Sometimes, you want to deploy your ESP8266 in a place where it is not easily accessible anymore. Or you have a few productive ESPs around your house, or also at other places, even in other countries. You want to be able to keep the software of all these ESPs up-to-date without going there and without opening the boxes the ESPs are built in.

Or you want to try a sketch I wrote and you do not want to install all the libraries and so on. You just want to use the software.

An App store for IOT devices would solve all these problems.

As an example: In video #72 and #76 we started to modify an electronic load by replacing the original display with an ESP8266 and a Nextion display. Because at the end, this will be a test equipment for the bench, I do not want to foresee any connectors for future updates of the software. But I still want to have the flexibility to enhance the software from time to time. This lead me to the following idea: To build a wireless application store for IOT devices.

In this video I will show you a first implementation of such an IOT-appstore and you will be able to use mine with your own ESP8266 board. Or you create your own.

Let’s get started.

In video #32 I showed you how you can upload your sketches over the air, or OTA, from your Arduino IDE. Here, we want to go one step further. We want, that your device is able to update itself without any intervention to the newest software version, just like your mobile or your tablet.

The newest ESP extensions of the Arduino IDE offer the possibility to do an OTA software update from a webserver. Instead of transferring your sketch from the IDE, your ESP calls a webserver to get the file to be flashed. The webserver provides this file and the update function in your ESP downloads it via Internet and stores it into a different area in the flash memory. If everything is ok, it replaces the current program the new one, and reboots. This is, why your ESP modules need at least 2x the flash size of your sketch. With our new 4 MByte modules, this is usually no problem.

You can easily try this function if you have a webserver. This can be a raspberry or also your windows PC. I will not cover the set-up of such a server in this video. Because not everybody has its own webserver I placed a few files on my webserver for your tests. So, you can try the mechanism right away.

But let’s first see, how this file is created. I use the normal “blink” sketch for now.

We create our sketch the normal way by replacing the LED pin with pin 2 of our ESP and upload it to our ESP via serial. Just as normal. As soon as your sketch is tested and ready for “deployment” to your wireless ESP through our new Webserver, you enable the flag “show verbose output during upload” in your IDE. Now, much more information is displayed during upload. Please search for the directory where the file with the \*.bin extension is stored. Pay attention, this directory can change if you use a different sketch.

If we now go to this folder, we find our bin file. It should have a name like our sketch. This binary file contains your sketch translated, or “compiled”, to machine readable code for your ESP. Now, you can copy this file to your webserver. I used Filezilla to copy exactly this file to my webserver.

Now you can take your ESP module. Let’s assume, you use this Lolit board. To start the process, you have to upload the sketch “httpUpdate” from the examples to your board. Please replace the credentials and the address of the file to be downloaded. If you want to use mine, use the following file name: <http://iotappstore/blink.bin>. Make sure you include the “http”.

If your sketch is uploaded to your board, you have to reset the ESP manually. Otherwise it will not automatically reboot after OTA flashing. You even can connect your ESP to a power bank without physical connection to the IDE. After a few seconds, your ESP should start to blink. This is the prove that you did your first OTA upload from a webserver. Easy.

If you want, you can leave your ESP connected to your PC and watch the process by enabling the Serial monitor.

Let’s assume, we improve our sketch to make it blink faster. Will it be downloaded automatically? Unfortunately, the normal blink sketch on your ESP has no possibility for a webupdate and therefore, this slow sketch will stay as long as you do not intervene manually and do the same thing we did again.

So, we have to build a “genetically engineered” blink sketch which, always after booting, downloads the newest file from the server. With this feature, your ESP would update automatically if I would replace the file on the server with a new one.

So, let’s start with this “genetic engineering”. We combine the two sketches by including the httpUpdate part into the setup() of our blink sketch. Like that, our sketch will start with the update process before it goes on with blinking. So, we always have the newest version on our ESP. We store this new file also on our webserver with the name Blink\_GMO. If you now use your httpUplaoad sketch again and replace the filename with the new file, your ESP downloads the new file, and, as planned, starts with the OTA. Great!

But, unfortunately, the newly downloaded file does not work. Your ESP does not connect to your Wi-Fi. Why is this?

The reason for this behavior is, that I (fortunately), do not know your Wi-Fi password. This is, why I had to use a dummy SSID and password when I created the file uploaded to the webserver. And this file is now on your ESP. We could overcome this problem if you would provide me your SSID and password. Then, I could write a special file for you and you could download it.

For both of us, this is not a good solution. For you, because you would have to give your password to me and for me, because I would have to maintain a special file for each of you.

If you work on your own webserver, however, this is already a feasible concept. But for our appstore, we have put this problem as the first entry on the open points list.

To go on, I replaced the credentials in the file on my webserver. Now, my ESP connects to the Wi-Fi and updates the sketch. Cool.

But wait: What happens after successful update? The ESP reboots again and starts the flashing over and over. It never reaches the blink part. Why is that happening? Because there is no mechanism to check, if the version in the flash memory is the same as the one on the server, it updates the sketch again and again. Another open point for our list.

But at least, the basic principle worked.

So, lets first solve point 2, because we have to solve this one even if you use your own webserver.

Fortunately, the guys who developed the webUpdate functionality provide a solution to that point: We do not download the file directly. At the beginning of the Upload, the uploader calls a PHP script on the server which decides, if our ESP is up-to-date or needs a new download. If it needs a new download, it sends the needed file and the ESP flashes it to its memory. If not, it just sends the answer to continue with the sketch.

So, this should work on your ESP. But how can the PHP script decide if, and also, which file to send? Maybe you have several ESP modules for different purposes. Then, you do not want, that, after the automatic update, the electronic load behaves like the irrigation controller or vice versa.

The PHP script uses a simple mechanism: Every device in the internet has a distinct so called “MAC address”. Not two devices should have the same address. Each of your ESPs therefore should have their own MAC address when you purchased them. You can easily display the mac address with the command wifi.macaddress().

The httpUpdate call automatically includes this address in its call to the webserver and the script knows which mac address needs which file. So, it sends the appropriate file to your ESP. Because the ESP also can send the name and the version of the currently installed file, the PHP script can also decide if it needs an update or not.

Let’s check: We want, that our downloader file downloads the enhanced version of blink, “fastblink\_001”. Because we need the MAC address, we enhance our initial uploader file with the possibility to print its mac address. We also replace the file to be downloaded with the name of the PHP script on the webserver, and we write the name and version of the currently installed sketch into the httpUpdate call.

Now we can try to update. Unfortunately, we get the response: “file not found”. Why is that? We forgot to enter the MAC address of our device into the PHP script. If we do that, the PHP scrips gets the MAC address and the name of the currently installed sketch. A comparison shows, that our ESP does not have the newest version. So, it has to send the “fastblinkGMO” file to our device and really, the LED starts to blink fast. If we reboot the ESP now, the php script checks the sketch versions and just signals “you are up-to-date” and the ESP continues into blinking. So, open point 2 is solved and we can go on to open point 1: To store the Wi-Fi credentials locally on your ESP.

This can be done by using the EEPROM functionality. Right after booting, we check a “magic byte” in the EEPROM, and if this byte is set correctly, we assume, that also the SSID and password in the EEPROM can be used. If not, and this is the case if you start with a new ESP, the sketch uses the SSID and password provided by you in your credentials.h file. After downloading the new file from the IOTappstore, the sketch will use your credentials, because the magic byte is set. So, the file in the IOTappstore does not need your credentials anymore. You find this initialUpdate file on my github. So, open point 1 is also solved.

Summarized, we built an infrastructure to update all your diverse ESP modules from an IOTappstore. We used the standard initialUpdate function of the ESP and a PHP script on the webserver. The webserver can be hosted by yourself, or you can use mine. The initialUpdate sketch gets your credentials from the EEPROM of your ESP, and calls the PHP script on the webserver. If the webserver knows the mac address of your device and your script needs an update, it sends the requested file and your ESP updates. If not, it sends “no update necessary” and your ESP starts its work. Everything completely without human intervention.

You can go one step further and automate the process completely by using the time() function shown in video #71 to schedule a planned version check every day, or every week.

If you now want to use the IOTappstore.org examples, please put the mac address of your ESP in the comment and also, whether you want, that your ESP blinks slow, fast, or if you want to get a surprise. For the surprise, you have to connect a loudspeaker between GPIO5 and ground.

I hope, this video was useful or at least interesting for you. Bye.