esp8266 / Arduino

Branch: master ▾ Arduino / doc / ota_updates / Create new file Upload files Find file History

jonhp and devyte Update readme.rst (#4131) … Latest commit d112dfa on 19 Mar

..		
a-ota-external-serial-terminal-output-failed.png	Extended troubleshooting section	2 years ago
a-ota-external-serial-terminal-output.png	IDE 1.6.7 support for ArduinoOTA	2 years ago
a-ota-network-terminal.png	Classic OTA images deleted	2 years ago
a-ota-ota-port-selection.png	Changed ArduinoOTA pictures	2 years ago
a-ota-ota-upload-complete.png	Changed ArduinoOTA pictures	2 years ago
a-ota-ota-upload-configuration.png	Changed ArduinoOTA pictures	2 years ago
a-ota-python-configuration.png	Classic OTA images deleted	2 years ago
a-ota-serial-upload-configuration.png	Changed ArduinoOTA pictures	2 years ago
a-ota-sketch-selection.png	IDE 1.6.7 support for ArduinoOTA	2 years ago
a-ota-ssid-pass-entry.png	IDE 1.6.7 support for ArduinoOTA	2 years ago
a-ota-upload-complete-and-joined-wifi.png	Changed ArduinoOTA pictures	2 years ago
a-ota-upload-password-authenticating-ok.png	Protecting ArduinoOTA uploads with a password	2 years ago
a-ota-upload-password-passing-again-upload…	Protecting ArduinoOTA uploads with a password	2 years ago
a-ota-upload-password-passing-upload-ok.png	Protecting ArduinoOTA uploads with a password	2 years ago
a-ota-upload-password-prompt.png	Protecting ArduinoOTA uploads with a password	2 years ago
ota-web-browser-form-ok.png	Initial release of Web Browser section	3 years ago
ota-web-browser-form.png	Initial release of Web Browser section	3 years ago
ota-web-path-to-binary.png	Initial release of Web Browser section	3 years ago
ota-web-serial-monitor-ready.png	Initial release of Web Browser section	3 years ago
ota-web-serial-monitor-reboot.png	Initial release of Web Browser section	3 years ago
ota-web-show-verbose-compilation.png	Initial release of Web Browser section	3 years ago
readme.rst	Update readme.rst (#4131)	2 months ago
termite-configuration.png	Troubleshooting of OTA from IDE	2 years ago
update_memory_copy.png	doc better quality of update_memory_copy.png	3 years ago

readme.rst

OTA Updates

Introduction

OTA (Over the Air) update is the process of loading the firmware to ESP module using Wi-Fi connection rather than a serial port. Such functionality became extremely useful in case of limited or no physical access to the module.

OTA may be done using:

- Arduino IDE
- Web Browser
- HTTP Server

Arduino IDE option is intended primarily for software development phase. The two other options would be more useful after deployment, to provide module with application updates manually with a web browser, or automatically using a http server.

In any case, the first firmware upload has to be done over a serial port. If the OTA routines are correctly implemented in a sketch, then all subsequent uploads may be done over the air.

There is no imposed security on OTA process from being hacked. It is up to developer to ensure that updates are allowed only from legitimate / trusted sources. Once the update is complete, the module restarts, and the new code is executed. The developer should ensure that the application running on the module is shut down and restarted in a safe manner. Chapters below provide additional information regarding security and safety of OTA process.

Security

Module has to be exposed wirelessly to get it updated with a new sketch. That poses chances of module being violently hacked and loaded with some other code. To reduce likelihood of being hacked consider protecting your uploads with a password, selecting certain OTA port, etc.

Check functionality provided with ArduinoOTA library that may improve security:

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

Certain protection functionality is already built in and do not require any additional coding by developer. ArduinoOTA and espota.py use Digest-MD5 to authenticate upload. Integrity of transferred data is verified on ESP side using MD5 checksum.

Make your own risk analysis and depending on application decide what library functions to implement. If required, consider implementation of other means of protection from being hacked, e.g. exposing module for uploads only according to specific schedule, trigger OTA only be user pressing dedicated "Update" button wired to ESP, etc.

Safety

OTA process takes ESP's resources and bandwidth during upload. Then module is restarted and a new sketch executed. Analyse and test how it affects functionality of existing and new sketch.

If ESP is placed in remote location and controlling some equipment, you should put additional attention what happens if operation of this equipment is suddenly interrupted by update process. Therefore, decide how to put this equipment into safe state before starting the update. For instance, your module may be controlling a garden watering system in a sequence. If this sequence is not properly shut down and a water valve left open, your garden may be flooded.

The following functions are provided with ArduinoOTA library and intended to handle functionality of your application during specific stages of OTA, or on an OTA error:

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

Basic Requirements

Flash chip size should be able to hold the old sketch (currently running) and the new sketch (OTA) at the same time.

Keep in mind that the File system and EEPROM for example needs space too (one time) see :ref:`Flash layout`.

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

can be used for checking the free space for the new sketch.

For overview of memory layout, where new sketch is stored and how it is copied during OTA process, see Update process - memory view.

The following chapters provide more details and specific methods of doing OTA.

Arduino IDE

Uploading modules wirelessly from Arduino IDE is intended for the following typical scenarios: - during firmware development as a quicker alternative to loading over a serial, - for updating small quantity of modules, - only if modules are available on the same network as the computer with Arduino IDE.

Requirements

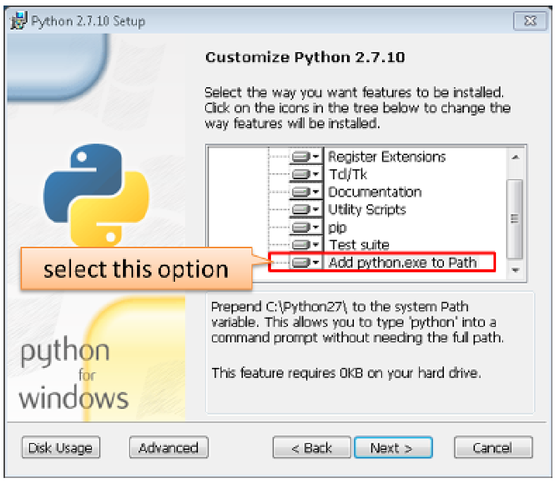
- The ESP and the computer must be connected to the same network.

Application Example

Instructions below show configuration of OTA on NodeMCU 1.0 (ESP-12E Module) board. You can use any other board assuming that it meets requirements described above. This instruction is valid for all operating systems supported by Arduino IDE. Screen captures have been made on Windows 7 and you may see small differences (like name of serial port), if you are using Linux and MacOS.

1. Before you begin, please make sure that you have the following s/w installed:
   - Arduino IDE 1.6.7 or newer - https://www.arduino.cc/en/Main/Software
   - esp8266/Arduino platform package 2.0.0 or newer - for instructions follow https://github.com/esp8266/Arduino#installing-with-boards-manager
   - Python 2.7 - https://www.python.org/

     Note: Windows users should select "Add python.exe to Path" (see below – this option is not selected by default).



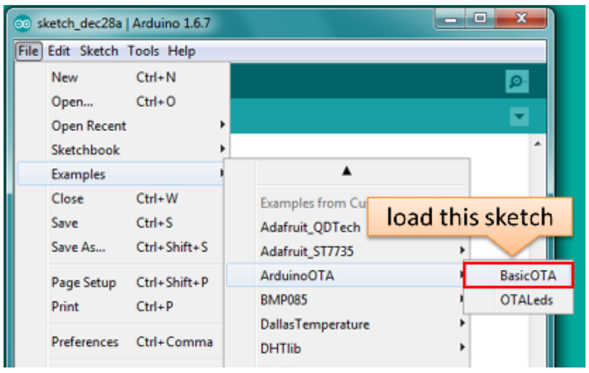
Customize Python 2.7.10

Select the way you want features to be installed. Click on the icons in the tree below to change the way features will be installed.

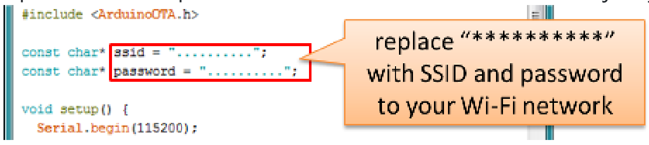
Prepend C:\Python27\ to the system Path variable. This allows you to type 'python' into a command prompt without needing the full path.

This feature requires 0KB on your hard drive.

2. Now prepare the sketch and configuration for the upload over a serial port.
   - Start Arduino IDE and load sketch BasicOTA.ino available under File > Examples > ArduinoOTA

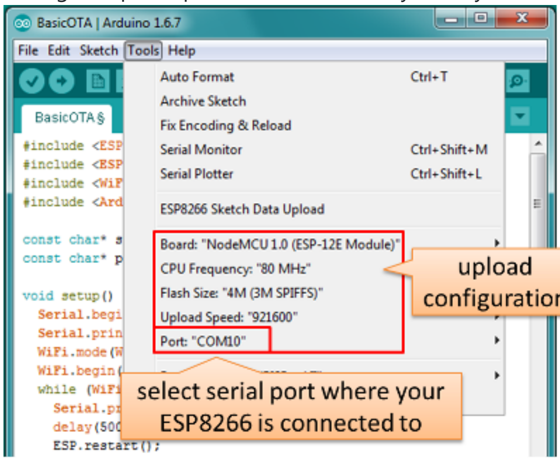


   - Update SSID and password in the sketch, so the module can join your Wi-Fi network



```
#include <ArduinoOTA.h>

const char* ssid = "..........";
const char* password = "..........";

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

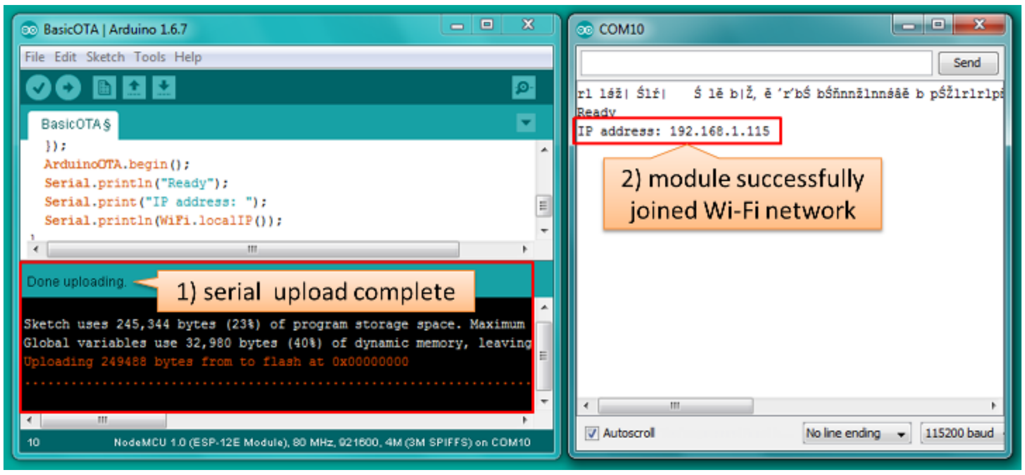
- Configure upload parameters as below (you may need to adjust configuration if you are using a different module):


BasicOTA | Arduino 1.6.7
File Edit Sketch Tools Help
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Serial Monitor Ctrl+Shift+M
Serial Plotter Ctrl+Shift+L
ESP8266 Sketch Data Upload
upload configuration
select serial port where your ESP8266 is connected to

```
Board: "NodeMCU 1.0 (ESP-12E Module)"
CPU Frequency: "80 MHz"
Flash Size: "4M (3M SPIFFS)"
Upload Speed: "921600"
Port: "COM10"
```

Note: Depending on version of platform package and board you have, you may see `Upload Using:` in the menu above. This option is inactive and it does not matter what you select. It has been left for compatibility with older implementation of OTA and finally removed in platform package version 2.2.0.

3. Upload the sketch (Ctrl+U). Once done, open Serial Monitor (Ctrl+Shift+M) and check if module has joined your Wi-Fi network:


BasicOTA | Arduino 1.6.7
File Edit Sketch Tools Help
BasicOTA §
Done uploading.
1) serial upload complete
COM10
Send
2) module successfully joined Wi-Fi network
Autoscroll
No line ending
115200 baud

```
  });
  ArduinoOTA.begin();
  Serial.println("Ready");
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());
```

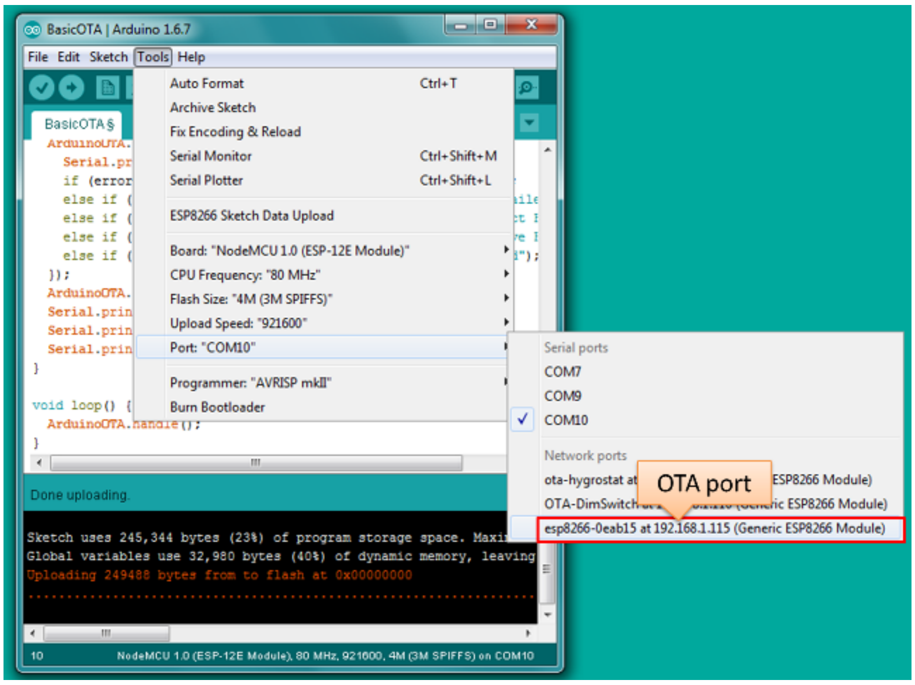
```
Sketch uses 245,344 bytes (23%) of program storage space. Maximum
Global variables use 32,980 bytes (40%) of dynamic memory, leaving
Uploading 249488 bytes from flash at 0x00000000
.............................................................
```

```
NodeMCU 1.0 (ESP-12E Module), 80 MHz, 921600, 4M (3M SPIFFS) on COM10
```

```
Ready
IP address: 192.168.1.115
```

Note: ESP module should be reset after serial upload. Otherwise subsequent steps will not work. Reset may be done automatically for you after opening serial monitor as visible on the screenshot above. It depends on how you have DTR and RTS wired from USB-Serial converter to the ESP. If reset is not done automatically, then do it by pressing reset button or manually cycling the power. For more details why this should be done please refer to FAQ regarding `ESP.restart()` .

4. Only if module is connected to network, after a couple of seconds, the esp8266-ota port will show up in Arduino IDE. Select port with IP address shown in the Serial Monitor window in previous step:

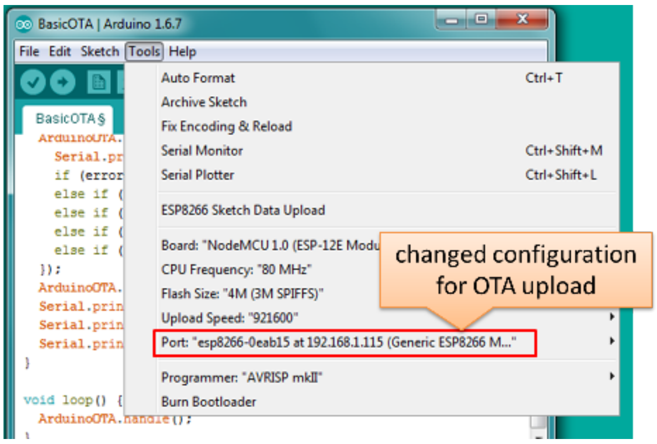

BasicOTA | Arduino 1.6.7
File Edit Sketch Tools Help
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Serial Monitor Ctrl+Shift+M
Serial Plotter Ctrl+Shift+L
ESP8266 Sketch Data Upload
Board: "NodeMCU 1.0 (ESP-12E Module)"
CPU Frequency: "80 MHz"
Flash Size: "4M (3M SPIFFS)"
Upload Speed: "921600"
Port: "COM10"
Programmer: "AVRISP mkII"
Burn Bootloader
Serial ports
COM7
COM9
✓ COM10
Network ports
ota-hygrostat at ESP8266 Module)
OTA-DimSwitch ic ESP8266 Module)
esp8266-0eab15 at 192.168.1.115 (Generic ESP8266 Module)
OTA port
BasicOTA §

```
Done uploading.

Sketch uses 245,344 bytes (23%) of program storage space. Maxi
Global variables use 32,980 bytes (40%) of dynamic memory, leaving
Uploading 249488 bytes from to flash at 0x00000000
..................................................................
```

```
10    NodeMCU 1.0 (ESP-12E Module), 80 MHz, 921600, 4M (3M SPIFFS) on COM10
```

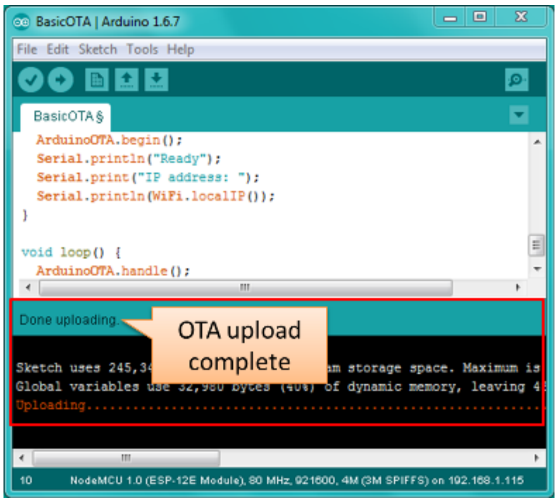
Note: If OTA port does not show up, exit Arduino IDE, open it again and check if port is there. If it does not help, check your firewall and router settings. OTA port is advertised using mDNS service. To check if port is visible by your PC, you can use application like Bonjour Browser.

5. Now get ready for your first OTA upload by selecting the OTA port:


BasicOTA | Arduino 1.6.7
File Edit Sketch Tools Help
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Serial Monitor Ctrl+Shift+M
Serial Plotter Ctrl+Shift+L
ESP8266 Sketch Data Upload
Board: "NodeMCU 1.0 (ESP-12E Modu
CPU Frequency: "80 MHz"
Flash Size: "4M (3M SPIFFS)"
Upload Speed: "921600"
Port: "esp8266-0eab15 at 192.168.1.115 (Generic ESP8266 M..."
Programmer: "AVRISP mkII"
Burn Bootloader
changed configuration for OTA upload
BasicOTA §

Note: The menu entry `Upload Speed:` does not matter at this point as it concerns the serial port. Just left it unchanged.

6. If you have successfully completed all the above steps, you can upload (Ctrl+U) the same (or any other) sketch over OTA:



```
  ArduinoOTA.begin();
  Serial.println("Ready");
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());
}

void loop() {
  ArduinoOTA.handle();
```

```
Done uploading.

Sketch uses 245,34                 m storage space. Maximum is
Global variables use 32,980 bytes (40%) of dynamic memory, leaving 4
Uploading...............................................................
```

```
10    NodeMCU 1.0 (ESP-12E Module), 80 MHz, 921600, 4M (3M SPIFFS) on 192.168.1.115
```

Note: To be able to upload your sketch over and over again using OTA, you need to embed OTA routines inside. Please use BasicOTA.ino as an example.

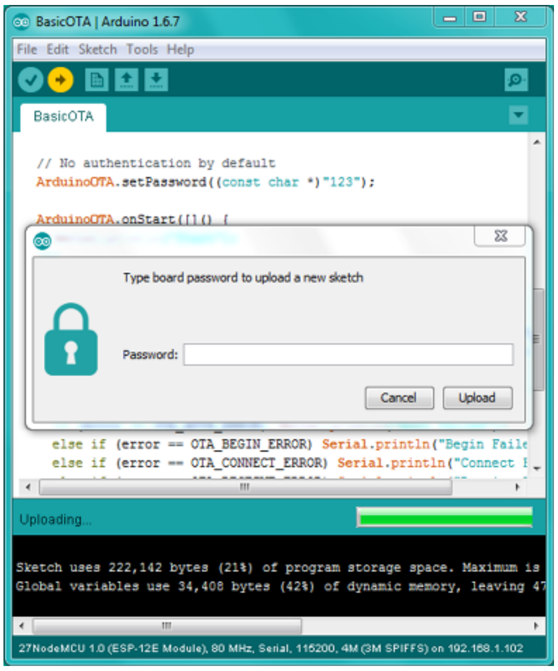
Password Protection

Protecting your OTA uploads with password is really straightforward. All you need to do, is to include the following statement in your code:

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

Where `123` is a sample password that you should replace with your own.

Before implementing it in your sketch, it is a good idea to check how it works using *BasicOTA.ino* sketch available under *File > Examples > ArduinoOTA*. Go ahead, open *BasicOTA.ino*, uncomment the above statement that is already there, and upload the sketch. To make troubleshooting easier, do not modify example sketch besides what is absolutely required. This is including original simple `123` OTA password. Then attempt to upload sketch again (using OTA). After compilation is complete, once upload is about to begin, you should see prompt for password as follows:



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

  ArduinoOTA.onStart([]() {
```

Type board password to upload a new sketch

Password:

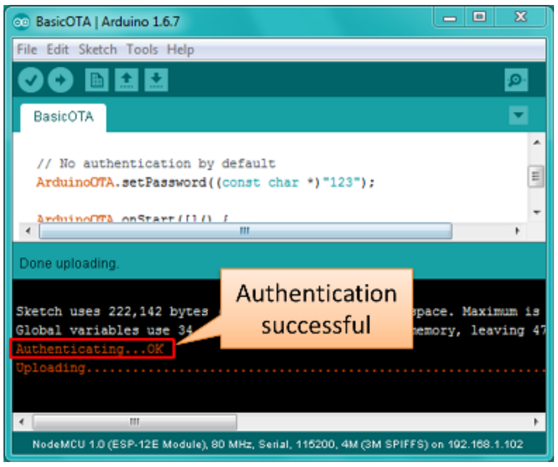
```
    else if (error == OTA_BEGIN_ERROR) Serial.println("Begin Faile
    else if (error == OTA_CONNECT_ERROR) Serial.println("Connect F
```

Uploading...

```
Sketch uses 222,142 bytes (21%) of program storage space. Maximum is
Global variables use 34,408 bytes (42%) of dynamic memory, leaving 47
```

27NodeMCU 1.0 (ESP-12E Module), 80 MHz, Serial, 115200, 4M (3M SPIFFS) on 192.168.1.102

Enter the password and upload should be initiated as usual with the only difference being `Authenticating...OK` message visible in upload log.



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

  ArduinoOTA.onStart([]() {
```

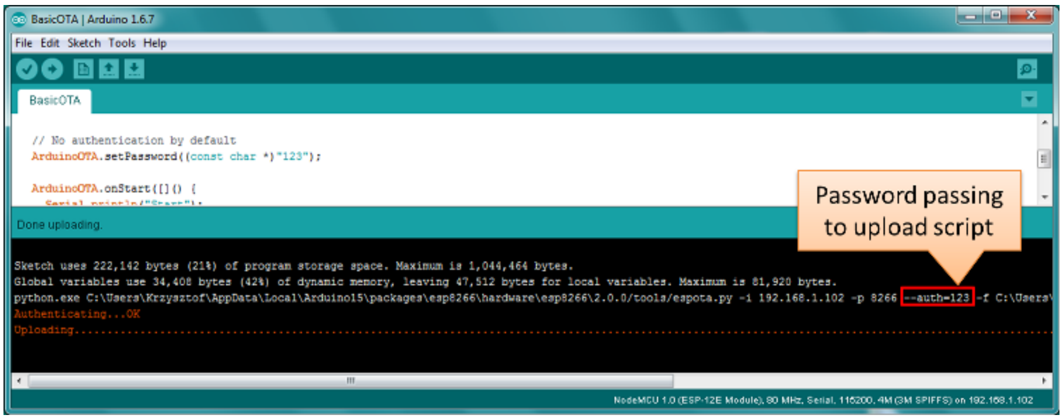
Done uploading.

```
Sketch uses 222,142 bytes                    space. Maximum is
Global variables use 34                      emory, leaving 47
Authenticating...OK
Uploading.............................................
```

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

You will not be prompted for a reentering the same password next time. Arduino IDE will remember it for you. You will see prompt for password only after reopening IDE, or if you change it in your sketch, upload the sketch and then try to upload it again.

Please note, it is possible to reveal password entered previously in Arduino IDE, if IDE has not been closed since last upload. This can be done by enabling *Show verbose output during: upload* in *File > Preferences* and attempting to upload the module.


BasicOTA | Arduino 1.6.7
File Edit Sketch Tools Help
BasicOTA
Password passing to upload script

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

ArduinoOTA.onStart([]() {
```

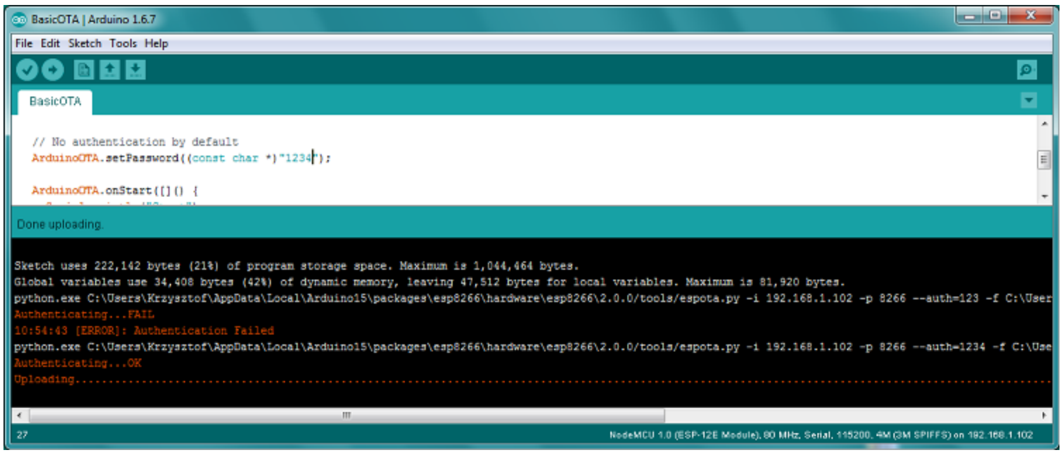
Done uploading.

```
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\
Authenticating...OK
Uploading..............................................................................................................
```

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

The picture above shows that the password is visible in log, as it is passed to *espota.py* upload script.

Another example below shows situation when password is changed between uploads.



```
// No authentication by default
ArduinoOTA.setPassword((const char *)"1234");

ArduinoOTA.onStart([]() {
```

Done uploading.

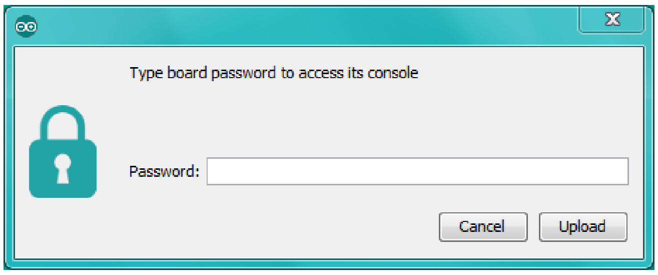
```
Sketch uses 222,142 bytes (21%) of program storage space. Maximum is 1,044,464 bytes.
Global variables use 34,408 bytes (42%) of dynamic memory, leaving 47,512 bytes for local variables. Maximum is 81,920 bytes.
python.exe C:\Users\Krzysztof\AppData\Local\Arduino15\packages\esp8266\hardware\esp8266\2.0.0/tools/espota.py -i 192.168.1.102 -p 8266 --auth=123 -f C:\User
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:\Use
Authenticating...OK
Uploading..............................................................................................................
```

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

When uploading, Arduino IDE used previously entered password, so the upload failed and that has been clearly reported by IDE. Only then IDE prompted for a new password. That was entered correctly and second attempt to upload has been successful.

Troubleshooting

If OTA update fails, first step is to check for error messages that may be shown in upload window of Arduino IDE. If this is not providing any useful hints, try to upload again while checking what is shown by ESP on serial port. Serial Monitor from IDE will not be useful in that case. When attempting to open it, you will likely see the following:

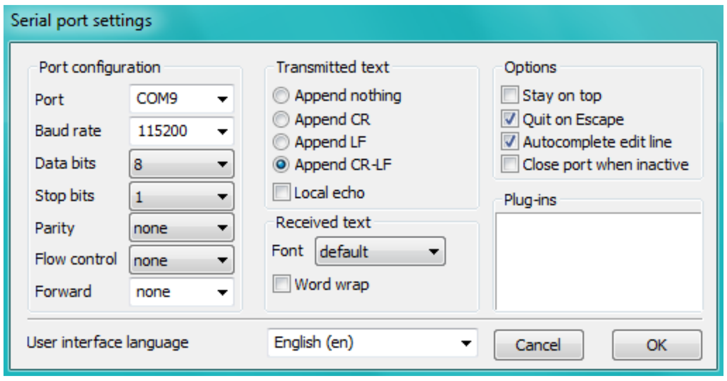

Type board password to access its console

Password:

This window is for Arduino Yún and not yet implemented for esp8266/Arduino. It shows up because IDE is attempting to open Serial Monitor using network port you have selected for OTA upload.

Instead you need an external serial monitor. If you are a Windows user check out Termite. This is handy, slick and simple RS232 terminal that does not impose RTS or DTR flow control. Such flow control may cause issues if you are using respective lines to toggle GPIO0 and RESET pins on ESP for upload.

Select COM port and baud rate on external terminal program as if you were using Arduino Serial Monitor. Please see typical settings for Termite below:


Serial port settings

Port configuration: Port COM9; Baud rate 115200; Data bits 8; Stop bits 1; Parity none; Flow control none; Forward none

Transmitted text: Append nothing; Append CR; Append LF; [✓ Append CR-LF]; Local echo

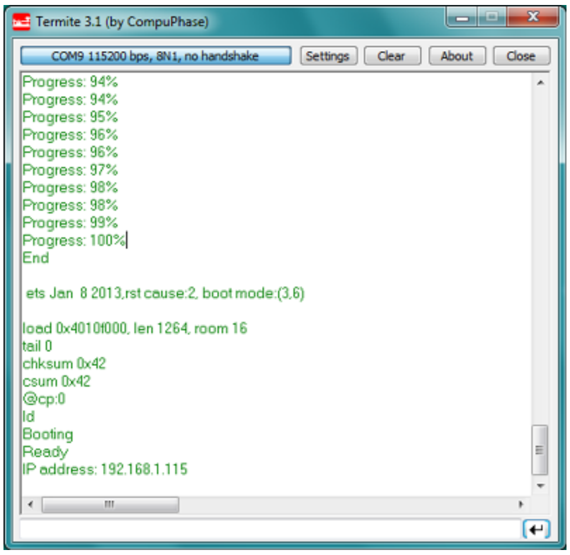
Received text: Font default; Word wrap

Options: Stay on top; [✓ Quit on Escape]; [✓ Autocomplete edit line]; Close port when inactive

Plug-ins

User interface language: English (en)

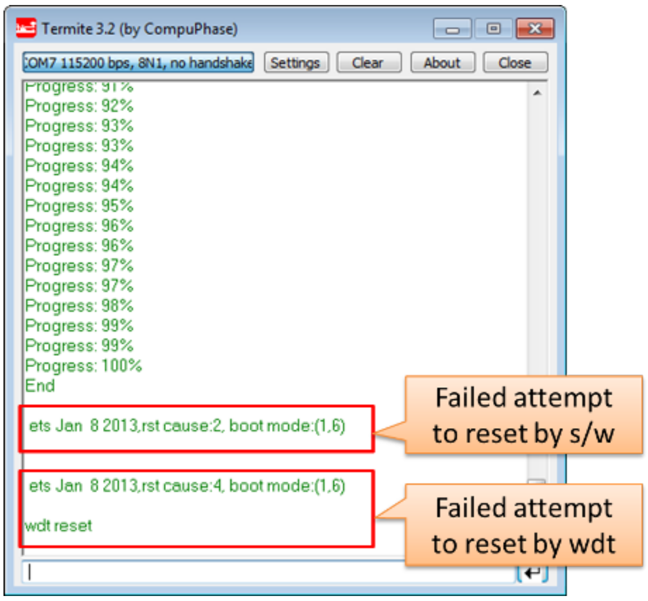
Then run OTA from IDE and look what is displayed on terminal. Successful ArduinoOTA process using BasicOTA.ino sketch looks like below (IP address depends on your network configuration):


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
```

If upload fails you will likely see errors caught by the uploader, exception and the stack trace, or both.

Instead of the log as on the above screen you may see the following:


Termite 3.2 (by CompuPhase)
COM7 115200 bps, 8N1, no handshake
Settings
Clear
About
Close
Failed attempt to reset by s/w
Failed attempt to reset by wdt

```
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
```

If this is the case, then most likely ESP module has not been reset after initial upload using serial port.

The most common causes of OTA failure are as follows: * not enough physical memory on the chip (e.g. ESP01 with 512K flash memory is not enough for OTA), * too much memory declared for SPIFFS so new sketch will not fit between existing sketch and SPIFFS – see Update process - memory view, * too little memory declared in Arduino IDE for your selected board (i.e. less than physical size), * not resetting the ESP module after initial upload using serial port.

For more details regarding flash memory layout please check File system. For overview where new sketch is stored, how it is copied and how memory is organized for the purpose of OTA see Update process - memory view.

Web Browser

Updates described in this chapter are done with a web browser that can be useful in the following typical scenarios:

- after application deployment if loading directly from Arduino IDE is inconvenient or not possible,
- after deployment if user is unable to expose module for OTA from external update server,
- to provide updates after deployment to small quantity of modules when setting an update server is not practicable.

Requirements

- The ESP and the computer must be connected to the same network.

Implementation Overview

Updates with a web browser are implemented using `ESP8266HTTPUpdateServer` class together with `ESP8266WebServer` and `ESP8266mDNS` classes. The following code is required to get it work:

setup()

```
MDNS.begin(host);

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

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

loop()

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

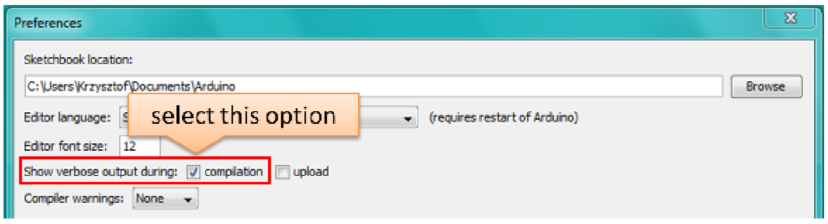
Application Example

The sample implementation provided below has been done using:

- example sketch WebUpdater.ino available in `ESP8266HTTPUpdateServer` library,
- NodeMCU 1.0 (ESP-12E Module).

You can use another module if it meets previously described requirements.

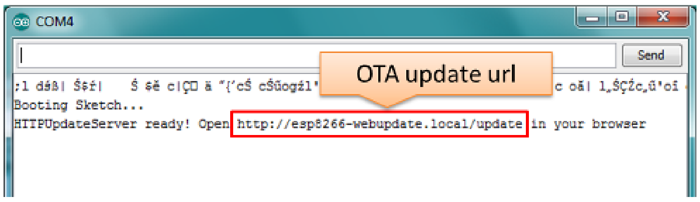
1. Before you begin, please make sure that you have the following software installed:
   - Arduino IDE and 2.0.0-rc1 (of Nov 17, 2015) version of platform package as described under https://github.com/esp8266/Arduino#installing-with-boards-manager
   - Host software depending on O/S you use:
     a. Avahi http://avahi.org/ for Linux
     b. Bonjour http://www.apple.com/support/bonjour/ for Windows
     c. Mac OSX and iOS - support is already built in / no any extra s/w is required
2. Prepare the sketch and configuration for initial upload with a serial port.
   - Start Arduino IDE and load sketch WebUpdater.ino available under File > Examples > ESP8266HTTPUpdateServer.
   - Update SSID and password in the sketch, so the module can join your Wi-Fi network.
   - Open File > Preferences, look for "Show verbose output during:" and check out "compilation" option.



Preferences
Sketchbook location:
C:\Users\Krzysztof\Documents\Arduino
Editor language: (requires restart of Arduino)
Editor font size: 12
Show verbose output during: [✓ compilation] upload
Compiler warnings: None

Note: This setting will be required in step 5 below. You can uncheck this setting afterwards.

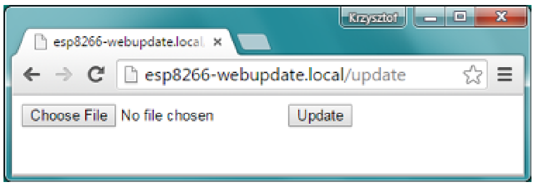
3. Upload sketch (Ctrl+U). Once done, open Serial Monitor (Ctrl+Shift+M) and check if you see the following message displayed, that contains url for OTA update.



```
;l dáß| Š$ŕ|    Š šě c|ÇD ä "{'cŠ cŠűogžl"                      c oä| l¸ŠÇŽc¸ű'oí
Booting Sketch...
HTTPUpdateServer ready! Open http://esp8266-webupdate.local/update in your browser
```

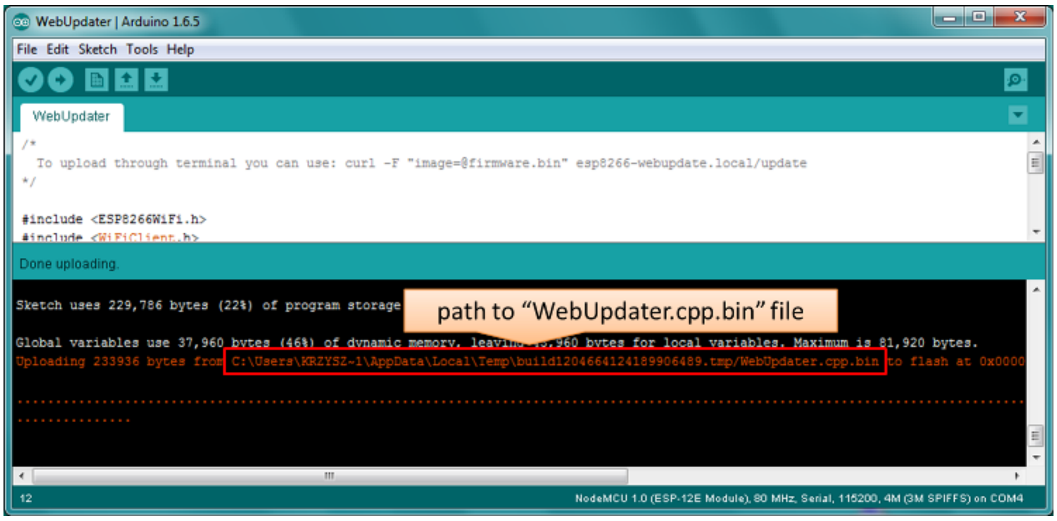
Note: Such message will be shown only after module successfully joins network and is ready for an OTA upload. Please remember about resetting the module once after serial upload as discussed in chapter Arduino IDE, step 3.

4. Now open web browser and enter the url provided on Serial Monitor, i.e. `http://esp8266-webupdate.local/update` . Once entered, browser should display a form like below that has been served by your module. The form invites you to choose a file for update.



Note: If entering `http://esp8266-webupdate.local/update` does not work, try replacing `esp8266-webupdate` with module's IP address. For example, if your module IP is `192.168.1.100` then url should be `http://192.168.1.100/update` . This workaround is useful in case the host software installed in step 1 does not work. If still nothing works and there are no clues on the Serial Monitor, try to diagnose issue by opening provided url in Google Chrome, pressing F12 and checking contents of "Console" and "Network" tabs. Chrome provides some advanced logging on these tabs.

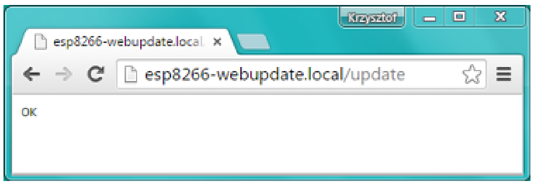
5. To obtain the file, navigate to directory used by Arduino IDE to store results of compilation. You can check the path to this file in compilation log shown in IDE debug window as marked below.


WebUpdater | Arduino 1.6.5
File Edit Sketch Tools Help
WebUpdater
Done uploading.
path to "WebUpdater.cpp.bin" file
12
NodeMCU 1.0 (ESP-12E Module), 80 MHz, Serial, 115200, 4M (3M SPIFFS) on COM4

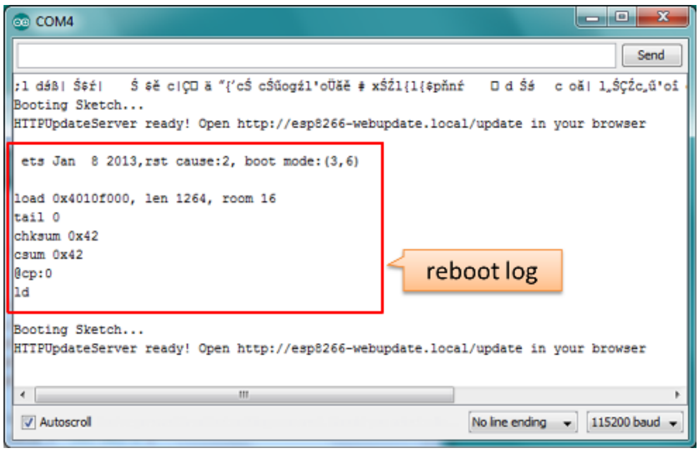
```
/*
  To upload through terminal you can use: curl -F "image=@firmware.bin" esp8266-webupdate.local/update
*/

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

```
Sketch uses 229,786 bytes (22%) of program storage

Global variables use 37,960 bytes (46%) of dynamic memory, leaving 43,960 bytes for local variables. Maximum is 81,920 bytes.
Uploading 233936 bytes from C:\Users\KRZYSZ~1\AppData\Local\Temp\build1204664124189906489.tmp/WebUpdater.cpp.bin to flash at 0x0000

.................................................................................................................................
..............
```

6. Now press "Choose File" in web browser, go to directory identified in step 5 above, find the file "WebUpdater.cpp.bin" and upload it. If upload is successful, you will see "OK" on web browser like below.



Module will reboot that should be visible on Serial Monitor:



```
;1 dśß| Ś$ŕ|    Ś śě c|ÇD ä "{'cŚ cŠűogźl'oŮăě # xŠŽl{l{$pňnŕ   D d Śś   c oă| l‚ŚÇŽc‚ű'oí
Booting Sketch...
HTTPUpdateServer ready! Open http://esp8266-webupdate.local/update in your browser

 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 Sketch...
HTTPUpdateServer ready! Open http://esp8266-webupdate.local/update in your browser
```

Just after reboot you should see exactly the same message `HTTPUpdateServer ready! Open http:// esp8266-webupdate.local /update in your browser` like in step 3. This is because module has been loaded again with the same code – first using serial port, and then using OTA.

Once you are comfortable with this procedure, go ahead and modify WebUpdater.ino sketch to print some additional messages, compile it, locate new binary file and upload it using web browser to see entered changes on a Serial Monitor.

You can also add OTA routines to your own sketch following guidelines in Implementation Overview above. If this is done correctly, you should be always able to upload new sketch over the previous one using a web browser.

In case OTA update fails dead after entering modifications in your sketch, you can always recover module by loading it over a serial port. Then diagnose the issue with sketch using Serial Monitor. Once the issue is fixed try OTA again.

HTTP Server

`ESPhttpUpdate` class can check for updates and download a binary file from HTTP web server. It is possible to download updates from every IP or domain address on the network or Internet.

Requirements

- web server

Arduino code

Simple updater

Simple updater downloads the file every time the function is called.

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

Advanced updater

Its possible to point update function to a script at the server. If version string argument is given, it will be sent to the server. Server side script can use this to check if update should be performed.

Server side script can respond as follows: - response code 200, and send the firmware image, - or response code 304 to notify ESP that no update is required.

```
t_httpUpdate_return ret = ESPhttpUpdate.update("192.168.0.2", 80, "/esp/update/arduino.php", "optional current versic
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;
}
```

Server request handling

Simple updater

For the simple updater the server only needs to deliver the binary file for update.

Advanced updater

For advanced update management a script needs to run at the server side, for example a PHP script. At every update request the ESP sends some information in HTTP headers to the server.

Example header data:

```
[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
```

With this information the script now can check if an update is needed. It is also possible to deliver different binaries based on the MAC address for example.

Script example:

```
<?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();
}

$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_ESP826
    || $_SERVER["HTTP_X_ESP8266_SKETCH_MD5"] != md5_file($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);
```

Stream Interface

TODO describe Stream Interface

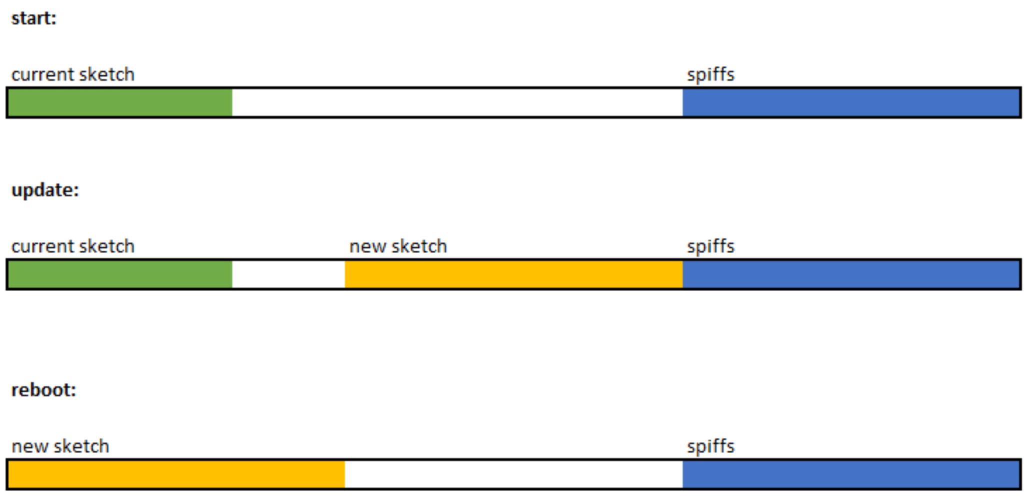
The Stream Interface is the base for all other update modes like OTA, http Server / client.

Updater class

Updater is in the Core and deals with writing the firmware to the flash, checking its integrity and telling the bootloader to load the new firmware on the next boot.

Update process - memory view

- The new sketch will be stored in the space between the old sketch and the spiff.
- on the next reboot the "eboot" bootloader check for commands.
- the new sketch is now copied "over" the old one.
- the new sketch is started.


start:
current sketch
spiffs
update:
current sketch
new sketch
spiffs
reboot:
new sketch
spiffs