In episode #72 we started to dig into the topic of making the electrons in our designs somehow visible for us. I choose a small project as an example: To enhance the user interface of a Chinese electronic load with a Nextion display and a data logging feature.

In episode #76 we advanced the project, but still had open points. I try now to close some of these points. But, as always in projects, new points popped up and I had to solve these, too.

Let’s continue the learning journey and see, how far we can get today.

The open points from the last episode were:

To get all working together. For debugging, we need a wireless Serial Monitor function.

Our device has to act according the reading of the displayed values. For example, we have to reset the load, select the mode of operation, switch the beeper off, set voltages and amperage, and start and stop the current. Synchronization with a software not written to be remotely controlled and error handling will be also important for this point.

We have to test the whole thing and find also infrequent errors and refine our skills in setting traps for errors

We have to introduce data logging. However, because this is the “cherry on the cake”, I will postpone it to a future episode.

The last two points also will not be covered in this episode. But we have to add another one: Simple Over the Air programming which is compatible with wireless Serial. In episode #78 you got a first glimpse into this topic. We will work on this and make it really simple.

Let’s continue:

The current status looks like that: Everything on the bench and on a breadboard. To get the things a little organized, I designed a small PCB which will replace the current display unit. Osh Park sent me three nice prototypes. They are double sided. The ESP-12E module will sit on one side, the rest of the parts on the other. I already started to solder the SMD parts using my Aldi oven. I have no fancy controller, I just added a temperature display and control the temperature manually.

Then, I soldered the pin headers manually to the board, and now, I have to solder the ESP-12 to the other side. But, before I solder it to the board, we have to divert into another topic and come back later.

One of the major problems in developing software for remote devices which are flashed over the air is, that, after deployment, we have no more serial connection to the ESP. The same applies in my project, because I use the serial connection to connect the Nextion display. To solve this, we can use a protocol called Telnet. Telnet is a very old and simple protocol which basically replaces the serial wires. In our case, we include some lines of code in our script.

We have to declare the telnet server, and write the small telnet handler which is called in the loop(). During setup, we just have to start telnet, and set the noDelay. If we want to print now, we just replace the “Serial” with “Telnet” and the output is sent via the telnet protocol.

To receive it, we still use our well known PUTTY terminal, we just select “telnet” instead of “serial”. But wait: Putty asks us for an IP address. How can we find this address?

The easiest way is, to print it during start-up using the command WiFi.localIP(). This is easy with the serial connection. But, everybody who works with Wi-Fi knows, that this address can change because it is allocated by a mechanism called DHCP. So, we have to find a way to create a permanent address, which is still compatible with our other devices. The first possibility is, to assign a fixed address to your ESP sketch. This is done by the command Wifi.config and the desired ip address, the gateway, and the subnet.

Now, we can enter this address into Putty, and really, we get a connection. Be aware, that you lose this connection if you reboot your ESP, even if the IP address stays the same. This is, why you can store some typical addresses in Putty and call them again with a double click.

The other possibility to assign a permanent address is using your DHCP server, which is often your router. There, you find a menu “DHCP, where you can assign a distinct address to a particular MAC address. In episode #78 we learned how to find the MAC address of our ESP and, after entering this combination, our ESP gets automatically this IP address assigned. We can change it without changing anything on the ESP itself, which is good, if it is deployed at an inaccessible place.

So, we have reached our goal of having a wireless serial connection. Perfect. Now, we need the possibility to update it over the air. For that, we can use the normal OTA functionality of the Arduino IDE to the ESP as described in episode #32. Unfortunately, this functionality is also based on telnet. I was not able to use both functionalities together in the same sketch. So, I had to find another possibility. I created the iotappstore of episode #78 where we can download the sketch from a webserver over the air. The Webupdate can be done at the beginning of the setup() routine, and we can start telnet afterwards. So, we have solved also this problem.

Now, I can give a short update on the iotappstore itself. I got a few comments arguing, that the way it was implemented, was quite cumbersome. And during my working with it, I had to agree. Especially the need to rename several things for each new version was not comfortable. Fortunately, I found a very good solution for that problem. The ESP Arduino team currently works on the release 2.4. And there, they will introduce a new possibility for the Webupdate: They use MD5 tags. These tags are calculated by adding all bytes in the code and create a so called “hash” value. Two different codes always have two different MD5 values, even if the difference is very small. The new concept uses this fact and compares the MD5 tag of the currently installed sketch on the ESP with the sketch on the webserver. If they are the same, no update is necessary. If they are different, an update has to be performed. Simple.

Now, the workflow is very simple. You change something in your Arduino IDE. If you want to test the change, you just press ctrl alt s or “export compiled binary” and the IDE writes the binary file into the folder of your ino file. The name of the file is similar to the name of the sketch. This name has to be written into the PHP script on the webserver, like in episode #78. But this name does not change anymore with each version, because the php script searches now always for the newest version using the MD5 tag. Exactly as described before. Therefore, the PHP script has to be adapted only once for a project. The name of the sketch, which also had to be adapted to each new version, is now irrelevant. It is no more used for the update process.

I use a small tool to automatically transfer the new bin file to the webserver. As soon as I boot the ESP, it calls the php script and gets the new file, because the two MD5 values are different. After the update, it calls the php script again, and gets the answer “no update necessary”. So, it can continue to do its work.

The initial loader has to be programmed physically via a serial connection. In our case, it has obviously to be done before the module is soldered to the PCB. And this is, where we diverted a few minutes ago.

I use my small tool from episode #30 to flash a “virgin” ESP. It just consists of pogo pins for all important programming pins and a USB to serial adapter. The pins are connected according the normal schema. I just added a switch to power the whole thing up. As said before, the uploading is done directly form the IDE. From now on, every time we boot this ESP, it checks, if a new version of the sketch defined in the PHP file is available and updates itself if necessary. Of course, without any physical connection.

So, I can continue with my project, which is way more complicated than thought at the beginning. You just see it if you look at the many versions I already had to build.

Let’s quick check, where we are:

We now have a wireless serial monitor which is compatible with OTA. I am still in the middle of the task to make the ESP sketch work with the code of the electronic load and I found also some infrequent errors, which mainly had to do with overrunning numbers and interrupts. But I am pretty sure, I am not through. So, we leave this point open. The data logging and the box is definitively still open. And for the evaluation, I can just tell you, that this project already took me more than 100 hours till now. But I learned a lot and have now my own iotappstore. Cool

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