# ESP8266 Workshop 1

Andy Gelme (@geekscape)

http://geekscape.org

Wednesday 2015-03-11

# Workshop overview

- ESP8266 introduction: hardware and software
- ESP-01 and ESP-12 set-up for development
- Flashing firmware using esptool
- NodeMCU and Lua introduction
- ESPlorer IDE
- Using a button and LED(s)
- Setting up as a Wi-Fi client
- Using UDP to send and receive messages
- Creating an MQTT client

# ESP8266 Hardware

- 3.3V device, operating current ~ 215 mA
- CPU: Tensilca Xtensa LX3: 32-bit, 72 MHz (?)
- ESP8266 SOC: Expressif
- RAM 32Kb, DRAM 80Kb, Flash 200Kb (for SDK)
- Wi-Fi 802.11 b/g/n 2.4 GHz radio (station or AP)
- Timers, deep sleep mode, JTAG debugging
- Peripherals …
  - GPIO (upto 16), PWM (3), ADC (one)
  - UART, I2C, SPI

# ESP8266 Memory Map

- More complicated than your AVR ATMega / Arduino
- SRAM 32Kb
  - NodeMCU heap memory available: print(node.heap())
- SPI Flash ROM layout: Since SDK 0.8 …
  - 00000h 248k app.v6.flash.bin User application
    3E000h 8k master_device_key.bin OTA device key
    40000h 240k app.v6.irom0text.bin SDK libraries
    7C000h 8k esp_init_data_default.bin Default configuration
    7E000h 8k blank.bin Filled with FFh. May be WiFi configuration
- https://github.com/esp8266/esp8266-wiki/wiki/Memory-Map

# ESP8266 Software stack

- Can build using FreeRTOS: http://www.freertos.org
  - https://github.com/espressif/esp_iot_rtos_sdk
- ESP8266 SDK: hardware access and C library
- Application layer, either …
  - AT commands (standard on ESP-01)
  - IoT demo (standard on ESP-12)
  - NodeMCU Lua interpreter
  - Your own application written in C

# ESP-xx modules

- Variety of form factors, ESP-01 through ESP-12 …
  - Antenna: None, Chip, PCB or UL connector
  - Which pins broken out and layout (2 mm pitch)
- Power: 3.3V typical, 3.0V min, 3.6V max
- Chip enable pin (CH_PD): Hold high
  - ESP-12 dev board does this for you
- GPIO0: Hold low on power-up to enter boot loader
  - Flash firmware
  - After power-up, can use GPIO0 as button input

# ESP-01 set-up

- PCB antenna
- 8 pins including UART Rx/Tx, GPIO (2), Reset
- No power regulator, use only 3.0V to 3.6V
- Need a means of holding GPIO0 low for firmware flash
- Standard firmware: AT commands

- Cable or use JohnS ESPkit-01 …
  - ESP-01 USB serial adaptor
    1 RXD <—— TXD
    2 VCC —— VCC 3.3V
    3 GPIO0 —— Connect to ground during power-up to flash firmware
    4 RESET —— VCC 3.3V
    5 GPIO2
    6 CH_PD —— VCC 3.3V
    7 GND —— GND
    8 TXD ——> RXD

# ESP-12 dev board set-up

- PCB antenna
- 16 pins inc. UART Rx/Tx, GPIO (9), ADC, Reset
  - RGB LED GPIOs: Red = 8,  Green = 6,  Blue = 7
- Uses 3x AA batteries with on-board 3.3v regulator
- Provides GPIO0 header for firmware flash
- Standard firmware: IoT demo
- Cable: Note ESP-12 dev board labels are confusing !
  - ESP-12 USB serial adaptor
    RXD <—— RXD
    GND —— GND
    TXD ——> TXD

# esptool overview

- Cross-platform Python, requires pySerial
  - https://github.com/themadinventor/esptool
- Various functions …
  - Flash firmware
    - Create bootable image from binary blobs
  - Read MAC address
  - Read chip identification
  - Read / write memory
- Hold GPIO0 low and power cycle ESP8266

# esptool flash firmware

- Flash standard NodeMCU pre-built image …
  - Download image …
  - https://github.com/nodemcu/nodemcu-firmware
  - esptool -p SERIAL_PORT write_flash 0x00000 nodemcu_20150212.bin

- Flash NodeMCU image with WS2812B support
  - https://github.com/geekscape/esp8266_nodemcu_examples/firmware
  - esptool -p SERIAL_PORT write_flash 0x00000 nodemcu_dev_0x00000.bin 0x10000 nodemcu_dev_0x10000.bin

# NodeMCU firmware

- eLua interpreter including ESP8266 SDK API
- NodeMCU API documentation …
  - https://github.com/nodemcu/nodemcu-firmware/wiki/nodemcu_api_en
- Serial terminal connection at 9,600 baud
  - Command line prompt is “> “
  - Enter valid Lua statements
- Starts with around 23Kb heap available
  - Need to take care with your memory consumption
- Further reading: nodemcu.com/index_en.html

# NodeMCU API

- node.restart(), heap(), compile()
- file.format(), list(), remove(), open(), read(), write()
- wifi.sta, wifi.ap sub-modules
- net module: TCP and UDP client and server
- mqtt module: Message Queue Telemetry Transport
- timer module, tmr.alarm(), tmr.stop(), tmr.time()
- gpio.mode(), read(), write(), trig()
- pwm, adc, i2c, spi, one-wire modules
- bit module: bit-wise logical operations

# Lua introduction - I

- http://www.lua.org/start.html
- print(1 + 2);  print("hello");  print(node.heap())
- Variables: boolean, numbers, strings, tables
  - Global by default, but can use "local" keyword
  - Variable type declarations are not required
  - colour = {};  colour["red"] = 1;  colour["blue"] = 2
- Functions are first-class values
  - function *name*(*parameters*)
    return *value*  - - Can return multiple values
    end

# Lua introduction - 2

- Control-flow …
  - if *condition* then
    print(true)
    else
    print(false)
    end
  - for *variable* = *first, last, delta* do
    end
  - while *condition* do
    end

# Lua introduction - 3

- NodeMCU after boot will try to run “init.lua”
- Compile using node.compile(“file.lua”)
  - Byte-code written to “file.lc”
  - Save memory
- Modules …
  - moduleA = require(“module.lua”)
  - local module = {}
    function module.name()
    end
    return module

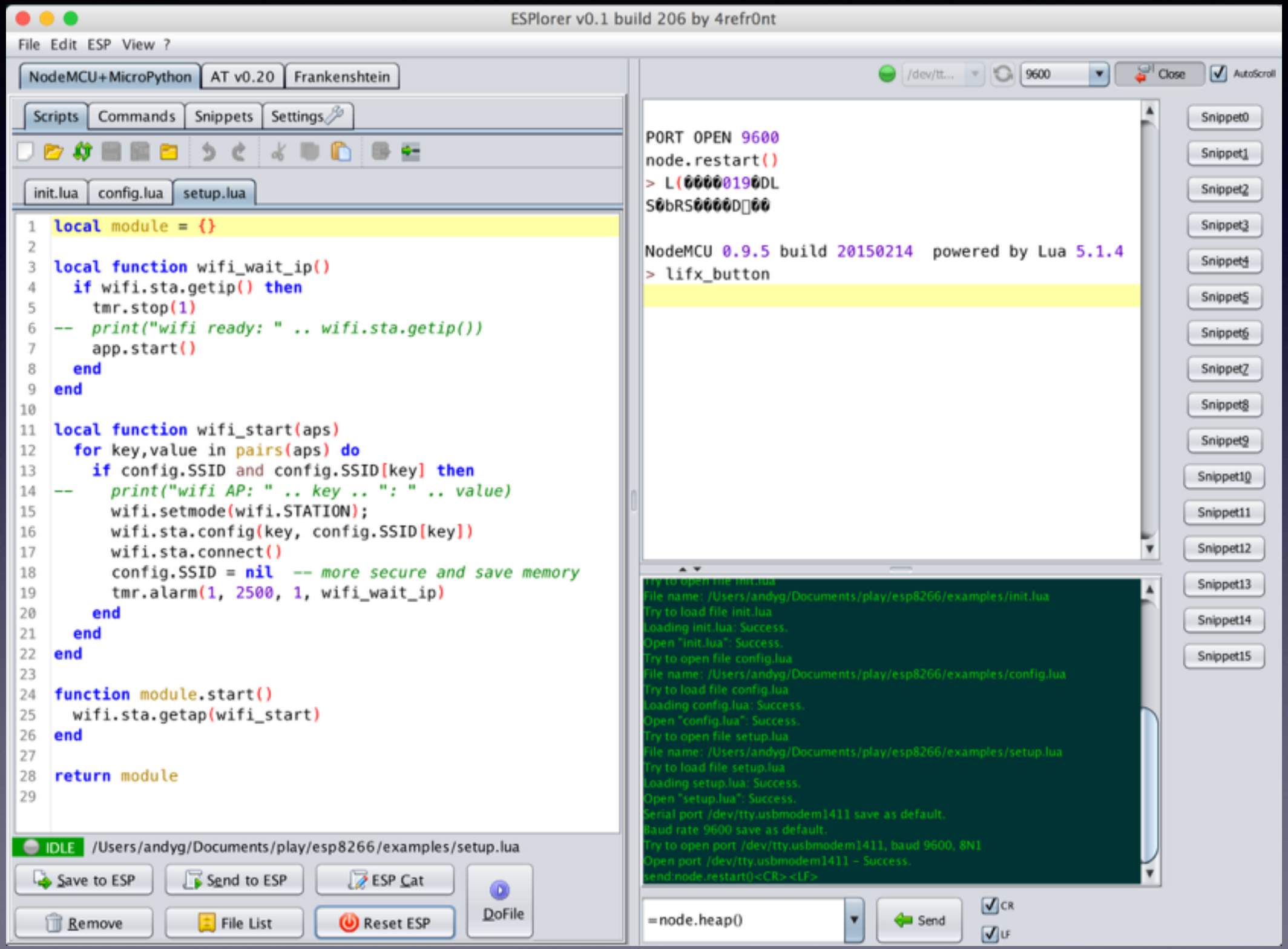
# NodeMCU example

- **You can't just type … print(file.list()) !**
- **You need to type …**
- **for n,s in pairs(file.list()) do print(n.." size: "..s) end**
- **file.list() … returns a table**
- **pairs() … returns an iterator**
- **for n,s in … loops through each iterator pair**
  - **n = file name, s = file size**

# ESPlorer IDE

- Simplify process of managing files and running scripts on NodeMCU or MicroPython
- Also short-cuts for AT commands firmware
- http://esp8266.ru/esplorer
- Open source …
  - https://github.com/4refr0nt/ESPlorer
- Start via command line … java -jar ESPlorer.jar
  - Scans for available serial ports
  - Provides Graphical User Interface for development

# ESPlorer IDE



# Button input

- Connect button between GPIO0 and Ground
- Example code …
  - https://github.com/geekscape/esp8266_nodemcu_examples/blob/master/examples/button.lua
    - Load into ESPlorer IDE: File -> Open
    - Save to ESP8266: [Save to ESP]
    - Run script: [Do File]
- ESP-12 bonus: Try to use LDR and adc.read()

# Button input - FAIL

- Our first attempt is likely to look like this …

```
PIN_BUTTON = 3    -- GPIO0
gpio.mode(PIN_BUTTON, gpio.INPUT, gpio.PULLUP)
```

```
while true do
  if gpio.read(PIN_BUTTON) == 0 then
    print("BUTTON PRESSED")
    tmr.delay(20000)  - - microseconds
  end
end
```

# Button input

- … but previous example, after a short while it fails :(
- More reliable to use timers, rather than delays
- Also, code is much more modular and re-usable
- Try … tmr.alarm(id, interval, repeat_flag, function)
  - id: timer number 0 to 6
  - interval: alarm time in milliseconds
  - repeat_flag: 0 = once, 1 = repeat until tmr.stop()
  - function: Lua function to invoke
- The following code is more reliable …

```
PIN_BUTTON = 3   -- GPIO0
TIME_ALARM = 25  -- 0.025 second, 40 Hz
gpio.mode(PIN_BUTTON, gpio.INPUT, gpio.PULLUP)
button_state = 1
button_time  = 0

function buttonHandler()
  button_state_new = gpio.read(PIN_BUTTON)
  if button_state == 1 and button_state_new == 0 then
    button_time = tmr.time()
    print("BUTTON PRESSED")
  elseif button_state == 0 and button_state_new == 1 then
    button_time_new = tmr.time()
    if tmr.time() > (button_time + 2) then
      tmr.stop(1)
      print("BUTTON DISABLED")
    end
  end
  button_state = button_state_new
end

tmr.alarm(1, TIME_ALARM, 1, buttonHandler)
```

# Regular LED output

- Connect LED and current limiting resistor between GPIO2 and Ground
- Example code …
- https://github.com/geekscape/esp8266_nodemcu_examples/blob/master/examples/led.lua
- Load into ESPlorer IDE: File -> Open
- Save to ESP8266: [Save to ESP]
- Run script: [Do File]
- Try … init.lua: dofile('led.lua')

# Regular LED output

```
LED_PIN    = 4          -- GPIO2
TIME_ALARM = 500  -- 0.5 second


gpio.mode(LED_PIN, gpio.OUTPUT)
```

```
led_state = gpio.LOW
function ledHandler()
  if led_state == gpio.LOW then
    led_state = gpio.HIGH
  else
    led_state = gpio.LOW
  end
  gpio.write(LED_PIN, led_state)
end

tmr.alarm(1, TIME_ALARM,  1, ledHandler)
```

# WS2812B LED output

- Original development by Markus Gritsch
  - http://www.esp8266.com/viewtopic.php?f=21&t=1143
- Isn't yet part of the NodeMCU master branch
- BRIGHT = 0.1;  ON = BRIGHT * 255
  LED_PIN = 4 -- GPIO2
  PIXELS = 8
  RED      = string.char( 0, ON,  0)
  GREEN = string.char(ON,  0,  0)
  BLUE    = string.char( 0,  0, ON)
  ws2812.write(LED_PIN, RED:rep(PIXELS))
  ws2812.write(LED_PIN, GREEN:rep(PIXELS))
  ws2812.write(LED_PIN, BLUE:rep(PIXELS))

# WS2812B LED example

- Connect WS2812B to GPIO2
- Example code …
  - https://github.com/geekscape/esp8266_nodemcu_examples/blob/master/examples/ws2812.lua
- Load into ESPlorer IDE: File -> Open
- Save to ESP8266: [Save to ESP]
- Run script: [Do File]
- Try … init.lua: dofile('ws2812.lua')

# Setting up Wi-Fi client

- Scan to get available Wi-Fi Access Points …
  - wifi.sta.getap(function)
  - Calls function when list of Access Points is ready
- Start Wi-Fi station (client) …
  - wifi.setmode(wifi.STATION);
  - wifi.sta.config(“ssid”, “passphrase”)
  - wifi.sta.connect()
- Returns your IP address when ready …
  - wifi.sta.getip()

- wifi.sta.getap(wifi_start)
- module.SSID = {}
  module.SSID["SSID1"] = "passphrase1"
  module.SSID["SSID2"] = "passphrase2"
- local function wifi_start(aps)
  for key,value in pairs(aps) do
  if config.SSID and config.SSID[key] then
  wifi.setmode(wifi.STATION);
  wifi.sta.config(key, config.SSID[key])
  wifi.sta.connect()
  tmr.alarm(1, 2500, 1, wifi_wait_ip)
  end
  end
  end
- local function wifi_wait_ip()
  if wifi.sta.getip() then
  tmr.stop(1)
  end
  end

# Wi-Fi client example - I

- Modular, flexible, timer-based application skeleton
- Example code …
  - https://github.com/geekscape/esp8266_nodemcu_examples/tree/master/skeleton

- Load into ESPlorer IDE: File -> Open
- Save to ESP8266: [Save to ESP]
- Reset ESP8266
- print(wifi.sta.getip())

# Wi-Fi client example - 2

- init.lua
  - Called on boot
  - Minimal, can’t be compiled :(
- config.lua
  - Application configuration, e.g SSID and passphrases
- setup.lua
  - Sets up Wi-Fi depending upon your location
- application.lua
  - Your code, change filename as appropriate
- Use node.compile() on everything except init.lua

# UDP message transmit

- **socket = net.createConnection(net.UDP, 0)**
  **socket:connect(*UDP_PORT*, *UDP_HOST_STRING*)**
  **socket:send(*MESSAGE*)**
  **socket:close()**

- Example code …
  - https://github.com/geekscape/esp8266_nodemcu_examples/blob/master/applications/button_udp.lua
- Test on laptop with command line UDP server …
  - nc -l -u IP_ADDRESS 4000

# UDP message receive

- udp_server = net.createServer(net.UDP)
  udp_server:on("receive", function(srv, data)
    print("udp: " .. data)
  end)
- udp_server:close()
- Example code …
  - https://github.com/geekscape/esp8266_nodemcu_examples/blob/master/applications/ws2812_udp.lua
- Test on laptop with command line UDP client …
  - nc -u IP_ADDRESS 4000

# MQTT client

- MQTT server address “192.168.0.32”
- m:on("message", function(conn, topic, data)
  print(topic .. ":" )
  if data ~= nil then print(data) end
end)

```
m:connect(“192.168.0.32", 1883, 0, function(conn)
  print("connected")
end)
m:subscribe("/topic",0, function(conn)
  print("subscribe success”)
end)
m:publish("/topic","hello",0,0, function(conn)
  print("sent")
end)
m:close();
```

# ESP8266 / NodeMCU resources

- **ESP8266 general information**
  - https://nurdspace.nl/ESP8266
  - https://nurdspace.nl/images/e/e0/ESP8266_Specifications_English.pdf
- **NodeMCU API documentation**
  - https://github.com/nodemcu/nodemcu-firmware/wiki/nodemcu_api_en
- Lua programming language: http://www.lua.org
- ESPlorer IDE
  - http://esp8266.ru/esplorer
- esptool (cross-platform) for flashing firmware
  - https://github.com/themadinventor/esptool