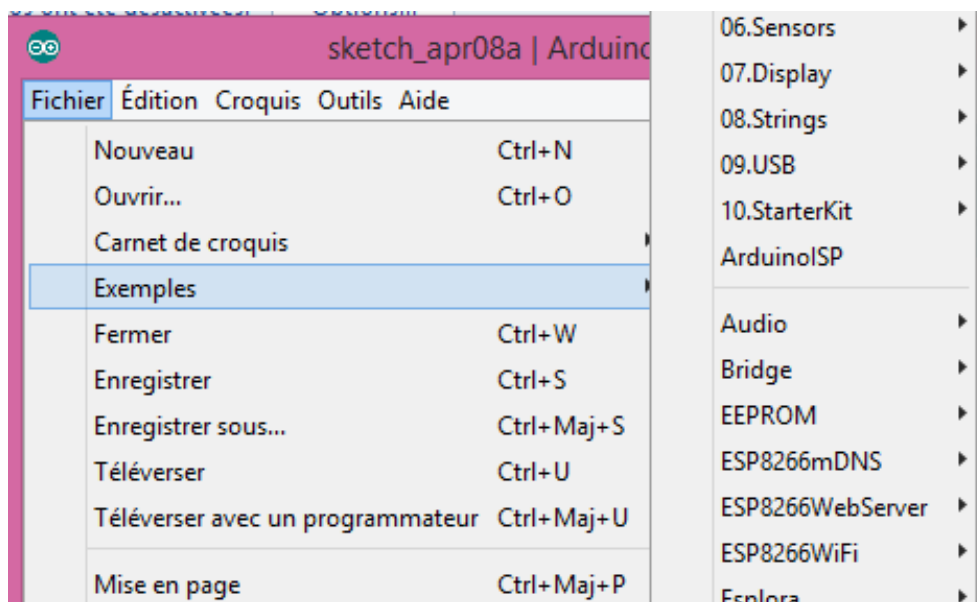
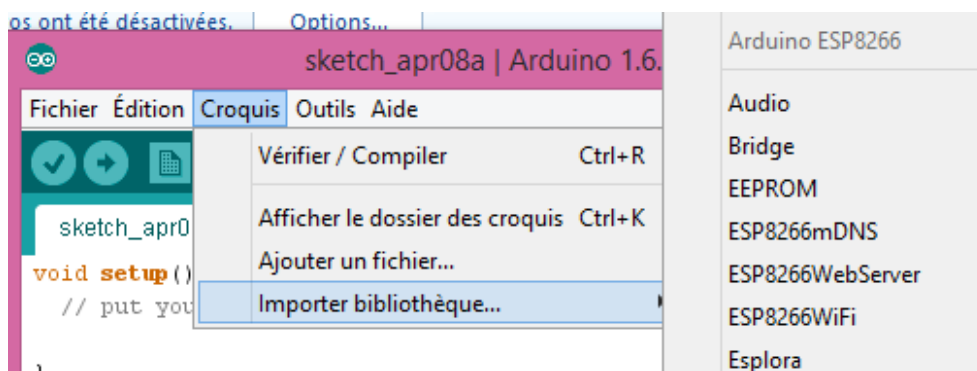
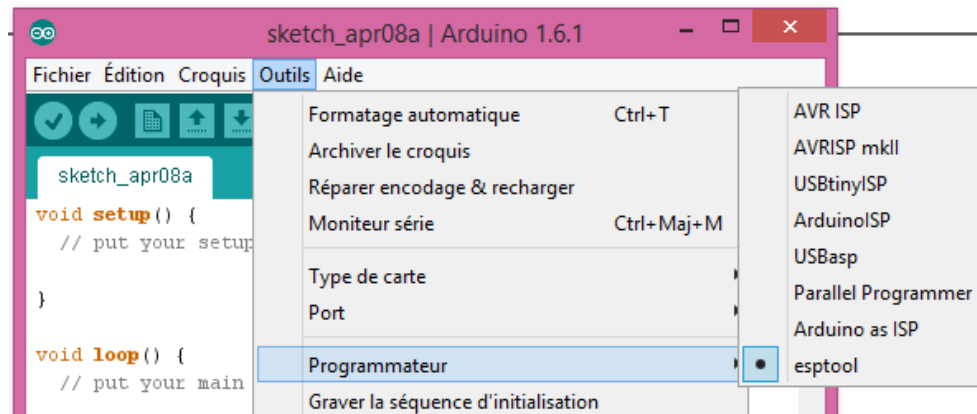
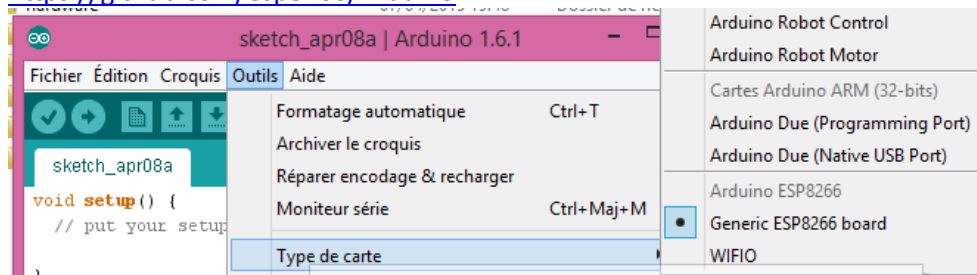


IDE Arduino ESP8266

<https://github.com/esp8266/Arduino>



Arduino-compatible IDE with ESP8266 support

This project brings support for ESP8266 chip to the Arduino environment. ESP8266WiFi library bundled with this project has the same interface as the WiFi Shield library, making it easy to re-use existing code and libraries.

Downloads

OS	Build status	Latest release
Linux		arduino-1.6.1-linux64.tar.xz
Windows		arduino-1.6.1-p1-windows.zip
OS X		arduino-1.6.1-macosx-java-latest-signed.zip

Building from source

```
$ git clone https://github.com/esp8266/Arduino.git
$ cd Arduino/build
$ ant dist
```

Supported boards

- [Wifiio](#)
- Generic esp8266 modules (without auto-reset support)

Things that work

Basic Wiring functions

`pinMode`, `digitalRead`, `digitalWrite` work as usual.

Pin numbers correspond directly to the esp8266 GPIO pin numbers. To read GPIO2, call `digitalRead(2)`;

GPIO0-GPIO15 can be `INPUT`, `OUTPUT`, `INPUT_PULLUP`, and `OUTPUT_OPEN_DRAIN`. GPIO16 can be `INPUT` or `OUTPUT`.

`analogRead(0)` reads the value of the ADC channel connected to the TOUT pin.

Pin interrupts are supported through `attachInterrupt`, `detachInterrupt` functions. Interrupts may be attached to any GPIO pin, except GPIO16. Standard Arduino interrupt types are supported: `CHANGE`, `RISING`, `FALLING`.

Timing and delays

`millis` and `micros` return the number of milliseconds and microseconds elapsed after reset, respectively.

`delay` pauses the sketch for a given number of milliseconds and allows WiFi and TCP/IP tasks to run.

`delayMicroseconds` pauses for a given number of microseconds.

Remember that there is a lot of code that needs to run on the chip besides the sketch when WiFi is connected. WiFi and TCP/IP libraries get a chance to handle any pending events each time the `loop()` function completes, OR when `delay(...)` is called. If you have a loop somewhere in your sketch that takes a lot of time (>50ms) without calling `delay()`, you might consider adding a call to `delay` function to keep the WiFi stack running smoothly.

There is also a `yield()` function which is equivalent to `delay(0)`. The `delayMicroseconds` function, on the other hand, does not yield to other tasks, so using it for delays more than 20 milliseconds is not recommended.

Serial

Serial object works much the same way as on a regular Arduino. Apart from hardware FIFO (128 bytes for TX and RX) HardwareSerial has additional 256-byte TX and RX buffers. Both transmit and receive is interrupt-driven. Write and read functions only block the sketch execution when the respective FIFO/buffers are full/empty.

Only 8n1 mode is supported right now.

By default the diagnostic output from WiFi libraries is disabled when you call `Serial.begin()`. To enable debug output again, call `Serial.setDebugOutput(true)`;

WiFi(ESP8266WiFi library)

This is mostly similar to WiFi shield library. Differences include:

- `WiFi.mode(m)`: set mode to `WIFI_AP`, `WIFI_STA`, or `WIFI_AP_STA`.
- call `WiFi.softAP(ssid)` to set up an open network
- call `WiFi.softAP(ssid, passphrase)` to set up a WPA2-PSK network
- `WiFi.macAddress(mac)` is for STA, `WiFi.softAPmacAddress(mac)` is for AP.
- `WiFi.localIP()` is for STA, `WiFi.softAPIP()` is for AP.
- `WiFi.RSSI()` doesn't work
- `WiFi.printDiag(Serial)`; will print out some diagnostic info

WiFiServer, WiFiClient, and WiFiUDP behave mostly the same way as with WiFi shield library. Three samples are provided for this library.

Ticker

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

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.

Three examples included.

I2C (Wire library)

Only master mode works, and `Wire.setClock` has not been verified to give exactly correct frequency. Before using I2C, pins for SDA and SCL need to be set by calling `Wire.pins(int sda, int scl)`, i.e. `Wire.pins(0, 2)`; on ESP-01.

SPI

An initial SPI support for the HSPI interface (GPIO12-15) was implemented by [Sermus](#). The implementation supports the entire Arduino SPI API including transactions, except setting phase and polarity as it's unclear how to set them in ESP8266 yet.

OneWire (from https://www.pjrc.com/teensy/td_libs_OneWire.html)

Library was adapted to work with ESP8266 by including register definitions into `OneWire.h` Note that if you have OneWire library in your `Arduino/libraries` folder, it will be used instead of the one that comes with the Arduino IDE (this one).

mDNS responder (ESP8266mDNS library)

Allows the sketch to respond to multicast DNS queries for domain names like "foo.local". See attached example and library README file for details.

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:

- [aREST](#) REST API handler library.
- [PubSubClient](#) MQTT library. Use this [sample](#) to get started.
- [DHT11](#) - initialize DHT as follows: `DHT dht(DHTPIN, DHTTYPE, 15);`
- [DallasTemperature](#)

Upload via serial port

Pick the correct serial port. You need to put ESP8266 into bootloader mode before uploading code (pull GPIO0 low and toggle power).

Things not done yet

- `analogWrite` (PWM). ESP8266 has only one hardware PWM source. It is not yet clear how to use it with `analogWrite` API. Software PWM is also an option, but apparently it causes issues with WiFi connectivity.
- `pulseIn`
- I2C slave mode
- Serial modes other than 8n1
- WiFi.RSSI. SDK doesn't seem to have an API to get RSSI for the current network. So far the only way to obtain RSSI is to disconnect, perform a scan, and get the RSSI value from there.
- Upload sketches via WiFi. Conceptually and technically simple, but need to figure out how to provide the best UX for this feature.
- Samples for all the libraries

Issues and support

Forum: <http://www.esp8266.com/arduino>

Submit issues on Github: <https://github.com/esp8266/Arduino/issues>

License and credits

Arduino IDE is based on Wiring and Processing. It is developed and maintained by the Arduino team. The IDE is licensed under GPL, and the core libraries are licensed under LGPL.

This build includes an xtensa gcc toolchain, which is also under GPL.

Espressif SDK included in this build is under Espressif Public License.

Esptool written by Christian Klippel is licensed under GPLv2, currently maintained by Ivan Grokhotkov:
<https://github.com/igrr/esptool-ck>.

ESP8266 core support, ESP8266WiFi, Ticker, ESP8266WebServer libraries were written by Ivan Grokhotkov,
ivan@esp8266.com.