcta
    strcpy_P(buffer + len, (PGM_P)str); //copia la cadena dentro usando strcpy_P
    len = newlen;
    return 1;
}
```

## 7.4 ¿Como declaro una cadena global flash y la uso?

```
static const char xyz[] PROGMEM = "Esta es una cadena almacenada en la flash.
↪Longitud = %u";

void setup() {
```

(continues on next page)

(proviene de la página anterior)

```

Serial.begin(115200); Serial.println();
Serial.println( FPSTR(xyz) ); // Para imprimir la cadena, debes convertirla a
↪FlashStringHelper primero usando FPSTR().
Serial.printf_P( xyz, strlen_P(xyz)); // Utiliza printf con la cadena PROGMEM
}

```

## 7.5 Como uso cadenas alineadas flash?

```

void setup() {
  Serial.begin(115200); Serial.println();
  Serial.println( F("Esto es una cadena alineada")); //
  Serial.printf_P( PSTR("Esto es una cadena alineada utilizando printf %s"), "hola
↪");
}

```

## 7.6 ¿Como declaro y uso datos en PROGMEM?

```

const size_t len_xyz = 30;
const uint8_t xyz[] PROGMEM = {
  0x53, 0x61, 0x79, 0x20, 0x48, 0x65, 0x6c, 0x6c, 0x6f, 0x20,
  0x74, 0x6f, 0x20, 0x4d, 0x79, 0x20, 0x4c, 0x69, 0x74, 0x74,
  0x6c, 0x65, 0x20, 0x46, 0x72, 0x69, 0x65, 0x6e, 0x64, 0x00};

void setup() {
  Serial.begin(115200); Serial.println();
  uint8_t * buf = new uint8_t[len_xyz];
  if (buf) {
    memcpy_P(buf, xyz, len_xyz);
    Serial.write(buf, len_xyz); // Da salida al buffer.
  }
}

```

## 7.7 ¿Como declaro algunos datos en PROGMEM y recupero un byte de él?

Declara el dato como se hizo anteriormente, entonces utiliza `pgm_read_byte` para coger el valor de vuelta.

```

const size_t len_xyz = 30;
const uint8_t xyz[] PROGMEM = {
  0x53, 0x61, 0x79, 0x20, 0x48, 0x65, 0x6c, 0x6c, 0x6f, 0x20,
  0x74, 0x6f, 0x20, 0x4d, 0x79, 0x20, 0x4c, 0x69, 0x74, 0x74,
  0x6c, 0x65, 0x20, 0x46, 0x72, 0x69, 0x65, 0x6e, 0x64, 0x00
};

void setup() {
  Serial.begin(115200); Serial.println();
  for (int i = 0; i < len_xyz; i++) {
    uint8_t byteval = pgm_read_byte(xyz + i);

```

(continues on next page)

(proviene de la página anterior)

```
Serial.write(byteval); // Da salida al buffer.  
}  
}
```

## 7.8 En resumen

Es fácil almacenar cadenas en la flash usando `PROGMEM` y `PSTR`, pero debe crear funciones que utilicen específicamente los punteros que generan, ya que básicamente son `const char *`. Por otro lado, `FPSTR` y `F()` te ofrecen una clase con la que puede hacer conversiones implícitas, muy útil cuando se sobrecarga funciones y se realizan conversiones de tipo implícitas. Vale la pena agregar que si desea almacenar un `int`, `float` o un puntero, estos pueden ser almacenados y leídos directamente ya que tienen 4 bytes de tamaño y por lo tanto siempre estarán alineados.

Espero que esto ayude.



---

## Utilizando GDB para depurar aplicaciones

---

Las aplicaciones ESP se pueden depurar usando GDB, el depurador de GNU, que se incluye con la instalación estándar de IDE. Esta nota solo discute los pasos específicos de ESP, así que por favor consulte la [documentación principal de GNU GDB](#).

Tenga en cuenta que a partir de 2.5.0, la cadena de herramientas se movió desde el parche ESPRESSIF, versión de código cerrado de GDB a la versión principal de GNU. Los formatos de depuración son diferentes, así que asegúrese de usar solo el último ejecutable GDB del toolchain de Arduino.

### 8.1 Nota CLI e IDE

Debido a que el IDE de Arduino no admite la depuración interactiva, las siguientes secciones describen la depuración utilizando la línea de comandos. Otros IDE que utilizan GDB en sus backends de depuración deberían funcionar de manera idéntica, pero es posible que deba editar sus opciones de archivos de configuración para habilitar la depuración de serie remota requerida y establecer las opciones estándar. ¡Felizmente, se aceptan PRs para actualizar este documento con IDEs adicionales!

### 8.2 Preparando tu aplicación para GDB

Las aplicaciones deben cambiarse para habilitar el soporte de depuración GDB. Este cambio agregará 2-3KB de flash y alrededor de 700 bytes de uso de IRAM, pero no debería afectar el funcionamiento de la aplicación.

En su archivo principal `sketch.ino`, agregue la siguiente línea en la parte superior de la aplicación:

```
#include <GDBStub.h>
```

Y en la función `void setup()` asegúrese de que el puerto serie esté inicializado y llame a `gdbstub_init()`:

```
Serial.begin(115200);  
gdbstub_init();
```

Vuelva a compilar y cargar su aplicación, debería ejecutarse exactamente como antes.

## 8.3 Iniciando la sesión de depuración

Una vez que su aplicación se está ejecutando, el proceso para añadir un depurador es bastante simple:

- Cierre el Monitor Serie de Arduino
- Localice el archivo Application.ino.elf
- Abra un Símbolo del sistema e inicie GDB
- Aplicar las configuraciones GDB
- Añadir el depurador
- A depurar!

### 8.3.1 Cierre el Monitor Serie de Arduino

Debido a que GDB necesita el control total del puerto serie, deberá cerrar todas las ventanas del Monitor Serie de Arduino que pueda tener abiertas. De lo contrario, GDB informará de un error al intentar depurar.

### 8.3.2 Localice el archivo Application.ino.elf

Para que GDB pueda depurar su aplicación, necesita localizar la versión compilada en formato ELF (que incluye los símbolos de depuración necesarios).

Bajo Linux, estos archivos se almacenan en `/tmp/arduino_build_*` y el siguiente comando ayudará a localizar el archivo correcto para su aplicación:

```
find /tmp -name "*.elf" -print
```

Bajo Windows, estos archivos se almacenan en `%userprofile%\AppData\Local\Temp\arduino_build_*` y el siguiente comando ayudará a localizar el archivo correcto para su aplicación:

```
dir %userprofile%\appdata\*.elf /s/b
```

Tenga en cuenta que la ruta completa del archivo ELF que corresponde al nombre de su boceto, se necesitará más adelante una vez que se inicie GDB.

### 8.3.3 Abra un Símbolo del sistema e inicie GDB

Abra un terminal o Símbolo de sistema y navegue hasta el directorio adecuado de la cadena de herramientas ESP8266.

Linux

```
~/arduino15/packages/esp8266/hardware/xtensa-lx106-elf/bin/xtensa-lx106-elf-gdb
```

Windows (Usando versión del Gestor de Tarjetas)

```
%userprofile%\AppData\Local\Arduino15\packages\esp8266\tools\xtensa-lx106-elf-gcc\2.5.0-3-20ed2b9\bin\xtensa-lx106-elf-gdb.exe
```

Windows (Usando versión Git)

```
%userprofile%\Documents\Arduino\hardware\esp8266com\esp8266\tools\xtensa-lx106-elf\bin\xtensa-lx106-elf-gdb.exe
```



Tenga en cuenta que el nombre correcto de GDB es «xtensa-lx106-elf-gdb». Si ejecuta accidentalmente «gdb», puede iniciar el propio GDB de su sistema operativo, que no sabrá cómo hablar con el ESP8266.

### 8.3.4 Aplicar las configuraciones GDB

En el prompt (gdb), ingrese las siguientes opciones para configurar GDB con el mapa de memoria ESP8266 y la configuración:

```
set remote hardware-breakpoint-limit 1
set remote hardware-watchpoint-limit 1
set remote interrupt-on-connect on
set remote kill-packet off
set remote symbol-lookup-packet off
set remote verbose-resume-packet off
mem 0x20000000 0x3fefffff ro cache
mem 0x3ff00000 0x3ffffff rw
mem 0x40000000 0x400fffff ro cache
mem 0x40100000 0x4013ffff rw cache
mem 0x40140000 0x5ffffff ro cache
mem 0x60000000 0x60001fff rw
set serial baud 115200
```

Ahora dile a GDB dónde se encuentra tu archivo ELF compilado:

```
file /tmp/arduino_build_257110/sketch_dec26a.ino.elf
```

### 8.3.5 Añadir el depurador

Una vez que GDB se haya configurado correctamente y haya cargado sus símbolos de depuración, conéctelo al ESP con el comando (reemplace ttyUSB0 o COM9 con el puerto serie de su ESP):

```
target remote /dev/ttyUSB0
```

o

```
target remote \\.\COM9
```

En este punto, GDB enviará una detención de la aplicación en el ESP8266 y podrá comenzar a configurar un punto de interrupción (break loop) o cualquier otra operación de depuración.

## 8.4 Ejemplo de sesión de depuración

Cree un nuevo boceto y pegue el siguiente código en él:

```
#include <GDBStub.h>

void setup() {
  Serial.begin(115200);
  gdbstub_init();
  Serial.printf("Iniciando...\n");
}
```

(continues on next page)

(proviene de la página anterior)

```
void loop() {
  static uint32_t cnt = 0;
  Serial.printf("%d\n", cnt++);
  delay(100);
}
```

Guárdelo, compílo y carguelo en su ESP8266. En el Monitor Serie debería ver algo como:

```
Iniciando...
1
2
3
....
```

Ahora cierre el Monitor Serie.

Abra un Símbolo de sistema y busque el fichero ELF:

```
earle@server:~$ find /tmp -name "*.elf" -print
/tmp/arduino_build_257110/testgdb.ino.elf
/tmp/arduino_build_531411/listfiles.ino.elf
/tmp/arduino_build_156712/SDWebServer.ino.elf
```

En este ejemplo, se encuentran varios archivos elf, pero solo nos importa el que acabamos de crear, testgdb.ino.elf.

Abre el GDB específico para ESP8266 adecuado

```
earle@server:~$ ~/.arduino15/packages/esp8266/hardware/xtensa-lx106-elf/bin/xtensa-
↳lx106-elf-gdb
GNU gdb (GDB) 8.2.50.20180723-git
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "--host=x86_64-linux-gnu --target=xtensa-lx106-elf".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
  <http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
  <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word".
(gdb)
```

Ahora estamos en el indicador de GDB, pero no se ha configurado nada para el ESP8266 y no se ha cargado información de depuración. Cortar y pegar las opciones de configuración:

Y dile a GDB dónde se encuentra el archivo ELF de información de depuración:

```
(gdb) file /tmp/arduino_build_257110/testgdb.ino.elf
Reading symbols from /tmp/arduino_build_257110/testgdb.ino.elf...done.
```

Ahora, conéctate al ESP8266 en ejecución:

```
(gdb) target remote /dev/ttyUSB0
Remote debugging using /dev/ttyUSB0
0x40000f68 in ?? ()
(gdb)
```

No se preocupe de que GDB no sepa qué hay en nuestra dirección actual, ingresamos el código en un lugar aleatorio y podríamos estar en una interrupción, en la ROM o en cualquier otro lugar. Lo importante es que ahora estamos conectados y ahora sucederán dos cosas: podemos depurar y la salida Serie de la aplicación se mostrará en la consola GDB.

Continúa la aplicación en ejecución para ver la salida en Serie:

```
(gdb) cont
Continuing.
74
75
76
77
...
```

La aplicación vuelve a funcionar y podemos detenerla en cualquier momento usando Ctrl-C:

En este punto, podemos establecer un punto de interrupción en el `loop()` principal y reiniciar para ingresar nuestro propio código:

```
(gdb) break loop
Breakpoint 1 at 0x40202e33: file /home/earle/Arduino/sketch_dec26a/sketch_dec26a.ino,
↳line 10.
(gdb) cont
Continuing.
Note: automatically using hardware breakpoints for read-only addresses.
bcn_timeout,ap_probe_send_start

Breakpoint 1, loop () at /home/earle/Arduino/sketch_dec26a/sketch_dec26a.ino:10
10 void loop()
(gdb)
```

Examinemos la variable local:

Y cambiémosla:

```
$2 = 114
(gdb) set cnt = 2000
(gdb) print cnt
$3 = 2000
(gdb)
```

Y reinicie la aplicación y vea que nuestros cambios surten efecto:

```
(gdb) cont
Continuing.
2000
Breakpoint 1, loop () at /home/earle/Arduino/sketch_dec26a/sketch_dec26a.ino:10
10 void loop() {
(gdb) cont
Continuing.
2001
Breakpoint 1, loop () at /home/earle/Arduino/sketch_dec26a/sketch_dec26a.ino:10
```

(continues on next page)

(proviene de la página anterior)

```
10 void loop() {  
(gdb)
```

Parece que dejamos el punto de interrupción en `loop()`, deshagámonos de él e intentemos nuevamente:

```
(gdb) delete  
Delete all breakpoints? (y or n) y  
(gdb) cont  
Continuing.  
2002  
2003  
2004  
2005  
2006  
.....
```

En este punto, podemos salir de GDB con `quit` o hacer más depuración.

## 8.5 Limitaciones de depuración del Hardware ESP8266

El ESP8266 solo admite un único punto de interrupción de hardware y un solo punto de vigilancia de datos de hardware. Esto significa que solo se permite un punto de interrupción en el código de usuario en cualquier momento. Considere usar el comando `thb` (punto de interrupción temporal de hardware) en GDB mientras realiza la depuración en lugar del comando más común `break`, ya que `thb` eliminará el punto de interrupción una vez que se alcance automáticamente y le ahorrará algunos problemas .

## 9.1 Módulo Genérico ESP8266

Estos módulos se encuentran con diferentes formas y pineado. Vea la página wiki de la comunidad ESP8266 para más información: [Módulos de la familia ESP8266](#).

Normalmente estos módulos no tienen resistencias de arranque en la tarjeta (pullup y pulldown), insuficientes condensadores de desacoplamiento, sin regulador de voltaje, sin circuito de reset y sin adaptador USB-Serie. Esto hace que usarlos sea algo complicado, en comparación con las placas de desarrollo que agregan estas características.

Para utilizar estos módulos, asegúrese de observar lo siguiente:

- **Proveer energía suficiente al módulo.** Para un uso estable del ESP8266 se requiere una fuente de alimentación con 3.3V y  $\geq 250\text{mA}$ . No se recomienda utilizar la alimentación del adaptador USB-Serie, Estos adaptadores normalmente no suministran corriente suficiente para correr el ESP8266 de forma segura en todas las situaciones. Es preferible un suministro externo o regulador junto con condensadores de filtrado.
- **Conectar resistencias de arranque** a GPIO0, GPIO2, GPIO15 de acuerdo con los esquemas a continuación.
- **Poner el ESP8266 en modo bootloader** antes de subir código.

## 9.2 Adaptador Serie

Hay muchos adaptadores/tarjetas USB a Serie diferentes. Para poner ESP8266 en modo bootloader utilizando líneas de handshaking en serie, necesita el adaptador que interrumpa las salidas RTS y DTR. CTS y DSR no son necesarios para cargar (son entradas). Asegúrese de que el adaptador funciona con voltaje de 3.3V IO: debe tener un puente o un interruptor para seleccionar entre 5V y 3.3V, o estar marcado como 3.3V solamente.

Los adaptadores basados en los siguientes chips deben funcionar:

- FT232RL
- CP2102
- CH340G

Los adaptadores basados en PL230 no funcionan en Mac OS X. Ver <https://github.com/igrr/esptool-ck/issues/9> para mas información.

## 9.3 Configuración Hardware mínima para Bootloading y ejecución

PIN	Resistencia	Adaptador Serie
VCC		VCC (3.3V)
GND		GND
TX o GPIO2*		RX
RX		TX
GPIO0	PullUp	DTR
Reset*	PullUp	RTS
GPIO15*	PullDown	
CH_PD	PullUp	

Note: - GPIO15 se puede llamar también MTDO. - Reset se puede llamar también RSBT o REST (añadir PullUp para mejorar la estabilidad del módulo). - GPIO2 es alternativo TX para el modo bootloader. - **Conectar directamente un pin a VCC o GND no sirve para sustituir una resistencia PullUp o PullDown, hacer esto puede afectar al control de subida y al monitor serie, también puede notar inestabilidad en algunos casos.**

## 9.4 ESP a Serie

### 9.4.1 Configuración Hardware mínima para Bootloading solo

ESPxx Hardware

PIN	Resistencia	Adaptador Serie
VCC		VCC (3.3V)
GND		GND
TX o GPIO2		RX
RX		TX
GPIO0		GND
Reset		RTS*
GPIO15	PullDown	
CH_PD	PullUp	

Note: - Si no utiliza RTS es necesario un reinicio manual de alimentación

### 9.4.2 Configuración Hardware mínima para solo ejecutar

ESPxx Hardware

PIN	Resistencia	Fuente aliment.
VCC		VCC (3.3V)
GND		GND
GPIO0	PullUp	
GPIO15	PullDown	
CH_PD	PullUp	

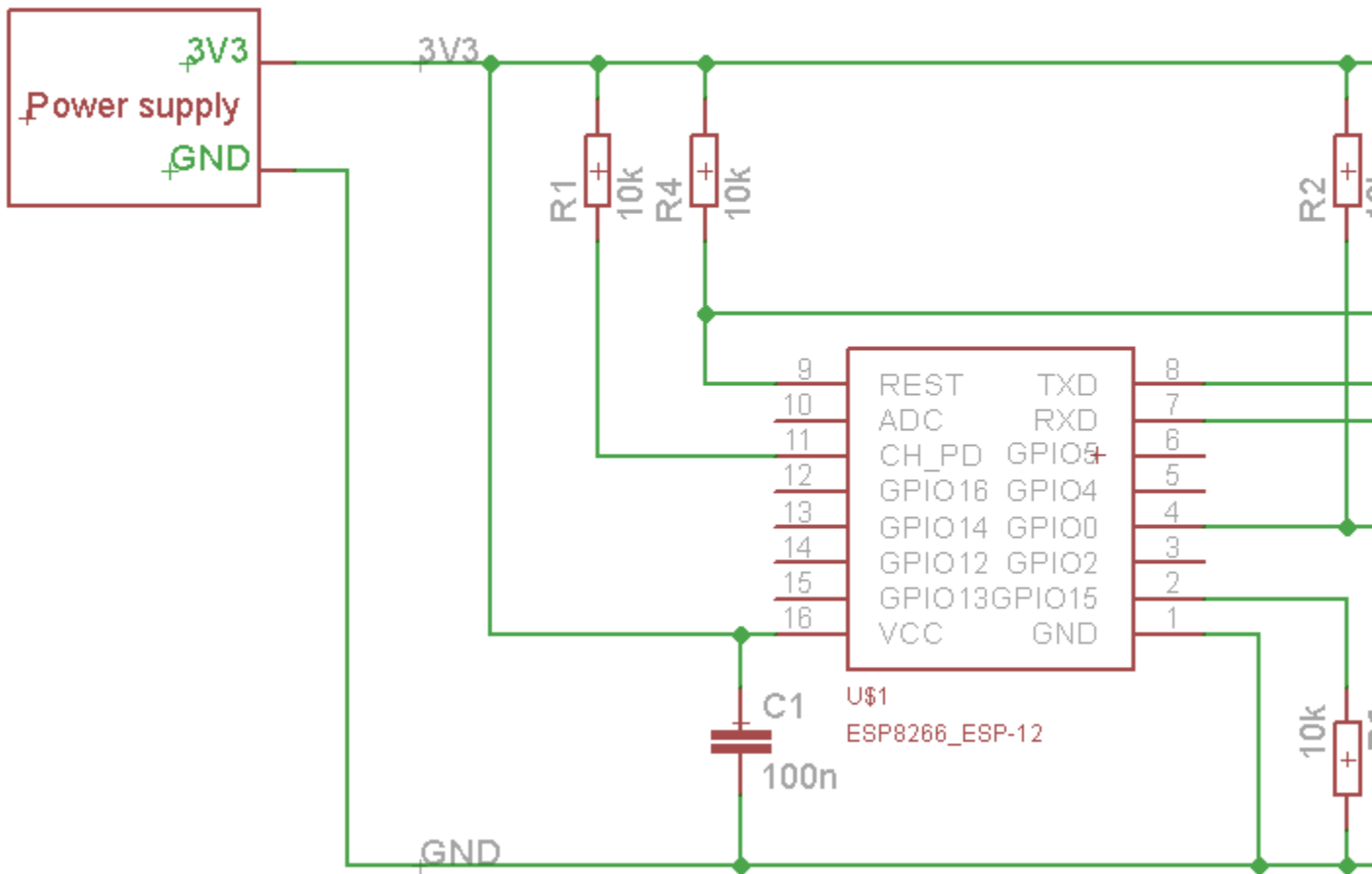


Figura 1: ESP to Serial

## 9.5 Mínimo

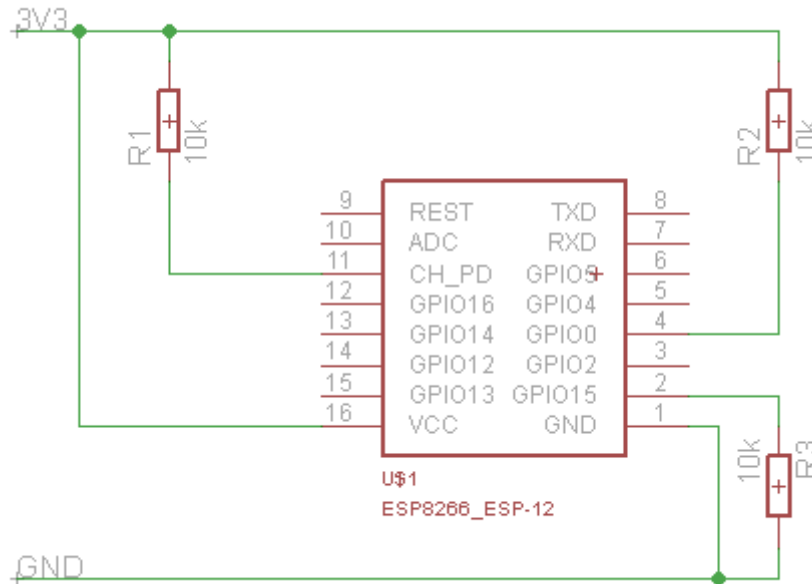


Figura 2: ESP min

## 9.6 Estabilidad mejorada

## 9.7 Mensajes de arranque y modos

El módulo ESP comprueba en cada arranque los pines 0, 2 y 15. Arrancando basado en ellos de diferente modo:

GPIO15	GPIO0	GPIO2	Modo
0V	0V	3.3V	UART Bootloader
0V	3.3V	3.3V	Inicia el sketch (SPI flash)
3.3V	x	x	Modo SDIO (no usado para Arduino)

Al inicio el ESP imprime el modo actual de arranque, ejemplo:

```
rst cause:2, boot mode:(3,6)
```

Nota: - GPIO2 se utiliza como salida TX y el Pullup interno está activo al arrancar.



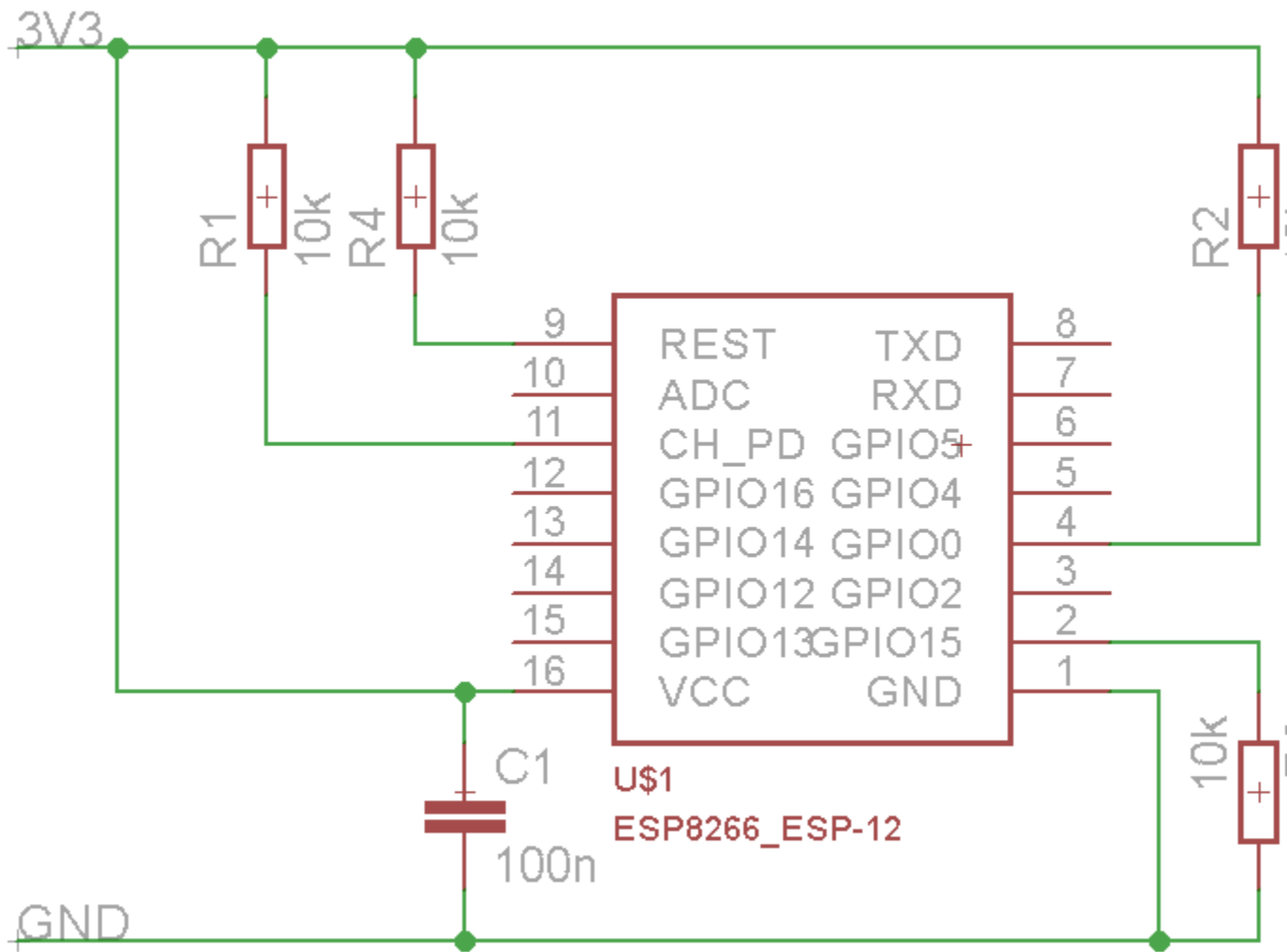


Figura 3: ESP con estabilidad mejorada

### 9.7.1 Causas de reset

Número	Descripción
0	desconocido
1	inicio normal
2	reset pin
3	software reset
4	watchdog reset

### 9.7.2 Modo de arranque

Es el primer valor respecto a la configuración de pines 0, 2 y 15.

Número	GPIO15	GPIO0	GPIO2	Modo
0	0V	0V	0V	No valido
1	0V	0V	3.3V	UART
2	0V	3.3V	0V	No valido
3	0V	3.3V	3.3V	Flash
4	3.3V	0V	0V	SDIO
5	3.3V	0V	3.3V	SDIO
6	3.3V	3.3V	0V	SDIO
7	3.3V	3.3V	3.3V	SDIO

Nota: - Numero = ((GPIO15 << 2) | (GPIO0 << 1) | GPIO2);

## 9.8 Módulo Genérico ESP8285

ESP8285 ([datasheet](#)) es un paquete multichip el cual contiene un ESP8266 y una flash de 1MB. Todos los puntos relacionados a resistencias de arranque y circuitos recomendados arriba también se aplican a ESP8285.

Nota: Debido a que ESP8285 tiene la memoria flash SPI conectada internamente en modo DOUT, los pines 9 y 10 pueden utilizarse como pines GPIO / I2C / PWM.

## 9.9 ESPDuino (Módulo ESP-13)

*TODO*

## 9.10 Adafruit Feather HUZAZH ESP8266

El ESP8266 Adafruit Feather HUZAZH es una tarjeta de desarrollo WiFi Arduino-compatible alimentada por un módulo ESP-12S de Ai-Thinker, con reloj a 80 MHz y lógica de 3.3V. Incluye un chip USB-Serie de alta calidad SiLabs CP2104 por lo que puedes subir el código a la abrasadora velocidad de 921600 baudios, para un tiempo de desarrollo rápido. También tiene reinicio automático, por lo que no hay que tocar ningún pin y ni presionar nada para reset. Incluye un conector de batería de polímero de litio de 3.7V, por lo que es ideal para proyectos portátiles. El ESP8266 Adafruit Feather HUZAZH recargará automáticamente una batería conectada cuando la alimentación USB esté disponible.

Página del producto: <https://www.adafruit.com/product/2821>

## 9.11 Invent One

Invent One es una tarjeta de desarrollo WiFi compatible con Arduino alimentada por un Ai-Thinker ESP-12F, con reloj a 80 MHz y lógica de 3.3V. Posee un ADC (PCF8591) integrado con multiples entradas analógicas para trabajar con ellas. Mas información aquí: <https://blog.inventone.ng>

Página del producto: <https://inventone.ng>

## 9.12 XinaBox CW01

XinaBox CW01 es una tarjeta de desarrollo WiFi compatible con Arduino alimentada por un Ai-Thinker ESP-12F, con reloj a 80 MHz y lógica de 3.3V. Posee un led RGB integrado.

Página del producto: <https://xinabox.cc/products/CW01>

## 9.13 ESPresso Lite 1.0

ESPresso Lite 1.0 (versión beta) es una tarjeta de desarrollo WiFi Arduino-compatible alimentada por su propio módulo Epressif System's WROOM-02. Posee un amigable pineado tipo breadboard con un LED integrado, dos botones reset/flash y un botón programable por el usuario. El voltaje de trabajo es 3.3VDC, regulado con corriente máxima 800mA. Una característica distintiva especial es que posee un pad integrado I2C pads el cual permite una conexión directa a un LCD OLED y tarjetas de sensores.

## 9.14 ESPresso Lite 2.0

ESPresso Lite 2.0 es una tarjeta de desarrollo WiFi Arduino-compatible basada en la V1 (beta versión). Rediseñada junto con Cytron Technologies, La nueva/revisada ESPresso Lite V2.0 posee la función de auto carga/auto programación, eliminando la anterior necesidad de resetear la tarjeta manualmente tras flasear un nuevo programa. También posee dos botones programables por el usuario y un botón de reset. El distintivo especial es que posee pads integrados para I2C sensor.

## 9.15 Phoenix 1.0

Página del producto: <http://www.espert.co>

## 9.16 Phoenix 2.0

Página del producto: <http://www.espert.co>

## 9.17 NodeMCU 0.9 (Módulo ESP-12)

### 9.17.1 Mapa de pines

La numeración de pines impresa en la tarjeta no se corresponde con la numeración GPIO del ESP8266. Se han definido constantes para facilitar un uso sencillo:

```
static const uint8_t D0 = 16;
static const uint8_t D1 = 5;
static const uint8_t D2 = 4;
static const uint8_t D3 = 0;
static const uint8_t D4 = 2;
static const uint8_t D5 = 14;
static const uint8_t D6 = 12;
static const uint8_t D7 = 13;
static const uint8_t D8 = 15;
static const uint8_t D9 = 3;
static const uint8_t D10 = 1;
```

Si deseas usar el pin 5 del NodeMCU, utiliza «D5» como número del pin y será traducido al real GPIO pin 14.

## 9.18 NodeMCU 1.0 (Módulo ESP-12E)

Este módulo se vende con muchos nombres en AliExpress por menos de 6.5\$ y es uno de los mas baratos, posee soluciones completamente integradas en el ESP8266.

Se trata de un diseño Open Hardware con un core ESP-12E y una flash SPI de 4 MB.

De acuerdo con el fabricante, «con un micro cable USB, puedes conectar el kit de desarrollo NodeMCU a tu portátil y flasearlo sin ningún problema». Este es mas o menos verdad: la tarjeta viene con un adaptador integrado USB-Serie CP2102 el cual funciona bien la mayoría de veces. Algunas veces falla y necesitas resetear la tarjeta pulsando y manteniendo FLASH y RST, soltando FLASH y entonces soltando RST. Esto fuerza al dispositivo CP2102 a realizar un ciclo de alimentación y a ser reenumerado en Linux.

La tarjeta también integra un regulador de voltaje NCP1117, un LED azul en GPIO16 y un divisor de voltaje 220k/100k Ohm en el pin de entrada ADC.

El ESP-12E normalmente tiene un led conectado en el GPIO2.

El pinout completo y esquema en PDF, se encuentra [aquí](#)

## 9.19 Olimex MOD-WIFI-ESP8266(-DEV)

Esta tarjeta tiene flash SPI de 2 MB y accesorios adicionales (p.ej. tarjeta de evaluación ESP8266-EVB o BAT-BOX para baterías).

El módulo básico tiene 3 jumpers soldados que te permiten cambiar el modo de operación entre SDIO, UART y FLASH.

La tarjeta se envía en el modo de operación FLASH, con jumpers TD0JP=0, IO0JP=1, IO2JP=1.

Como el jumper IO0JP está vinculado a GPIO0, que es PIN 21, tendrás que conectarlo a tierra antes de programarlo con un adaptador de USB a Serie y reiniciar la placa apagándola.

Los pines UART para programación y E/S Serial son GPIO1 (TXD, pin 3) y GPIO3 (RXD, pin 4).

Puedes ver el esquema de la tarjeta [aquí](#)

## 9.20 SparkFun ESP8266 Thing

Página del producto: <https://www.sparkfun.com/products/13231>

## 9.21 SparkFun ESP8266 Thing Dev

Página del producto: <https://www.sparkfun.com/products/13711>

## 9.22 SweetPea ESP-210

*TODO*

## 9.23 LOLIN(WEMOS) D1 R2 & mini

Página del producto: <https://www.wemos.cc/>

## 9.24 LOLIN(WEMOS) D1 mini Pro

Página del producto: <https://www.wemos.cc/>

## 9.25 LOLIN(WEMOS) mini Lite

### 9.25.1 Parámetros en el IDE Arduino:

- Tarjeta: «WEMOS D1 Mini Lite»
- Tamaño de la flash: «1M (512K SPIFFS)»
- Frecuencia de la CPU: «80 Mhz»
- Velocidad de subida: «230400»

### 9.25.2 Potencias:

- Pin 5V : salida 4.7V 500mA cuando la tarjeta se alimenta mediante USB; Entrada 3.5V-6V
- Pin 3V3 : salida regulada 3.3V 500mA
- Pines digitales : 3.3V 30mA.

### 9.25.3 Enlaces:

- Página del producto: <https://www.wemos.cc/>
- Esquema de la tarjeta: [https://wiki.wemos.cc/\\_media/products:d1:sch\\_d1\\_mini\\_lite\\_v1.0.0.pdf](https://wiki.wemos.cc/_media/products:d1:sch_d1_mini_lite_v1.0.0.pdf)
- Datasheet del ESP8285: [https://www.espressif.com/sites/default/files/0a-esp8285\\_datasheet\\_en\\_v1.0\\_20160422.pdf](https://www.espressif.com/sites/default/files/0a-esp8285_datasheet_en_v1.0_20160422.pdf)
- Datasheet del regulador de voltaje: <http://pdf-datasheet.datasheet.netdna-cdn.com/pdf-down/M/E/6/ME6211-Microne.pdf>

## 9.26 WeMos D1 R1

Página del producto: <https://www.wemos.cc/>

## 9.27 ESPino (Módulo ESP-12)

ESPino integra el módulo ESP-12 con un regulador de 3.3v, adaptador USB-Serie CP2104 y un conector micro USB para una fácil programación. Está diseñado para adaptarse a una a breadboard ay tiene un led RGB y 2 botones para prototipado fácil.

Mas información sobre el hardware, pinout, diagrama y procedimiento de programación, por favor vea el [datasheet](#).

Página del producto: <http://www.espino.io/en>

## 9.28 ThaiEasyElec's ESPino

ESPino by ThaiEasyElec utiliza el módulo WROOM-02 de Espressif Systems con flash de 4 MB.

Se actualizará una descripción pronto. - Página del producto: <http://thaieasyelec.com/products/wireless-modules/wifi-modules/espino-wifi-development-board-detail.html> - Esquema: [www.thaieasyelec.com/downloads/ETEE052/ETEE052\\_ESPino\\_Schematic.pdf](http://www.thaieasyelec.com/downloads/ETEE052/ETEE052_ESPino_Schematic.pdf) - Dimensiones: [http://thaieasyelec.com/downloads/ETEE052/ETEE052\\_ESPino\\_Dimension.pdf](http://thaieasyelec.com/downloads/ETEE052/ETEE052_ESPino_Dimension.pdf) - Pinouts: [http://thaieasyelec.com/downloads/ETEE052/ETEE052\\_ESPino\\_User\\_Manual\\_TH\\_v1\\_0\\_20160204.pdf](http://thaieasyelec.com/downloads/ETEE052/ETEE052_ESPino_User_Manual_TH_v1_0_20160204.pdf) (Ver pág.. 8)

## 9.29 WifInfo

WifInfo integra el módulo ESP-12 o el ESP-07+Ext antena con un regulador de 3.3v y el hardware es capaz de medir la telemetría francesa a partir de la salida en serie del medidor de potencia ERDF. Tiene un conector USB para alimentación, un Led RGB WS2812, conector I2C de 4 pines para adaptarse a OLED o sensor, y dos botones + conector FTDI y función de reinicio automático.

Mas información sobre WifInfo, ver el siguiente [blog](#) , [Github](#) y [foro de la comunidad](#).

## 9.30 Arduino

*TODO*

## 9.31 4D Systems gen4 IoD Range

gen4-IoD Range de ESP8266 tiene módulos de pantalla de 4D Systems.

2.4», 2.8» y 3.2» TFT LCD con uSD card socket y táctil resistivo. Chip de antena + conector uFL.

Datasheet y descargas asociadas pueden encontrarse en la página del producto de 4D Systems.

La gama de productos gen4-IoD puede programarse utilizando el IDE de Arduino y también IDE 4D Systems Workshop4, el cual incorpora muchos beneficios gráficos adicionales. La librería GFX4d está disponible, junto con una serie de aplicaciones de demostración.

Página del producto: <http://www.4dsystems.com.au/product/gen4-IoD>

## 9.32 Digistump Oak

El Oak requiere un *Adaptador Serie* para la conexión serie o flaseado; Su puerto micro USB es solo para alimentación.

Para realizar la conexión serie, conecte el adaptador **TX a P3, RX a P4 y GND a GND**. Alimentar 3.3v desde el adaptador serie si no está ya alimentado mediante el USB.

Para poner la tarjeta en modo bootloader, configure una conexión serie como anteriormente y conecte **P2 a GND**, después vuelva a alimentar. Una vez que el flaseado se ha completado, elimine la conexión de P2 a GND, entonces vuelva a alimentar para iniciar en modo normal.

## 9.33 WiFiduino

Página del producto: <https://wifiduino.com/esp8266>

## 9.34 Amperka WiFi Slot

Página del producto: <http://wiki.amperka.ru/wifi-slot>

## 9.35 Seeed Wio Link

Wio Link está diseñado para simplificar tu desarrollo IoT. Es una tarjeta de desarrollo open-source WiFi basada en un ESP8266 para crear aplicaciones IoT virtualizando módulos «plug and play» a APIs RESTful con aplicaciones APP. Wio Link también es compatible con el IDE Arduino.

ADVIERTA que DEBE pull up el pin 15 para activar la alimentación de los puertos Grove, la placa está diseñada de esta manera para la administración de energía de periféricos.

Página del producto: <https://www.seeedstudio.com/Wio-Link-p-2604.html>

## 9.36 ESPECTRO Core

ESPECTRO Core es una placa de desarrollo ESP8266 culminación de nuestra experiencia de más de 3 años en la exploración y el desarrollo de productos con ESP8266 MCU.

Inicialmente diseñado para niños en mente, todos deberían poder usarlo. Sin embargo, sigue siendo fácil para los hackers informáticos ya que rompemos todos los pines del ESP8266 ESP-12F.

Más detalles en <https://shop.makestro.com/product/espectrocore/>



El propósito de este apartado «Preguntas - FAQ» es responder a las preguntas mas comunes en la sección [Issues](#) y en el [Foro de la comunidad ESP8266](#).

Siempre que sea posible, vamos directamente a la respuesta y la proporcionamos dentro de uno o dos párrafos. Si la respuesta es mas larga, verá un enlace para leer mas detalles.

Siéntase libre de contribuir si cree que alguna pregunta frecuente no se encuentra cubierta.

### **10.1 Obtengo el error «espcomm\_sync failed» cuando intento subir a mi ESP. ¿Como resuelvo este problema?**

Este mensaje indica un problema al subir al módulo ESP mediante conexión por puerto serie. Existen varias posibles causas que dependen del tipo de módulo y de si tiene un convertidor serie independiente.

[Leer mas.](#)

### **10.2 ¿Porqué no aparece esptool en el menú «Programador»? ¿Como subo al ESP sin él?**

No te preocupes por el menú «Programador» del IDE Arduino. No importa qué se seleccione en él, siempre está predeterminado para usar esptool.

Ref. [#138](#), [#653](#) y [#739](#).

### **10.3 Mi ESP se bloquea al correr el programa. ¿Como lo resuelvo?**

El programa puede bloquearse por un error software o hardware. Antes de abrir un nuevo issue, por favor realice una serie de comprobaciones iniciales.

Leer mas.

## 10.4 ¿Como puedo obtener algunos KBs extra en la flash?

La utilización de `*printf()` con floats está activada por defecto. Puedes salvar algunos KBs de flash utilizando la opción `--nofloat` con el generador de tarjetas:

```
./tools/boards.txt.py --nofloat --boardsgen
```

Utiliza la opción del nivel de debug `NoAssert-NDEBUG` (en el menú Herramientas).

Leer mas.

## 10.5 Sobre WPS

A partir de la versión 2.4.2 y superiores, no utilizar WPS libera ~4.5KB extra en heap.

En la versión 2.4.2 solo, WPS está desactivado por defecto y se requiere el Generador de tarjetas para activarlo:

```
./tools/boards.txt.py --allowWPS --boardsgen
```

Leer mas.

Para `platformIO` (y posiblemente en otros entornos de desarrollo), también necesitas añadir la bandera de compilación (build flag): `-D NO_EXTRA_4K_HEAP`

La selección manual no es necesaria a partir de la versión 2.5.0 (y en la versión git). El WPS está siempre disponible y no usarlo libera ~4.5KB comparado con las versiones anteriores a 2.4.1 (incluida).

## 10.6 Esta librería de Arduino no funciona en ESP. ¿Como la hago funcionar?

Te gustaría usar una librería de Arduino con ESP8266 y no funciona. Si no se encuentra entre las bibliotecas verificadas para trabajar con ESP8266:

Leer mas.

## 10.7 En el IDE, para ESP-12E que tiene una flash de 4M, puedo seleccionar 4M (1M SPIFFS) o 4M (3M SPIFFS). No importa lo que seleccione, el IDE me dice que la capacidad máxima es de 1M. ¿Donde va mi flash?

La razón de que no podamos tener mas de 1MB de código en la flash tiene que ver con limitaciones hardware. El hardware de la cache flash en el ESP8266 solo permite mapear 1MB de código en el espacio de direcciones de la CPU en cualquier momento dado. Puedes cambiar el desplazamiento de mapeo, por lo que técnicamente puede tener más de 1 MB total, pero cambiar esos «bancos» sobre la marcha no es fácil y eficiente, así que no nos molestamos en hacerlo. Además, nadie se ha quejado hasta ahora de que los aproximadamente 1 MB de espacio de código sea insuficiente para fines prácticos.

La opción de seleccionar 3M o 1M SPIFFS es para optimizar el tiempo de subida. Subir 3MB toma mas tiempo que subir 1MB. Otras capacidades de flash 2MB también pueden utilizarse con las APIs `ESP.flashRead` y `ESP.flashWrite` si es necesario.

## 10.8 He observado que `ESP.restart()` no funciona. Cual es la razón

Verá este problema solo si después de subir el programa mediante puerto serie no realiza un reset físico (por ejemplo, reinicio de la alimentación). Para un dispositivo que se encuentre en ese estado, `ESP.restart` no funcionará. Aparentemente, el problema está causado por uno de los registros internos que no se actualiza correctamente hasta el reseteo físico. Este problema solo afecta a las subidas mediante puerto serie. Las subidas mediante OTA no se ven afectadas. Si está utilizando `ESP.restart`, solo reinicie ESP físicamente una vez después de cada subida por puerto serie.

Ref. #1017, #1107, #1782

## 10.9 ¿Como solucionar el error «Board generic (platform esp8266, package esp8266) is unknown»?

Este error puede aparecer al cambiar entre los paquetes de instalación de ESP8266/Arduino `staging` y `stable`, o tras actualizar de versión de paquete.

Leer mas.

## 10.10 ¿Cómo borrar PCBs TCP en estado de espera de tiempo?

Esto ya no es necesario:

Los PCBs en tiempo de estado de espera están limitados a 5 y se eliminan cuando ese número es excedido.

Ref. lwIP-v1.4, lwIP-v2

Como información:

El estado Time-wait PCB ayuda al TCP a no confundir dos conexiones consecutivas con el mismo: IP de origen ip, puerto de origen, IP de destino y puerto de destino, cuando el primero ya está cerrado pero aún están llegando tarde durante segundos paquetes duplicados perdidos en internet. Limpiarlos artificialmente es una solución alternativa para ayudar a salvar heap preciosos.

La líneas siguientes son compatibles con ambas versiones de lwIP:

```
// no need for #include
struct tcp_pcb;
extern struct tcp_pcb* tcp_tw_pcbs;
extern "C" void tcp_abort (struct tcp_pcb* pcb);

void tcpCleanup (void) {
    while (tcp_tw_pcbs)
        tcp_abort(tcp_tw_pcbs);
}
```

Ref. #1923

## 10.11 ¿Por qué hay un generador de tarjetas y para que sirve?

El generador de tarjetas es una secuencia de comandos python originalmente destinada a facilitar el archivo de configuración *boards.txt* de Arduino IDE sobre la multitud de tarjetas disponibles, especialmente cuando los parámetros comunes deben actualizarse para todos ellos.

Este script también se usa para administrar opciones poco comunes que actualmente no están disponibles en el menú IDE.

Leer mas.

## Exception Causes (EXCCAUSE)

EXCCAUSE Code	Cause Name	Cause Description
0	IllegalInstructionCause	Illegal instruction
1	SyscallCause	SYSCALL instruction
2	InstructionFetchErrorCause	Processor internal physical address or data error during instruction fetch
3	LoadStoreErrorCause	Processor internal physical address or data error during load or store
4	Level1InterruptCause	Level-1 interrupt as indicated by set level-1 bits in the INTERRUPT register
5	AllocaCause	MOVSP instruction, if caller's registers are not in the register file
6	IntegerDivideByZeroCause	QUOS, QUOU, REMS, or REMU divisor operand is zero
7	Reserved for Tensilica	
8	PrivilegedCause	Attempt to execute a privileged operation when CRING != 0
9	LoadStoreAlignmentCause	Load or store to an unaligned address
10..11	Reserved for Tensilica	
12	InstrPIFDataErrorCause	PIF data error during instruction fetch
13	LoadStorePIFDataErrorCause	Synchronous PIF data error during LoadStore access
14	InstrPIFAddrErrorCause	PIF address error during instruction fetch
15	LoadStorePIFAddrErrorCause	Synchronous PIF address error during LoadStore access
16	InstTLBMissCause	Error during Instruction TLB refill
17	InstTLBMultiHitCause	Multiple instruction TLB entries matched
18	InstFetchPrivilegeCause	An instruction fetch referenced a virtual address at a ring level less than CRING
19	Reserved for Tensilica	
20	InstFetchProhibitedCause	An instruction fetch referenced a page mapped with an attribute that does not permit loads
21..23	Reserved for Tensilica	
24	LoadStoreTLBMissCause	Error during TLB refill for a load or store
25	LoadStoreTLBMultiHitCause	Multiple TLB entries matched for a load or store
26	LoadStorePrivilegeCause	A load or store referenced a virtual address at a ring level less than CRING
27	Reserved for Tensilica	
28	LoadProhibitedCause	A load referenced a page mapped with an attribute that does not permit loads
29	StoreProhibitedCause	A store referenced a page mapped with an attribute that does not permit stores
30..31	Reserved for Tensilica	

Tabla 1 – proviene de la página anterior

EXCCAUSE Code	Cause Name	Cause Description
32..39	CoprocessornDisabled	Coprocessor n instruction when cpn disabled. n varies 0..7 as the cause varies
40..63	Reserved	

Infos from Xtensa Instruction Set Architecture (ISA) Reference Manual

### 12.1 Introducción

Desde 2.1.0-rc1 el core incluye una función de depuración que es controlable desde el menú IDE.

Los nuevos puntos de menú administran los mensajes de depuración en tiempo real.

#### 12.1.1 Requerimientos

Para realizar la depuración se requiere una conexión serie (Serial o Serial1).

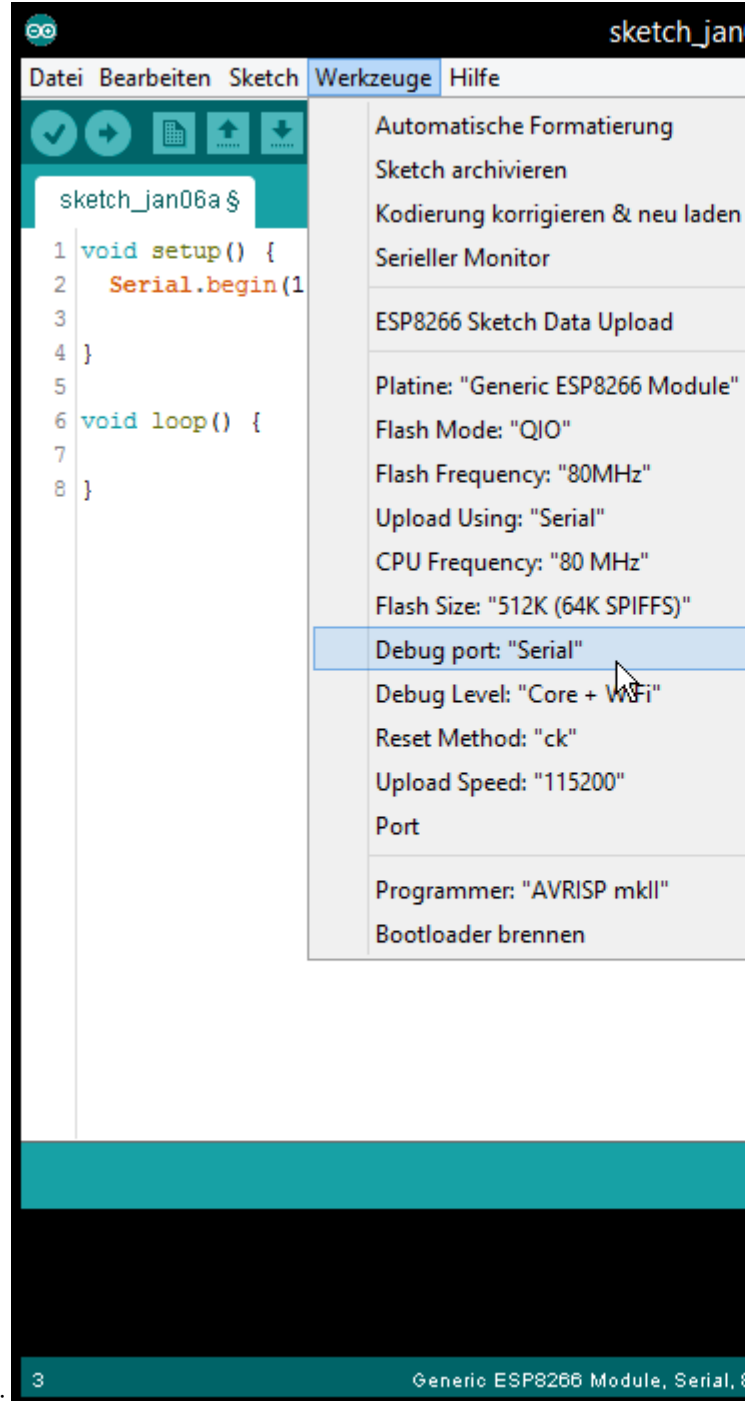
El interfaz Serial necesita ser inicializado en `setup()`.

Establece la velocidad en baudios de Serial tan alta como te lo permita tu Hardware.

Sketch mínimo para realizar la depuración:

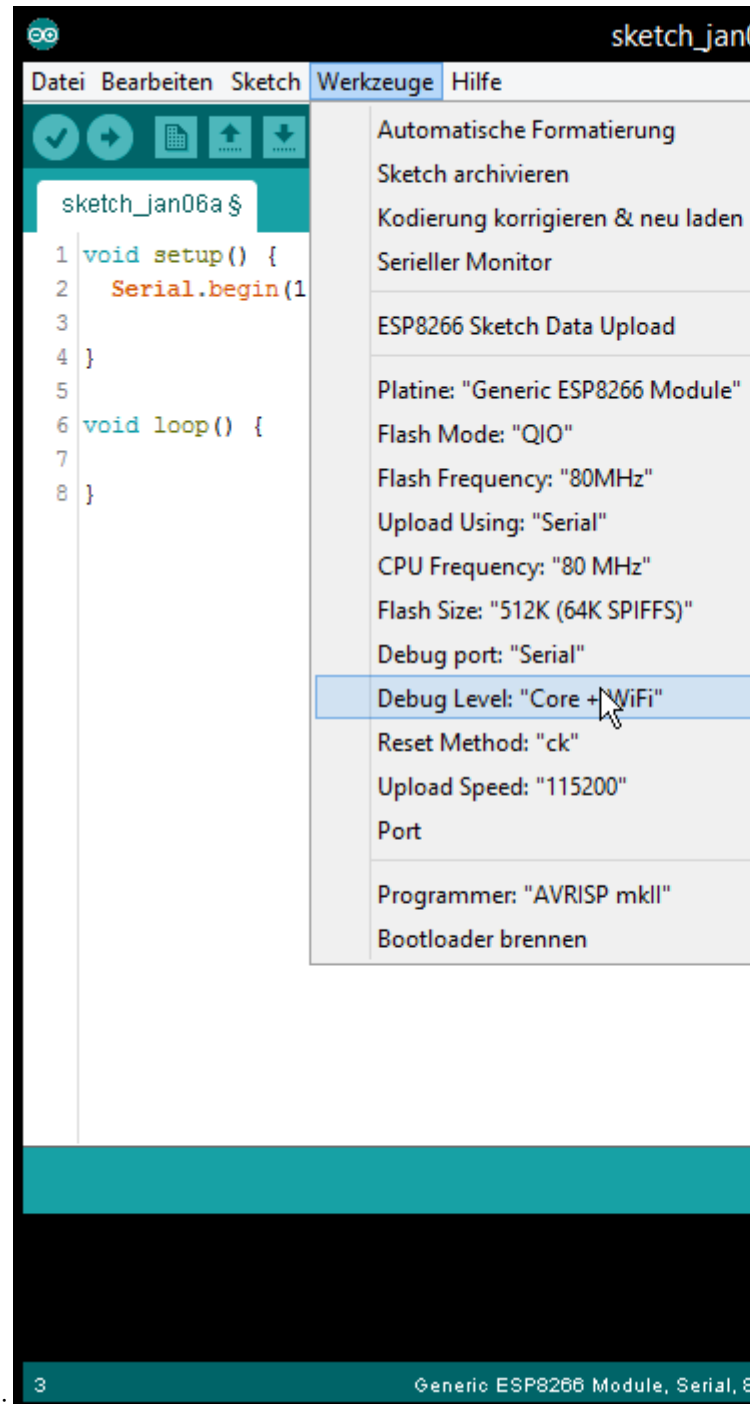
```
void setup() {  
    Serial.begin(115200);  
}  
  
void loop() {  
}
```

## 12.1.2 Uso



1. Selecciona el interfaz Serie para los mensajes de depuración:





2. Selecciona que tipo/nivel deseas de mensajes de depuración:
3. Comprueba si el interfaz Serial está inicializado en `setup()` (ver *Requerimientos*).
4. Sube el sketch.
5. Comprueba la salida serie.

## 12.2 Información

Funciona con cada sketch que active el interfaz serie que debe ser seleccionado como Debug Port.

El interfaz serie puede usarse normalmente también en el sketch.

La salida de depuración es adicional y no desactiva ningún interfaz del sketch.

### 12.2.1 Para desarrolladores

Para el manejo de la depuración utiliza defines.

La definición se realiza por líneas de comandos.

#### Debug Port - Puerto de depuración

El puerto tiene el define `DEBUG_ESP_PORT` posibles valores: - Desactivado: no existe el define. - Serial: `Serial` - Serial1: `Serial1`

#### Debug Level - Nivel de depuración

Todos los defines para los diferentes niveles comienzan con `DEBUG_ESP_`

Puede encontrar una lista completa en el fichero `boards.txt`

#### Ejemplo para sus propios mensajes de depuración

Los mensajes de depuración serán mostrados solo cuando se establezca el Debug Port en el menú del IDE.

```
#ifndef DEBUG_ESP_PORT
#define DEBUG_MSG(...) DEBUG_ESP_PORT.printf( __VA_ARGS__ )
#else
#define DEBUG_MSG(...)
#endif

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

  delay(3000);
  DEBUG_MSG("Iniciando...\n");
}

void loop() {
  DEBUG_MSG("loop %d\n", millis());
  delay(1000);
}
```

### 13.1 Introducción

Si se bloquea el ESP, se mostrará la Causa de excepción y se descargará la pila actual.

Ejemplo:

```
Exception (0): epc1=0x402103f4 epc2=0x00000000 epc3=0x00000000 excvaddr=0x00000000_
↳depc=0x00000000

ctx: sys
sp: 3ffffc10 end: 3fffffb0 offset: 01a0

>>>stack>>>
3ffffdb0: 40223e00 3fff6f50 00000010 60000600
3ffffdc0: 00000001 4021f774 3fffc250 4000050c
3ffffdd0: 400043d5 00000030 00000016 ffffffff
3ffffde0: 400044ab 3fffc718 3ffffed0 08000000
3ffffdf0: 60000200 08000000 00000003 00000000
3ffffe00: 0000ffff 00000001 04000002 003fd000
3ffffe10: 3fff7188 000003fd 3fff2564 00000030
3ffffe20: 40101709 00000008 00000008 00000020
3ffffe30: c1948db3 394c5e70 7f2060f2 c6ba0c87
3ffffe40: 3fff7058 00000001 40238d41 3fff6ff0
3ffffe50: 3fff6f50 00000010 60000600 00000020
3ffffe60: 402301a8 3fff7098 3fff7014 40238c77
3ffffe70: 4022fb6c 40230ebe 3fff1a5b 3fff6f00
3ffffe80: 3ffffec8 00000010 40231061 3fff0f90
3ffffe90: 3fff6848 3ffed0c0 60000600 3fff6ae0
3ffffea0: 3fff0f90 3fff0f90 3fff6848 3fff6d40
3ffffeb0: 3fff28e8 40101233 d634fe1a fffffeff
3ffffec0: 00000001 00000000 4022d5d6 3fff6848
3ffffed0: 00000002 4000410f 3fff2394 3fff6848
3ffffee0: 3fffc718 40004a3c 000003fd 3fff7188
```

(continues on next page)

(proviene de la página anterior)

```
3ffffef0: 3fffc718 40101510 00000378 3fff1a5b
3fffff00: 000003fd 4021d2e7 00000378 000003ff
3fffff10: 00001000 4021d37d 3fff2564 000003ff
3fffff20: 000003fd 60000600 003fd000 3fff2564
3fffff30: ffffffff00 55aa55aa 00000312 0000001c
3fffff40: 0000001c 0000008a 0000006d 000003ff
3fffff50: 4021d224 3ffecf90 00000000 3ffed0c0
3fffff60: 00000001 4021c2e9 00000003 3fff1238
3fffff70: 4021c071 3ffecf84 3ffecf30 0026a2b0
3fffff80: 4021c0b6 3fffdab0 00000000 3fffdcb0
3fffff90: 3ffecf40 3fffdab0 00000000 3fffdcc0
3fffffa0: 40000f49 40000f49 3fffdab0 40000f49
<<<stack<<<
```

El primer número tras `Exception` devuelve la causa del reset. Puede encontrar una lista completa de las causas [aquí](#), los valores hexadecimales posteriores son el volcado de la pila.

### 13.1.1 Decodificado

Es posible decodificar la pila a información legible. Mas información vea la herramienta [Decodificador de excepciones ESP](#).

Arduino File Edit Sketch Tools Help

ESP\_RF12B\_RCV

```

1 #include <ESP8266WiFi.h>
2 #include <ArduinoOTA.h>
3 #include <SPI.h>
4 #include <Wire.h>
5 #include <SSD1306.h>
6 #include <RFM12B_ESP.h>
7 #include "OneWire.h"
8
9 #define ENABLE_SERIAL_DEBUG
10 #define RFM12B_NODE_ID 1
11 #define RFM12B_NETWORK_ID 100
12
13 const char* ssid = "nbis-test";
14 const char* password = "1234567890";
15
16 //ADC_MODE(ADC_TOUT);
17 SSD1306 display;
18 RFM12B radio;
19 OneWire ds(16);
20
21
22 uint8_t ds_addr[8];
23 uint8_t ds_data[12];
24 typedef enum { DS_IDLE, DS_START, DS_...
25 ds_state_t ds_state = DS_IDLE;

```

Tools menu options:

- Auto Format ⌘T
- Archive Sketch
- Fix Encoding & Reload
- Serial Monitor ⌘M
- Serial Plotter ⌘L
- ESP Exception Decoder**
- ESP8266 Sketch Data Upload
- Board: "Generic ESP8266 Module" ▶
- Flash Mode: "QIO" ▶
- Flash Size: "4M (3M SPIFFS)" ▶
- Debug port: "Disabled" ▶
- Debug Level: "None" ▶
- Reset Method: "nodemcu" ▶
- Flash Frequency: "80MHz" ▶
- Upload Using: "Serial" ▶
- CPU Frequency: "160 MHz" ▶
- Upload Speed: "115200" ▶
- Port: "/dev/cu.usbserial-A50285BI" ▶
- Programmer: "AVRISP mkII" ▶
- Burn Bootloader

Decode Success

Using library SPI at version 1.0 in folder: /Users/fiteto/Desktop/ESP8266/Arduino-Main/build/macosx/work/Arduino.app/Contents/Resources/Java/libraries/SPI

Using library Wire at version 1.0 in folder: /Users/fiteto/Desktop/ESP8266/Arduino-Main/build/macosx/work/Arduino.app/Contents/Resources/Java/libraries/Wire

Using library SSD1306 in folder: /Users/fiteto/Documents/Arduino/libraries/SSD1306 (legacy)

Using library RFM12B\_ESP in folder: /Users/fiteto/Documents/Arduino/libraries/RFM12B\_ESP (legacy)

Using library OneWire in folder: /Users/fiteto/Documents/Arduino/libraries/OneWire (legacy)

Using library ESP8266mDNS in folder: /Users/fiteto/Desktop/ESP8266/Arduino-Main/build/macosx/work/Arduino.app/Contents/Resources/Java/libraries/ESP8266mDNS

Sketch uses 255,441 bytes (24%) of program storage space. Maximum is 1,044,464 bytes.  
Global variables use 34,773 bytes (42%) of dynamic memory, leaving 47,147 bytes for local variables. Maximum is 81,920 bytes.

2 Generic ESP8266 Module, Serial, 160 MHz, 80MHz, QIO, 115200, 4M (3M SPIFFS), nodemcu, Disabled, None on /dev/cu.usbserial-A50285BI



### 14.1 Que descargar

- IDE Arduino
- IDE Eclipse para desarrolladores C/C++
- Java

### 14.2 Configurar Arduino

Ver el documento

### 14.3 Configurar Eclipse

- Paso 1
- Paso 2
- Ve a Window → preferences → Arduino
- Añade en «Private hardware path» una nueva línea a la ruta de Arduino ESP8266

Ejemplo de Ruta privada de hardware (Private hardware path)

```
Windows: C:\Users\[username]\AppData\Roaming\Arduino15\packages\esp8266\hardware  
Linux: /home/[username]/.arduino15/packages/esp8266/hardware
```

## 14.4 Eclipse no compila

Si Eclipse no encuentra la ruta al compilador añade a «platform.txt» tras:

```
version=1.6.4
```

esto:

```
runtime.tools.xtensa-lx106-elf-gcc.path={runtime.platform.path}/../../../../tools/xtensa-  
↳lx106-elf-gcc/1.20.0-26-gb404fb9  
runtime.tools.esptool.path={runtime.platform.path}/../../../../tools/esptool/0.4.4
```

Notas:

- La ruta puede cambiar, comprueba la versión actual.
- Cada actualización del IDE Arduino eliminará la solución.
- Puede no necesitarse en el futuro si el Plugin Eclipse se actualiza.