Some more Hello World examples for ESPHome

By jean-claude.feltes@education.lu 14.3.2023

The first part of this document is found here:
http://staff.ltam.lu/feljc/electronics/homeassistant/ESPHome_programming_1.pdf

1. Common part of the code for the examples

I have used a common part of code for all the examples, something like this:

```
substitutions:
  devicename: <name>

esphome:
  name: $devicename

esp8266:
  board: d1_mini

logger:
  level: DEBUG

api:
  password: !secret api_password

ota:
  password: !secret ota_password

wifi:
  ssid: !secret wifi_ssid
  password: !secret wifi_password

  ap:
    ssid: "Fallback Hotspot"
    password: !secret ap_password

  manual_ip:
    static_ip: 192.168.0.33
    gateway: 192.168.0.100
    subnet: 255.255.255.0
```

(With SSID and password in the file secrets.yaml)

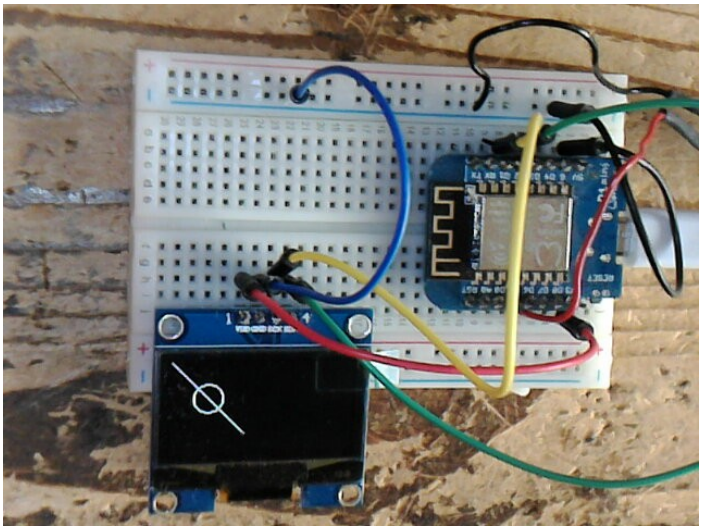
The code listed beneath is simply appended to the YAML file.

2. OLED-Displays

2.1. A simple static drawing example

A simple example drawing a static image consisting of a line and a circle:

```
# Display:
i2c:
  sda: GPIO4
  scl: GPIO5

display:
  - platform: ssd1306_i2c
    model: "SH1106 128x64"
    address: 0x3C
    lambda: |-
      it.fill(COLOR_OFF);
      it.line(0, 0, 50, 50);
      it.circle(25, 25, 10);
```



What is the mysterious “it” object?
In this forum discussion https://community.home-assistant.io/t/lambda-function-in-esphome-for-displays-what-is-this-it-object/371934/2 there is more about it:

‘it’ is an object (display buffer, basically) defined within ESPHome and passed to the YAML-macro’s lambda on each callback. If you let ESPHome manage the display (and you should), then all display actions should go through ‘it.’

For a more profound idea of what it can do you have to consult the API documentation:

https://esphome.io/api/classesphome_1_1display_1_1_display_buffer.html

2.2. Multiple pages

This example switches between 2 different display pages every 5s:

```
i2c:
  sda: GPIO4
  scl: GPIO5
  scan: false

font:
  - file: "font/courier.ttf"
    id: myfont
    size: 16


interval:
  - interval: 5s
    then:
      - display.page.show_next: mydisplay
```

```
      - component.update: mydisplay

display:
  - platform: ssd1306_i2c
    model: "SH1106_128X64"
    address: 0x3C
    update_interval: 5s
    id: mydisplay
    pages:
      - id: page1
        lambda: |-
          it.print(0, 10, id(myfont), "page 1");
      - id: page2
        lambda: |-
          it.print(0, 10, id(myfont), "page 2");
```

https://esphome.io/components/display/index.html

2.3. Showing the boot process with multiple pages

My idea was to indicate the boot process on the OLED.

This simple idea is not as easy to realise in ESPHome as one could think, as there is no relation between the order of statements in the YAML file and the order of execution. Everything is happening in a somewhat asynchronous way, or at least it seems so.

The keywords “on_boot” and “priority” help here.

In the on_boot section there are wait_until statements, we wait until first the WiFi is connected, and second there is a connection to HA. Priority 800 means the very beginning of initialization.

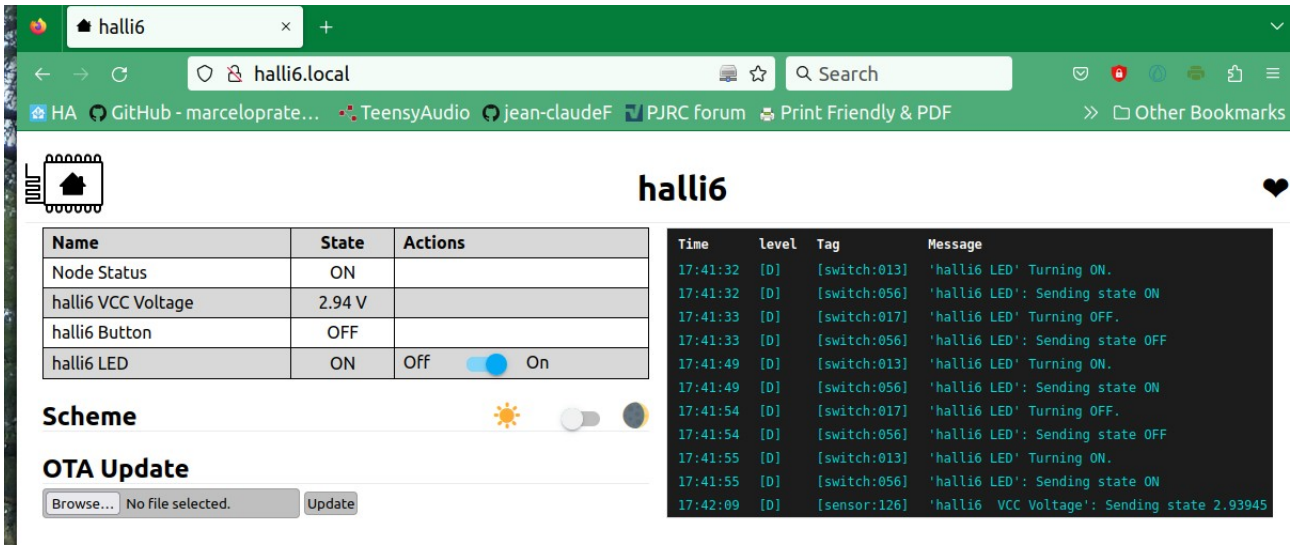
I wanted to add an indication “Booting” at the very start (page 0), but this didn’t work as the hardware isn’ initialized at that moment. That’s why these statements are commented out.

```
substitutions:
  devicename: halli6

esphome:
  name: $devicename
  on_boot:
    priority: 800
    then:
      #- display.page.show: page0
      #- component.update: mydisplay
      #- delay: 1s

      - wait_until:
          wifi.connected:
      - display.page.show: page1
      - component.update: mydisplay

      - wait_until:
          api.connected
```

```
      - display.page.show: page2
      - component.update: mydisplay


# ...


display:
  - platform: ssd1306_i2c
    model: "SH1106_128X64"
    address: 0x3C
    id: mydisplay
    update_interval: never

    pages:
      - id: page0
        lambda: |-
          it.print(0, 10, id(myfont), "Booting");
      - id: page1
        lambda: |-
          it.print(0, 10, id(myfont), "WiFi ON");
      - id: page2
        lambda: |-
          it.print(0, 10, id(myfont), "HA ON");
```

3. A webserver in 2 lines of code

The interesting contribution https://tech.scargill.net/my-esphome-adventure/ told me that I can very easily add a web server to my project (here an example with LED, button and adc for the supply voltage):



And this is simply done by adding

```
web_server:
    port: 80
```

to the YAML file.

The server is available through <name>.local in the browser.

4. Showing WiFi information in HA

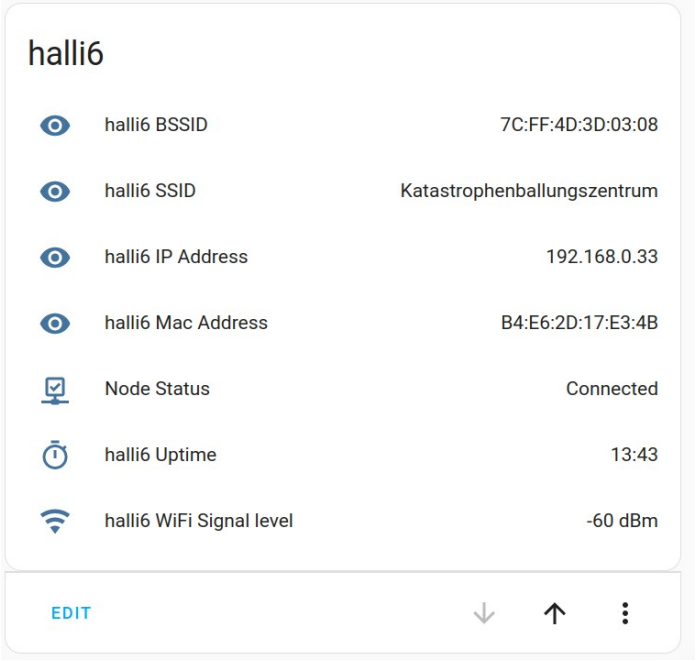
Interesting WiFi information (for the ESP) can be shown in HA.

The new items wifi_signal and uptime are added to the sensor section.
(Take care: only one sensor section is allowed, so add the items to an eventually existing sensor section, or create a new one):

```
sensor:
 - platform: adc
   pin: VCC
   name: $devicename  VCC Voltage
   id: vcc

 - platform: <other sensors...>

 - platform: wifi_signal
   name: "WiFi Signal Sensor"
   update_interval: 15s
   id: sstrength

 - platform: uptime
   name: Uptime Sensor
   id: upt
```

A text sensor sends information about IP address, SSID, BSSID and MAC address to HA.

```
text_sensor:
  - platform: wifi_info
    ip_address:
      name: $devicename IP Address
      id: ipaddr
    ssid:
      name: $devicename   SSID
    bssid:
      name: $devicename   BSSID
    mac_address:
      name: $devicename  Mac Address
      id: mac
```



5. Using node status and C code to display WiFi info

We have already used boot priority and multiple pages to show the status on the display.

There is another way to do it that is maybe even more elegant.

There is a binary sensor that gives the status:

```
binary_sensor:
  - platform: …
    ...

  - platform: status
    name: "Node Status"
    id: system_status
```

(Remember that there has to be only one paragraph “binary_sensor” that groups all sensors with 2 states, so we may have GPIO pins as input and other thins in this paragraph.)

Here https://tech.scargill.net/my-esphome-adventure/ is a nice way to use this information on the display:

```
display:
  - platform: ssd1306_i2c
    model: "SH1106_128X64"
    address: 0x3C
    update_interval: 5s
    id: mydisplay
    lambda: |-
      it.rectangle(0,  0, 128, 64);
      it.rectangle(0,  0, 128, 16);
      if (id(system_status).state) {
        it.print(124, 2, id(myfont), TextAlign::TOP_RIGHT, "Online");
        it.printf(0, 20, id(myfont), "CH0=%2.3fV", id(vcc).state);
        it.strftime(0, 40, id(myfont), "%H:%M:%S", id(esptime).now());
      }
      else {
        it.print(124, 2, id(myfont), TextAlign::TOP_RIGHT, "Offline");
      }
```

The lambda function is written in C, ssso here you can write or draw anything as you would do in Arduino C.

6. Complex programming / if / for / while / repeat etc.

There are several ways to do more complex programming using control structures. We can write the code as lambda in C, as shown in the last chapter. Or we can use ESPHome’s syntax and write the same in YAML code.

Info is here:

https://esphome.io/guides/automations.html

It was irritating for me to find basic concepts of a programming language (and this is one, right?) like if, while, repeat, wait_until, delay … under “Automations and templates”. But that’s the place you have to look for these.

Important concepts:

- **Triggers** (event driven actions) like **on_**press, when a button is pushed
  https://esphome.io/guides/automations.html?highlight=automation#global-variables
- **Actions** like output.turn_off or output.turn_on
  But also mqtt_publish, execute_script, deep_sleep.enter, servo_write, uart.write, http_request.get

  But in this chapter you find also
  delay, lambda, if, while, wait_until
  number.set, number.increment …

  https://esphome.io/guides/automations.html?highlight=automation#all-actions

- **Conditions**
  lambda condition, **if** action (under lambda condition)
  for
  switch.is_on
  time.has_time
  text_sensor.state
  number.in_range
- **Lambda calls execute C code**
- **Scripts** (with and without parameters) can group a part of code.
  They would be called **functions** in other programming languages.
- Global variables
  https://esphome.io/guides/automations.html?highlight=automation#global-variables

All of these automations works with and without connection to HA.
It is comparable to programming the controller in Arduino C.

7. Connecting an ADC: ADS1115

https://esphome.io/components/sensor/ads1115.html#id1

The ADS1115 is a component I like very much. It has 4 channels that have a very high precision, unlike the internal ADCs of the ESPs.

It is connected via I2C. First we have to define the address:

```
ads1115:
- address: 0x48
```

There are 4 possible addresses that depend on the connection of the address pin.

ADR - GND	0x48
ADR - VCC	0x49
ADR - SDA	0x4A
ADR - SCL	0x4B

In the sensor section we add the code for the ADS1115:

```
sensor:
  - platform: …
    ...

  - platform: ads1115
    multiplexer: 'A0_GND'
    gain:  2.048
    name: "ADS1115 CH0"
    id: CH0
```

The gain parameter can be set to

```
0.256
0.512
1.024
2.048
4.096
6.144
```

These values define the measurement range (e.g. gain = 2.048 → range 0...2.048V)
The ranges 4.096 and 6.144 are possible, but anyway the maximum input voltage is 3.3V, so they are not so useful.

This makes CH0 available to HA.
It can also be diplayed on the OLED

```
display:
  - platform: ssd1306_i2c
```

```
    ...
    lambda: |-

      if (id(system_status).state) {
        ...
        it.printf(0, 36, id(myfont), "VCC=%2.3fV", id(CH0).state);

      }
      else {
        it.print(124, 2, id(myfont), TextAlign::TOP_RIGHT, "Offline");
      }
```

By the way, a fact that is not known to everybody is that the ADS1115 **can convert negative input values**, even in one-channel mode. The range is however restricted to about -0.2V. This is due to the input protection diodes that begin to conduct at that value (see datasheet).

But it is interesting for measuring bipolar currents with a shunt resistor.

8. Manipulating sensor values

Templates and filters allow to change the value a sensor outputs

https://esphome.io/components/sensor/template.html

https://esphome.io/components/sensor/index.html#sensor-filters

Example:
I have a 47k/2.7k voltage divider before the input of the ADC. So the real voltage is 49.7/2.7 times the ADC value.

This is done by a simple multiply filter with a small offset to set zero:

```
sensor:
  - platform: ads1115
    multiplexer: 'A0_GND'
    gain: 2.048
    update_interval: 1s
    name: "ADS1115 CH0"
    id: CH0
    filters:
    - offset: -0.001
    - multiply: 18.4074
```

There are many other (some more complex) possibilities like

offset, calibrate_linear, calibrate_polynomial,
min, max,
linear + exponential **moving average**,
throttle = reduce publishing speed of the values,
heartbeat = send the values at regular intervals,

debounce,
delta = send the difference from last value

And for custom filters, program them yourself in C: lambda

For example to convert °C to Fahrenheit:

```
filters:
  - lambda: return x * (9.0/5.0) + 32.0;
unit_of_measurement: "°F"
```

9. Links

Guides:
https://esphome.io/guides/

Components index:
https://esphome.io/index.html

FAQ:
https://esphome.io/guides/faq.html#tips-for-using-esphome

Display:
https://esphome.io/components/display/index.html#
https://esphome.io/index.html#display-components

Other info:

https://esphome.io/guides/configuration-types.html

https://www.youtube.com/watch?v=a3iay-g1AsI

https://tech.scargill.net/my-esphome-adventure/