Libraries

WiFi (ESP8266WiFi library)

ESP8266WiFi library has been developed basing on ESP8266 SDK, using naming convention and overall functionality philosophy of the Arduino WiFi Shield library. Over time the wealth Wi-Fi features ported from ESP8266 SDK to this library outgrew the APIs of WiFi Shield library and it became apparent that we need to provide separate documentation on what is new and extra.

ESP8266WiFi library documentation

Ticker

Library for calling functions repeatedly with a certain period. Three examples included.

It is currently not recommended to do blocking IO operations (network, serial, file) from Ticker callback functions. Instead, set a flag inside the ticker callback and check for that flag inside the loop function.

Here is library to simplificate `Ticker` usage and avoid WDT reset: TickerScheduler

EEPROM

This is a bit different from standard EEPROM class. You need to call `EEPROM.begin(size)` before you start reading or writing, size being the number of bytes you want to use. Size can be anywhere between 4 and 4096 bytes.

`EEPROM.write` does not write to flash immediately, instead you must call `EEPROM.commit()` whenever you wish to save changes to flash. `EEPROM.end()` will also commit, and will release the RAM copy of EEPROM contents.

EEPROM library uses one sector of flash located just after the embedded filesystem.

Three examples included.

Note that the sector needs to be re-flashed every time the changed EEPROM data needs to be saved, thus will wear out the flash memory very quickly even if small amounts of data are written. Consider using one of the EEPROM libraries mentioned down below.

I2C (Wire library)

Wire library currently supports master mode up to approximately 450KHz. Before using I2C, pins for SDA and SCL need to be set by calling `Wire.begin(int sda, int scl)`, i.e. `Wire.begin(0, 2)` on ESP-01, else they default to pins 4(SDA) and 5(SCL).

SPI

SPI library supports the entire Arduino SPI API including transactions, including setting phase (CPHA). Setting the Clock polarity (CPOL) is not supported, yet (SPI_MODE2 and SPI_MODE3 not working).

The usual SPI pins are:

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

There's an extended mode where you can swap the normal pins to the SPI0 hardware pins. This is enabled by calling `SPI.pins(6, 7, 8, 0)` before the call to `SPI.begin()`. The pins would change to:

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

This mode shares the SPI pins with the controller that reads the program code from flash and is controlled by a hardware arbiter (the flash has always higher priority). For this mode the CS will be controlled by hardware as you can't handle the CS line with a GPIO, you never actually know when the arbiter is going to grant you access to the bus so you must let it handle CS automatically.

SoftwareSerial

An ESP8266 port of SoftwareSerial library done by Peter Lerup (@plerup) supports baud rate up to 115200 and multiples SoftwareSerial instances. See https://github.com/plerup/espsoftwareserial if you want to suggest an improvement or open an

issue related to SoftwareSerial.

ESP-specific APIs

Some ESP-specific APIs related to deep sleep, RTC and flash memories are available in the `ESP` object.

`ESP.deepSleep(microseconds, mode)` will put the chip into deep sleep. `mode` is one of `WAKE_RF_DEFAULT`, `WAKE_RFCAL`, `WAKE_NO_RFCAL`, `WAKE_RF_DISABLED`. (GPIO16 needs to be tied to RST to wake from deepSleep.) The chip can sleep for at most `ESP.deepSleepMax()` microseconds. If you implement deep sleep with `WAKE_RF_DISABLED` and require WiFi functionality on wake up, you will need to implement an additional `WAKE_RF_DEFAULT` before WiFi functionality is available.

`ESP.deepSleepInstant(microseconds, mode)` works similarly to `ESP.deepSleep` but sleeps instantly without waiting for WiFi to shutdown.

`ESP.rtcUserMemoryWrite(offset, &data, sizeof(data))` and `ESP.rtcUserMemoryRead(offset, &data, sizeof(data))` allow data to be stored in and retrieved from the RTC user memory of the chip respectively. `offset` is measured in blocks of 4 bytes and can range from 0 to 127 blocks (total size of RTC memory is 512 bytes). `data` should be 4-byte aligned. The stored data can be retained between deep sleep cycles, but might be lost after power cycling the chip. Data stored in the first 32 blocks will be lost after performing an OTA update, because they are used by the Core internals.

`ESP.restart()` restarts the CPU.

`ESP.getResetReason()` returns a String containing the last reset reason in human readable format.

`ESP.getFreeHeap()` returns the free heap size.

`ESP.getHeapFragmentation()` returns the fragmentation metric (0% is clean, more than ~50% is not harmless)

`ESP.getMaxFreeBlockSize()` returns the largest contiguous free RAM block in the heap, useful for checking heap fragmentation. **NOTE:** Maximum `malloc()` -able block will be smaller due to memory manager overheads.

`ESP.getChipId()` returns the ESP8266 chip ID as a 32-bit integer.

`ESP.getCoreVersion()` returns a String containing the core version.

`ESP.getSdkVersion()` returns the SDK version as a char.

`ESP.getCpuFreqMHz()` returns the CPU frequency in MHz as an unsigned 8-bit integer.

`ESP.getSketchSize()` returns the size of the current sketch as an unsigned 32-bit integer.

`ESP.getFreeSketchSpace()` returns the free sketch space as an unsigned 32-bit integer.

`ESP.getSketchMD5()` returns a lowercase String containing the MD5 of the current sketch.

`ESP.getFlashChipId()` returns the flash chip ID as a 32-bit integer.

`ESP.getFlashChipSize()` returns the flash chip size, in bytes, as seen by the SDK (may be less than actual size).

`ESP.getFlashChipRealSize()` returns the real chip size, in bytes, based on the flash chip ID.

`ESP.getFlashChipSpeed(void)` returns the flash chip frequency, in Hz.

`ESP.getCycleCount()` returns the cpu instruction cycle count since start as an unsigned 32-bit. This is useful for accurate timing of very short actions like bit banging.

`ESP.random()` should be used to generate true random numbers on the ESP. Returns an unsigned 32-bit integer with the random number. An alternate version is also available that fills an array of arbitrary length. Note that it seems as though the WiFi needs to be enabled to generate entropy for the random numbers, otherwise pseudo-random numbers are used.

`ESP.checkFlashCRC()` calculates the CRC of the program memory (not including any filesystems) and compares it to the one embedded in the image. If this call returns `false` then the flash has been corrupted. At that point, you may want to consider trying to send a MQTT message, to start a re-download of the application, blink a LED in an *SOS* pattern, etc. However, since the flash is known corrupted at this point there is no guarantee the app will be able to perform any of these operations, so in safety critical deployments an immediate shutdown to a fail-safe mode may be indicated.

`ESP.getVcc()` may be used to measure supply voltage. ESP needs to reconfigure the ADC at startup in order for this feature to be available. Add the following line to the top of your sketch to use `getVcc`:

```
ADC_MODE(ADC_VCC);
```

TOUT pin has to be disconnected in this mode.

Note that by default ADC is configured to read from TOUT pin using `analogRead(A0)`, and `ESP.getVCC()` is not available.

mDNS and DNS-SD responder (ESP8266mDNS library)

Allows the sketch to respond to multicast DNS queries for domain names like “foo.local”, and DNS-SD (service discovery) queries. See attached example for details.

SSDP responder (ESP8266SSDP)

SSDP is another service discovery protocol, supported on Windows out of the box. See attached example for reference.

DNS server (DNSServer library)

Implements a simple DNS server that can be used in both STA and AP modes. The DNS server currently supports only one domain (for all other domains it will reply with NXDOMAIN or custom status code). With it, clients can open a web server running on ESP8266 using a domain name, not an IP address.

Servo

This library exposes the ability to control RC (hobby) servo motors. It will support up to 24 servos on any available output pin. By default the first 12 servos will use Timer0 and currently this will not interfere with any other support. Servo counts above 12 will use Timer1 and features that use it will be affected. While many RC servo motors will accept the 3.3V IO data pin from a ESP8266, most will not be able to run off 3.3v and will require another power source that matches their specifications. Make sure to connect the grounds between the ESP8266 and the servo motor power supply.

Other libraries (not included with the IDE)

Libraries that don’t rely on low-level access to AVR registers should work well. Here are a few libraries that were verified to work:

- Adafruit_ILI9341 - Port of the Adafruit ILI9341 for the ESP8266
- arduinoVNC - VNC Client for Arduino
- arduinoWebSockets - WebSocket Server and Client compatible with ESP8266 (RFC6455)
- aREST - REST API handler library.
- Blynk - easy IoT framework for Makers (check out the Kickstarter page).
- DallasTemperature

- DHT-sensor-library - Arduino library for the DHT11/DHT22 temperature and humidity sensors. Download latest v1.1.1 library and no changes are necessary. Older versions should initialize DHT as follows: `DHT dht(DHTPIN, DHTTYPE, 15)`
- DimSwitch - Control electronic dimmable ballasts for fluorescent light tubes remotely as if using a wall switch.
- Encoder - Arduino library for rotary encoders. Version 1.4 supports ESP8266.
- esp8266_mdns - mDNS queries and responses on esp8266. Or to describe it another way: An mDNS Client or Bonjour Client library for the esp8266.
- ESP-NOW - Wrapper lib for ESP-NOW (See #2227)
- ESPAsyncTCP - Asynchronous TCP Library for ESP8266 and ESP32/31B
- ESPAsyncWebServer - Asynchronous Web Server Library for ESP8266 and ESP32/31B
- Homie for ESP8266 - Arduino framework for ESP8266 implementing Homie, an MQTT convention for the IoT.
- NeoPixel - Adafruit's NeoPixel library, now with support for the ESP8266 (use version 1.0.2 or higher from Arduino's library manager).
- NeoPixelBus - Arduino NeoPixel library compatible with ESP8266. Use the "DmaDriven" or "UartDriven" branches for ESP8266. Includes HSL color support and more.
- PubSubClient - MQTT library by @Imroy.
- RTC - Arduino Library for Ds1307 & Ds3231 compatible with ESP8266.
- Souliss, Smart Home - Framework for Smart Home based on Arduino, Android and openHAB.
- ST7735 - Adafruit's ST7735 library modified to be compatible with ESP8266. Just make sure to modify the pins in the examples as they are still AVR specific.
- Task - Arduino Nonpreemptive multitasking library. While similar to the included Ticker library in the functionality provided, this library was meant for cross Arduino compatibility.
- TickerScheduler - Library provides simple scheduler for `Ticker` to avoid WDT reset
- Teleinfo - Generic French Power Meter library to read Teleinfo energy monitoring data such as consuption, contract, power, period, ... This library is cross platform, ESP8266, Arduino, Particle, and simple C++. French dedicated post on author's blog and all related information about Teleinfo also available.
- UTFT-ESP8266 - UTFT display library with support for ESP8266. Only serial interface (SPI) displays are supported for now (no 8-bit parallel mode, etc). Also includes support for the hardware SPI controller of the ESP8266.
- WiFiManager - WiFi Connection manager with web captive portal. If it can't connect, it starts AP mode and a configuration portal so you can choose and enter WiFi credentials.
- OneWire - Library for Dallas/Maxim 1-Wire Chips.
- Adafruit-PCD8544-Nokia-5110-LCD-Library - Port of the Adafruit PCD8544 - library for the ESP8266.
- PCF8574_ESP - A very simplistic library for using the PCF857//PCF8574A I2C 8-pin GPIO-expander.
- Dot Matrix Display Library 2 - Freetronics DMD & Generic 16 x 32 P10 style Dot Matrix Display Library

- SdFat-beta - SD-card library with support for long filenames, software- and hardware-based SPI and lots more.
- FastLED - a library for easily & efficiently controlling a wide variety of LED chipsets, like the Neopixel (WS2812B), DotStar, LPD8806 and many more. Includes fading, gradient, color conversion functions.
- OLED - a library for controlling I2C connected OLED displays. Tested with 0.96 inch OLED graphics display.
- MFRC522 - A library for using the Mifare RC522 RFID-tag reader/writer.
- Ping - lets the ESP8266 ping a remote machine.
- AsyncPing - fully asynchronous Ping library (have full ping statistic and hardware MAC address).
- ESP_EEPROM - This library writes a new copy of your data when you save (commit) it and keeps track of where in the sector the most recent copy is kept using a bitmap. The flash sector only needs to be erased when there is no more space for copies in the flash sector.
- EEPROM Rotate - Instead of using a single sector to persist the data from the emulated EEPROM, this library uses a number of sectors to do so: a sector pool.