Hello YouTubers, here is the guy with the Swiss accent. Last week we started with our home automation system based on PIR sensors and Sonoff Switches. Both devices were operated by a Firmware based on Tinkerman’s Espurna framework.

The “brains” of the system is a Raspberry Pi zero and a script provided by Peter Scargill. And all communication is done via the MQTT protocol.

Today, we will bring these things together and dig a little into node-red and SQLite. I will also show you, how you can monitor your Pi Zero from a web interface.

So, let’s start:

You remember, I have 2 PIR sensors, one Alexa, and two Sonoff switches. This is just a small setup, but we can learn a lot which can be used if we extend this setup.

Last week, we were able to send messages to a public MQTT broker. Because we do not want to leave our secure Wi-Fi network, we first setup our own MQTT broker on a Raspberry Pi behind our Firewall. I did this in video #126. I even posted an image for a Pi Zero. You find a link in the description. So, you have an easy start. If you want to use another Raspberry, you have to install Raspbian and run the script of Peter. You find a link in the description.

So, we have now a working MQTT broker. With the utility “DietPi-setup”, we can give it a fixed IP address. You can also work with the name of your device, but, I decided for a fixed address. I also decided to use my old Pi zero without Wi-Fi, because I will place the device close to my switch and I am able to connect it to a Ethernet cable. For that, I purchased a 3$ USB hub with Ethernet connection. It runs without any problems. I even can insert the Keyboard in one of the three USB plugs if I want.

So, The IP address is 192.168.0.203 and its name is hub.local. Next, we have to change the MQTT broker addresses in our PIR sensors and our Sonoff switches. This can be done by the webinterface introduced in Video #127. Here, we are also able to set the first part of the topics string. The device will automatically insert its board name and add things like “relay” or “alexa”. For the moment, I do not use username/password for MQTT. Now, we can start MQTT with the command mosquito -v. When we power our devices, we should now see a lot of messages coming. Each message has a topic and a payload. This is, because the devices publish many different topics to the MQTT broker.

We can now “subscribe” to particular topics using “mosquito-sub” plus the desired topic. Now, we only see the messages of this topic.

If you use my SD card image, you have to issue two additional commands now. Otherwise, later-on, you will see an empty page.

Now, we know, that the broker runs and we can go on to the next step. We start node-red on our Pi. As soon as it runs, we can connect to it from a browser by calling the address of our raspberry. Just type the IP address in a browser window and you get this nice menu. For the next step, we go to node-red control panel. If there is already something there, you can delete it. For this video, we start with a fresh Flow. But how do you get this flow? This is really easy. You go to my Github and open the file “flow.txt” with any editor. Then, you select the whole text and copy it with crtl-c. Next, you go to node-red and import from clipboard. Paste the content of your clipboard and decide, if you want a new flow. Hit import and the miracle happens. You have exactly my flow on your computer. Cool. BTW. You can do this with all sorts of objects you find in the internet.

Now we start from left to right. The first node is a so called MQTT node. You see this if you compare the color and the symbol with the “palette” on the left. You see also, that it is an “input node”.

Next, we have to connect this input node with our MQTT broker. Easy: Because we run on the same computer, we just type-in localhost and the Mosquitto port is 1883. Fortunately, we only have to enter this once and can refer later to it. Because this node should connect to the main PIR sensor, we have to select the right topic. The client id and the other tabs are not important for the moment. If you are interested in these, you can watch my video #125.

If you do not remember the topics, just enter # as topic, “done” and deploy. Connected to this node is a so-called “debug node”. This node sends messages to the debug window. We can define, how much info we want and we can switch it on or off. Now, we see the topic and the payload. “0” payload means “off” and “1” means on. This is defined by the firmware of our PIR sensor. So, the connection to our PIR sensor works.

Next, I want, that the lamps keep on for 5 minutes after my last movement. This is to prevent, that the light goes on-and off if I am thinking…

To achieve that, we can choose a so-called “function” node. I use the trigger node for our purpose. This has the desired function: It sends an “on” message to its output as soon as it gets any message at its input. After 5 minutes, it automatically sends an “off” message. We also select re-trigger.

As we have seen before, the PIR sensor sends on and off messages. If we want, that only on messages reach our “Pulse” node, we can filter them with a switch node. Here, I define, that I only want the “on” messages forwarded. The off messages are suppressed. I could add a second output for the off messages…

The pulse node has a nice feature: it shows us its status with a blue dot. Later on, we will build such a feature for our own nodes.

Another nice feature is the sound node: It helps for example during debugging, when you work with your sensor and you cannot monitor node red at the same time. So, I added an audio output. In several languages, of course! Now, I can concentrate on my work and get acoustic feedback. How cool is that? BTW, you can look at these nodes if you are interested on how you can achieve this function.

We could connect the output of the pulse node directly to our LIGHT\_Main node. LIGHT\_Main, by the way, is also an MQTT node, this time an output node. And its topic is the topic of the Sonoff to switch the light on.

But, we want more: We want, that we are able to override the PIR sensors with either a dashboard or with Alexa.

So, we introduce a new input node: A dashboard button node. Its function is simple: It creates or “injects” a 1 for on, a 0 for off, and a “A” for automatic. But is not operated from this page. If we go back to our main page and select “Node Red UI Desktop”, we get a new window. Here, we see for each of the nodes a button with a label. I also grouped the different buttons according their purpose. So, I have now a small “command center”.

But now, we have two different chains of command for our Sonoff switches: The PIR signals and the signals from our Dashboard. It is quite obvious, how this should work: If we press the on or off button on the dashboard, we want, that the lamp is on or off, even if a conflicting signal comes from the PIR sensor. Only if it is on “Automatic”, we want, that the signal is transferred directly from our “Pulse” node to the Lamp. But how can we achieve this?

Here, we need a few things: We have to use a fully programmable node, and we have also to have variables which can store a particular status. The first is easy: We just take the empty “function” node. This node is programmable in java script.

So, let’s have a closer look. If we look at a new node, we just see, that it returns the message it gets. So, we learn two things:

1. The full message on the input is called msg and
2. We can transfer a message to the output by “returning” it.

Simple. Now, let’s look at such a message. To do that, we use our PIR sensor and the debug node. The only thing we do is, to select the complete message. Then, we see an “object”, and its content, which we easily refer to. You will later see, how.

To play with that setup, I place the function node between the PIR sensor and the debug node. So, if we want to refer to the topic, we use msg.topic. So, if we want to transfer the topic also to the payload, we just write this small code. And if we test now, we see in the debug tab, that the magic works. But very often, the debug tab is clouded by many different messages. We have to find a better way. What, if we could see important info in proximity to the node, like with the pulse sensor? Easy, if you know, how. We just add this line of code, and we now see the payload directly in our flow. Nice.

But how do we get this “override” behavior? Here, we can add a few lines of code: If the “override” variable is “ON” we want, that the payload always is “1”, and if “OFF”, it has to be “0”. If the variable is “auto”, we transfer the payload without change from the input to the output. So, we have to find a way to set and store the override flag. So far, we only had messages to work with. If the next message arrives, we lose the content of the former. If we look at our override node, we get sometimes a message from the “always on” button, or from the PIR sensor and so on. To achieve our goal, we have to store the override flag every time a message from the dashboard arrives. And the override flag should not be affected by the messages from the PIR sensor. These lines of code do exactly that. Our dashboard buttons all include the word “OVERRIDE” in their topic. So, we filter these messages and set the override flag accordingly. If we now test our setup, it behaves exactly as desired. We can use Alexa to switch the lamp on or off, even if the PIR sensor is still on.

We could now go on to the next topic. But I do not want to omit telling you, that Peter Scargill wrote a small article about variables in node-red. You can, for example, store global variables in one place and use them in another. If we store the result of the override flag in a global variable, we can use it in our second node. We will use this function in our data logger.

From our previous videos, we know, that Peter’s script also installed the SQLite database. Here, we want to store all events from the two PIR sensors, together with the override flags. For that, we go to the main page and from there, to SQLite administrator. Here, we create a database called Sonoff and add four fields: Topic, payload, timestamp, overrideMain, and overrideTable. That is all here. We go back to our node red tab and start with connecting the two PIR sensors to a function node. This function node is connected to a SQLite node. Now, our function node gets all messages from the two PIR sensors. Now, we just have to bring all into a form in which the SQLite node will store it properly away. In my last video, I showed you one version to do that. Some of my viewers made a point, that this method is not very secure. So, this time, I use a different method and hope, it is better.

The SQLite node needs a “SQL statement as a topic and the field content as a payload. So, the SQL statement is “INSERT INTO Sonoff” and all fields of the database. At the end, we add a “VALUES” template with the same number of question marks. And the payload contains the content of the variables in the right order. The rest of the code is easily understandable. It only formats the time stamp in a “human readable format”. And, of course, displays the last update time as a reference below the node.

Now, everything is done and we can deploy the flow a last time. If you want to create a backup, you can export the whole content or only selected nodes into your clipboard. Then, you paste it in a text file and you are done. Very simple, too.

And, if you want to know how much resources node red uses, you go back to the main page, and to “Web Administrator”. Here, you can do all the small stuff an administrator has to do.

So, summarized,

we are able to control our two lamps by two PIR sensors

can override their signal by either pressing a button on a dashboard or by calling Alexa

Were able to use voice as an output

were able to store a nice log in a structured database.

And we even learned a few lines of java script.

For me, I see now in the morning, if my PIR sensors were triggered during night. Then, the usual suspect would be our cat called Dishka. fortunately, we did not name her “Alexa”.

I hope, this video was useful or at least interesting. If true, then Like. Bye