ESP32forth and Arduino C++ - notes about ESP32 and OpenWeatherMap forecast.

ESP32forth is written in Arduino C and some knowledge of C is very helpful. As amateur programmer, with basic knowledge of Forth only, I resolved to learn also basics of Arduino C to be able to better understand and use ESP32forth. I created some notes about my examples to help others in the same situation.

ESP32 chip has WiFi onboard and ESP32forth support for it, so it is possible access internet data. My experiments are based on free texts on excellent web site *https://randomnerdtutorials.com/* . For practical usage of tool for JSON data I created ESP32forth program for retrieving weather forecast data from OpenWeatherMap.org web, which are delivered in JSON format. The goal is create simple stand alone construction to show weather information on E-paper 7 inch display. Presented program is first part of project, second part will be program for transformation of forecast to display on E-paper screen. Idea is put ESP32 into deepsleep and in some intervals wake it up to connect to OWMap web, download forecast, show on E-paper and go to sleep again.
Full files from my article are located on github, in article are shortened parts:
https://github.com/Vaclav-Poselt/ESP32_forth_code

At first it was necessary to create some code for retrieving desired information from JSON file from OpenWeatherMap, they deliver free 3 hour 5 days forecast, so good amount of data. For access it is necessary at first obtain free API key. For parsing forecast file it was to write some JSON forth parser or use ArduinoJson C library. My current effort is to learn some C so I use second option. I created userwords.h file to implement new words tailored for this JSON forecast + http access..

Stack diagrams for new JSON words available now in ESP32forth:

```
JsonDocument doc;       // doc variable for ArduinoJSON
JsonDocument filter;    // filter variable for deserialization filtering
/*
JSON.filter ( z-filter errorbuf ---z-errorbuf ) create filter for deserialization
JSON.deserialize ( z-payload errorbuf---z-errorbuf ) create deserialized doc
   from z-payload, in z-errorbuf is "OK" or error text )
JSON.Fdeserialize ( z-payload errorbuf---z-errorbuf ) create deserialized
   filtered doc from z-payload and filter, in z-errorbuf is "OK" or error text ),
   use only after JSON.filter
JSON.serialize ( size buffer---n z-buffer ) create new z-buffer string from doc,
   primary for test of result after JSON.Fdeserialize, size is size of buffer,
   n is no of written bytes into buffer, if n>size there is error-too big for stringbuf
JSON.getString ( stringbuf stringbuffersize z-key---z-stringbuf ) retrive string key value
   to stringbuf
JSON.getNum ( z-key--- n) retrive integer key value, also for bool as 1/0
JSON.get2String ( stringbuf stringbuffersize z-key1 z-key2---z-stringbuf ) retrive
    string from 2 keys object to stringbuf
JSON.get2Num (z-key1 z-key2---n ) retrive integer value from 2 keys
   object, also for bool as 1/0
JSON.size ( ---n ) return no of elements in the JSON doc after deserialization
JSON.getFNum ( z-key---<fp>) retrive float key value to floating point stack
JSON.get2FNum (z-key1 z-key2---<fp>) retrive float value from 2 keys
   object to fp stack
JSON.get3String ( stringbuf stringbuffersize z-key1 index z-key2---z-stringbuf )
JSON.get3Num (z-key1 index z-key2---n )
JSON.get2FNum (z-key1 index z-key2---<fp>)
```

JSON.get4FNum (z-key1 index z-key2 key3---<fp>)
JSON.get4Num (z-key1 index z-key2 key3---n)
JSON.get5Num (z-key1 index1 z-key2 index 2 key3---n)
JSON.get5String (stringbuf stringbufsize z-key1 index1 z-key2 index2 key3---z-stringbuf)

note: this JSON.getxxxs have simple error checking for errors from doc[] with not existent keys: returned string "NOTFOUND", returned number 0, 0.00 on f-stack
*******/*
These words are based on analysis of what types json key:value pairs is necessary to use:
- key->string getString
- key->number getNum
- key key->string (city name) get2String
- key key->number (city sunrise) get2Num
- key n key->num (list 0 visibility) get3Num n is array index
- key n key key ->float (list 0 main temp) get4FNum
- key n key->string (list 0 dt_txt) get3String
- key n key key ->num (list 0 main pressure) get4Num
- key n key n key->num (list 0 weather 0 id) get5Num
- key n key n key->string (list 0 weather 0 description) get5String

Obtained individual forecast values with *JSON.getxxx* words are returned as text in c-type format (string with trailing 0), integer on forth stack or float on forth float stack.
After compilation of modified ESP32forth with new words it is possible to start with forth code.

```
 6
 7  s" /spiffs/strings.fs" included \ strings functions
 8
 9  decimal forth only wifi
10
11  create payload 16384 allot       \ buffer for getString and deserialize
12      payload 16384 erase
13  create errorbuf 120 allot        \ buffer for deserialize error
14      errorbuf 120 erase
15  create stringbuf 120 allot       \ buffer for getString
16      stringbuf 120 erase
17  120 string serverPath            \ variable for HTTP.begin
18  6 string City                    \ city variable
19  40 string OWMapApiKey            \ variable for API key
20  20 string units                  \ variable for units
21                                   \ next fill string variables for serverPath
22  s" http://api.openweathermap.org/data/2.5/forecast?q=" serverPath $!
23  s" Prague" City $!
24  s" &YOUR API KEY" OWMapApiKey $!
25  s" &units=metric&cnt=3" units $!    \ cnt=3 timestamps for first testing
26
27  City serverPath append$
28  OWmapApiKey serverPath append$
29  units serverPath append$         \ construct full serverPath
30
```

On line 11 is created *payload* space for retrieved JSON data. String variable *serverPath* contains text for call to OpenWeatherMap. On line 25 is added variable &cnt=3 for first testing to obtain only 3 blocks of 3 hour forecast, for full forecast remove this and obtain full 40 blocks.

Next is wifi connection and fill *payload* with forecast.

```
: ztype ( c-addr---)            \ helping word to type z-string
    z>s type
;
: s120  ( ---stringbuf 120 )        \ helping word
    stringbuf 120
;
: wificonnection    ( -- )          \ connect to my wifi
    z" YOUR SSID" z" YOUR PSW" login

;

: retriveFORE   ( z-serverPath--- z-payload) \ retrive 5 days forecast from OWMap
    HTTP.begin
    if cr ." connected to OpenWeatherMapAPI"
        else
        cr ." no connection to OWMAPI"
    then
    begin
    200 ms
    HTTP.get    \ send GET
        dup cr ." GET response: " .
    0 > until
    payload 16384 erase      \ clear payload buffer
    payload 16384 HTTP.getString ( --- z-payload) \ read weather forecast JSON
    dup cr ." obtained payload:" cr ztype
    http.end                \ close connection
    wifi.disconnect         \ end wifi connection
;
```

For direct access to key:value pairs from json data it is to create doc data in C code, so do deserialization of *payload*.

```

: retriveJSONdata ( z-payload--- )  \ create json doc from received data
    errorbuf JSON.deserialize
    cr ." deserialization error: " ztype \ if correctly created doc shows OK
;
```

When this runs correctly it is possible to obtain forecast values for all 40 3 hour forecast blocks for next processing of forecast. This json elaboration is explained on web ArduinoJson.org for this C library.
For test of usage there is word *printnForecast* to print some usefull data from forecast no 0-39 in this 5 days data. It uses values from this C *doc* text not directly accessible from forth code, only with use of this new words calling C functions from ArduinoJson library. On next page is forth code for this output.

```
: printnForecast { n1 }         \ print n1th 3h forecast from 0-39 5days forecast
   2 set-precision              \ set 2 decimal places for f.
   s120 z" city" z" name" JSON.get2String
   cr ." Weather forecast for city: " ztype
   s120 z" list" n1 z" dt_txt" JSON.get3String
   cr ." Forecast for date time: " ztype
   z" list" n1 z" main" z" temp" JSON.get4FNum
   cr ." temperature deg C: " f.
   z" list" n1 z" main" z" pressure" JSON.get4Num
   cr ." air pressure hPa: " .
   z" list" n1 z" main" z" humidity" JSON.get4Num
   cr ." humidity %: " .
   s120 z" list" n1 z" weather" 0 z" description" JSON.get5String
   cr ." weather description: " ztype
   z" list" n1 z" wind" z" speed" JSON.get4FNum
   cr ." wind speed m/sec: " f.
   z" list" n1 z" rain" z" 3h" JSON.get4FNum
   cr ." rain mm for last 3h: " f.
   z" list" n1 z" snow" z" 3h" JSON.get4FNum
   cr ." snow mm for last 3h: " f. cr
;
```

In ArduinoJson library is also interesting possibility to filter json data during deserialization to create smaller doc data. In weather forecast there are some key:data values which are not interesting for my weather forecast. So it is possible to remove them during deserialization with help of special filter string. I was interested in it , but I dont know if it will be used in my application. But why not test it?
So there are added 3 new words *JSON.filter, JSON.Fdeserialize* and *JSON.serialize*. With *JSON.filter* is possible to create *filter* C variable from definition string with requested json keys marked as true, this *filter* is than used in deserialization of obtained weather forecast with word *JSON.Fdeserialize*. After it we have C doc variable containing only requested data. For testing there is problem as doc is not directly accessible from forth program, so it is possible to use *JSON.serialize* to make c-string with filtered content of forecast.

OK, next is screenshot from terminal program MyFORTHshell with testing sequence from weather forecast. There is used parameter &cnt=3 to shorten example to only 3 blocks of 3 hour forecast.

```
 ok
--> wificonnection
192.168.1.62
MDNS started
 ok
--> serverpath s>z retrivefore

connected to OpenWeatherMapAPI
GET response: 200
obtained payload:
{"cod":"200","message":0,"cnt":3,"list":[{"dt":1737320400,"main":{"temp":-
1.96,"feels_like":-5.87,"temp_min":-1.96,"temp_max":-
0.71,"pressure":1025,"sea_level":1025,"grnd_level":988,"humidity":94,"temp_kf"
:-1.25},"weather":[{"id":804,"main":"Clouds","description":"overcast
clouds","icon":"04n"}],"clouds":{"all":100},"wind":
{"speed":3.02,"deg":94,"gust":4.96},"visibility":10000,"pop":0,"sys":
{"pod":"n"},"dt_txt":"2025-01-19 21:00:00"},{"dt":1737331200,"main":{"temp":-
1.66,"feels_like":-5.3,"temp_min":-1.66,"temp_max":-
1.07,"pressure":1025,"sea_level":1025,"grnd_level":987,"humidity":93,"temp_kf"
:-0.59},"weather":[{"id":803,"main":"Clouds","description":"broken
clouds","icon":"04n"}],"clouds":{"all":68},"wind":
{"speed":2.8,"deg":93,"gust":4.28},"visibility":10000,"pop":0,"sys":
{"pod":"n"},"dt_txt":"2025-01-20 00:00:00"},{"dt":1737342000,"main":{"temp":-
1.63,"feels_like":-4.01,"temp_min":-1.63,"temp_max":-
1.47,"pressure":1024,"sea_level":1024,"grnd_level":987,"humidity":93,"temp_kf"
:-0.16},"weather":[{"id":802,"main":"Clouds","description":"scattered
clouds","icon":"03n"}],"clouds":{"all":41},"wind":
{"speed":1.75,"deg":95,"gust":3.18},"visibility":10000,"pop":0,"sys":
{"pod":"n"},"dt_txt":"2025-01-20 03:00:00"}],"city":
{"id":3067696,"name":"Prague","coord":
{"lat":50.088,"lon":14.4208},"country":"CZ","population":1165581,"timezone":36
00,"sunrise":1737269505,"sunset":1737300838}} ok
1073670060 --> retrivejsondata

deserialization error: Ok ok
--> 2 printnforecast

Weather forecast for city: Prague
Forecast for date time: 2025-01-20 03:00:00
temperature deg C: -1.63
air pressure hPa: 1024
humidity %: 93
weather description: scattered clouds
wind speed m/sec: 1.75
rain mm for last 3h: 0.00
snow mm for last 3h: 0.00
 ok
-->
```

In the end are printed some data from 3rd block with command *2 printnforecast*.

Next is code for test of filtering of JSON data. Filter string is on line 90 with only 2 keys to obtain, so it is possible to have only 5 days temperature forecast data. For *retriveFORE* is necessary to have active WiFi connection.

```
87  \ to create smaller json doc it can be used filter for desired only data
88  \ here possible testing sequence for use from command line
89  120 string filter    \ variable for JSON filter
90  r| {"list":[{"main":{"temp":true},"dt_txt":true}]}| filter $! \ desired keys
91  filter s>z errorbuf JSON.filter \ create json filter
92  serverpath s>z retriveFORE       \ retrive data from OWMap
93  errorbuf JSON.fdeserialize ztype \ Fdeserialize and print error
94  16384 payload JSON.serialize ztype ( ---n z-payload) \ type content of filtered
95                                      \ doc, n is lenght of created data
96
97
```

And of course screenshot from terminal. again with only 3 blocks of forecast.

```
Ok ok
--> serverpath s>z retriveFORE

connected to OpenWeatherMapAPI
GET response: 200
obtained payload:
{"cod":"200","message":0,"cnt":3,"list":[{"dt":1737320400,"main":{"temp":-
1.98,"feels_like":-5.9,"temp_min":-1.98,"temp_max":-
0.71,"pressure":1025,"sea_level":1025,"grnd_level":988,"humidity":94,"temp_kf"
:-1.27},"weather":[{"id":804,"main":"Clouds","description":"overcast
clouds","icon":"04n"}],"clouds":{"all":100},"wind":
{"speed":3.02,"deg":94,"gust":4.96},"visibility":10000,"pop":0,"sys":
{"pod":"n"},"dt_txt":"2025-01-19 21:00:00"},{"dt":1737331200,"main":{"temp":-
1.68,"feels_like":-5.33,"temp_min":-1.68,"temp_max":-
1.07,"pressure":1025,"sea_level":1025,"grnd_level":987,"humidity":93,"temp_kf"
:-0.61},"weather":[{"id":803,"main":"Clouds","description":"broken
clouds","icon":"04n"}],"clouds":{"all":68},"wind":
{"speed":2.8,"deg":93,"gust":4.28},"visibility":10000,"pop":0,"sys":
{"pod":"n"},"dt_txt":"2025-01-20 00:00:00"},{"dt":1737342000,"main":{"temp":-
1.64,"feels_like":-4.02,"temp_min":-1.64,"temp_max":-
1.47,"pressure":1024,"sea_level":1024,"grnd_level":987,"humidity":93,"temp_kf"
:-0.17},"weather":[{"id":802,"main":"Clouds","description":"scattered
clouds","icon":"03n"}],"clouds":{"all":41},"wind":
{"speed":1.75,"deg":95,"gust":3.18},"visibility":10000,"pop":0,"sys":
{"pod":"n"},"dt_txt":"2025-01-20 03:00:00"}],"city":
{"id":3067696,"name":"Prague","coord":
{"lat":50.088,"lon":14.4208},"country":"CZ","population":1165581,"timezone":36
00,"sunrise":1737269505,"sunset":1737300838}} ok
1073670060 --> errorbuf JSON.fdeserialize ztype
Ok ok
--> 16384 payload JSON.serialize ztype
{"list":[{"main":{"temp":-1.98},"dt_txt":"2025-01-19 21:00:00"},{"main":
{"temp":-1.68},"dt_txt":"2025-01-20 00:00:00"},{"main":{"temp":-
1.64},"dt_txt":"2025-01-20 03:00:00"}]} ok
175 -->
```

If it helps somebody I will be pleased, if there are errors in my explanation I will be also pleased to be corrected.
Files from my article are located on github:
https://github.com/Vaclav-Poselt/ESP32_forth_code
Final note: tested on ESP32forth 7.0.7.20, Arduino IDE2.3.2, ESP32 lib 2.0.14, ESP32 Dev Module board and ArduinoJson.h ver. 7.