ESP32forth and Arduino C++ - notes about ESP32 WiFi and JSON usage.

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. My experiments are based on free texts on excellent web site *https://randomnerdtutorials.com/* about Scan Wi-Fi Networks and .WorldTimeAPI using ArduinoJSON library. We can start with Scan WiFi networks.
At first I tested Arduino C code for scanning for available WiFi networks, I dont copy code here, all is on randomnerstutorials. After test of C code there was to create similar solution in ESP32forth.
But implementation of WiFi from WiFi.h in ESP32forth.ino is covering only part of functions for connection and communication, not for WiFi scanning. So I created necessary *userwords.h* file to incorporate missing functions for scanning:

```
#include <string.h>
#define USER_WORDS \
X("WiFi.scanNetworks", wifi_scan, PUSH WiFi.scanNetworks();) \
X("WiFi.scanDelete", wifi_scdel, WiFi.scanDelete();) \
X("WiFi.SSID", wifi_ssid, strcpy(c1, WiFi.SSID(n0).c_str()); DROP; ) \
X("WiFi.RSSI", wifi_rssi, n0= WiFi.RSSI(n0);) \
X("WiFi.RSSI1", wifi_rssi1, PUSH  WiFi.RSSI();) \
X("WiFi.channel", wifi_chan, n0= WiFi.channel(n0);) \
X("WiFi.channel1", wifi_chan1, PUSH WiFi.channel();) \
X("WiFi.encryptionType", wifi_encrypt, n0=
static_cast<int>(WiFi.encryptionType(n0));)
```

Stack diagrams for new words:
WiFi.scanNetworks (---n) \ return no of available wifi networks
WiFi.scanDelete (---) \ delete previous scan results
WiFi.SSID (z-textbuffer no---) \ fill z-textbuffer with SSID for no network
WiFi.RSSI (no--- RSSI) \ return RSSI of no network
WiFi.RSSI1 (--- RSSI) \ return RSSI of current WiFi connection
WiFi.channel (no--- channel) \ return channel for no network
WiFi.channel1 (--- channel) \ return channel of current WiFi connection
WiFi.encryptionType (no--- en) \ return enumeration no of encryption for no network

Small note.
Creation of X macros for *userwords.h* is not so easy for C beginner as me. Here I use special AI tool for Arduino ESP32 C++. This is invaluable to me, in seconds I have recomendation for solution including C++ sample .ino file, analyze of parts of c code, helping in my battle with this static char* or xxx.c_str() asp. I use paid version, but this 2 beers per month price is well-deserved.

OK, so next ESP32forth program for WiFi scanner.

```
3
4  defined? --WIFIsc 0<> [if] forget --WIFIsc [then]
5  create --WIFIsc
6  decimal forth only wifi
7
8  : setup WIFI_MODE_STA WiFi.Mode          \ adjust station mode of wifi
9  WiFi.Disconnect                  \ and disconnect  for sure
10 cr ." Basic setup done" 500 ms
11 ;
12 : printHeader                    \ print header of table
13 ." No.  SSID                                RSSI   Ch   Encryption" cr
14 ;
15 : .r ( n1 n2-- )                 \ print n1 justified to n2 places
16     >r str r> over - spaces type \ not ANSI
17 ;
18
19 create SSIDbuf 32 allot SSIDbuf 32 0 fill \ 32 bytes for SSID z-string
20
21 : scanNetworks ( ---n ) \ search for wifi networks, return no of networks
22     cr ." Now start wifi scanning"
23     WiFi.scanNetworks cr ." Scanning finished"
24     dup                              \ no of found networks
25     0= if cr ." No networks found" drop
26         else dup cr . ." Networks found." cr
27         then
28 ;
29
30 : mainloop ( --- )
31     setup
32     begin                            \ start repeating scanning loop
33         scanNetworks                 \ (--n) no of found netw
34         printHeader
35          0 do                       \ print results of scanning from 0!
36             i 2 .r ."  | "           \ nr.
37             SSIDbuf i WiFi.SSID      \ ( ---ssidbuf ) read SSID into buf
38             z>s dup >r type \ print SSID
39             32 r> - spaces           \ adjust to lenght 32
40             ."  | "
41             i WiFi.RSSI 4 .r         \ print RSSI
42             ."  | "
43             i WiFi.channel 2 .r      \ print channel
44             ."  | "
45             i WiFi.encryptionType    \ returns ordinal no of enumeration
46                                      \ type wifi_auth_mode_t print encryption
47             case
48                 0 of s" open"        endof
49                 1 of s" WEP"         endof
50                 2 of s" WPA"         endof
51                 3 of s" WPA2"        endof
52                 4 of s" WPA+WPA2"    endof
53                 5 of s" WPA2-EAP"    endof
54                 6 of s" WPA3"        endof
55                 7 of s" WPA2+WPA3"   endof
56                 8 of s" WAPI"        endof
57             endcase
58             type cr 10 ms
59             loop
60     WiFi.scanDelete                  \ clear scan results
61     2000 ms
62     key? until
63     ;
```

Only short comments about program:
setup adjust station mode, *SSIDbuf* is 32 bytes buffer for max 31 bytes of text+trailing 0 of z-string.

scanNetworks do full scan and returns no of found available WiFi networks. The rest is done by word *mainloop* using new functions from *userwords.h*, where WiFi.SSID fills SSIDbuf with SSID name of network in c-type string, so for print is used *z>s* to create *addr len* forth format. After print there are deleted scan results and WiFi scan is after 2 seconds repeated.
Result of scan where only one network was found:

```
1 networks found:

Nr | SSID                             | RSSI | CH | Encryption
 1 | Viska123                         |  -53 |  1 | WPA2
```

In next part about program for obtaining of exact time from internet using *WorldTimeAPI.*
Again first I started with Arduino C program. Arduino C source is on above mentioned web *randomnerdtutorials* , search for key word WorldTimeAPI and in C code find function *void get_date_and_time()*, where is all code for retrieving date and time.
C program uses 3 C libraries - *<WiFi.h>, <HTTPClient.h> and <ArduinoJson.h>*.
For use in ESP32forth WiFi functions are included, so we need some functions from *HTTPclient.h* to do connection to server WorldTimeAPI and retriving desired date time information. This data are returned in JSON format, which is not supported by ESP32forth. But this is plain text format, so with some string functions is not difficult to filter desired text data. Other possibility is to incorporate JSON serialize/deserialize functions from ArduinoJson.h. At first sight it is overkill , but JSON format is used in internet communication for IoT, so I decided to implement it. Here I will use for short only few functions for retriving data from JSON and not for creating (serialization) of JSON. So again start with *userwords.h* file creation. Here is first part for HTTPClient.

```
#include <HTTPClient.h>
#include <ArduinoJson.h>
HTTPClient http;
JsonDocument doc;   // doc variable for ArduinoJSON
#define USER_WORDS \
X("HTTP.begin", httpbegin, n0=http.begin(c0); ) \
X("HTTP.get", httpget, PUSH http.GET(); ) \
X("HTTP.getString", httpgetstring, const char* zpay=http.getString().c_str(); \
    strlcpy(c1, zpay, n0); DROP;) \
X("HTTP.end", httpend, http.end(); ) \
and stack diagrams
```

HTTP.begin (z-url---error-t/f) open connection with url, returns t/f
http.get (---n) send GET , return negative as false/code, code is 200 as OK, 404 ...
http.getString (payloadbufer size--- payloadbuffer) reads respond max size bytes into z-payloadbuf
http.end (---) ends connection with url

Function for obtaining payload requires some buffer, I use *create payload 1024 allot* . But obtained payload can be longer, so second input parameter is size of buffer. *http.getString* fills buffer with maximum size-1 characters and not used last bytes of buffer previously erased create trailing 0 for z-string.
And second part for JSON.

```
X("JSON.deserialize", jsondeserialize, DeserializationError error = deserializeJson(doc, c1); \
    strcpy(c0, error.c_str()) ; NIP ) \
X("JSON.getString", jsongetstring, const char* retrievedValue=doc[c0].as<const char*>(); \
     strlcpy(c2, retrievedValue, n1); DROPn(2); ) \
X("JSON.getNum", jsongetnum, n0=doc[c0]; )
```

and stack diagrams

JSON.deserialize (z-payload errorbuf---z-errorbuf) create deserialized doc
from z-payload, in z-errorbuf is "OK" or error text
JSON.getString (stringbuf stringbuffersize z-key---z-stringbuf) retrive string key value
to stringbuf
JSON.getNum (z-key--- n) retrive numerical key value, also for bool as 1/0

If interested in details of JSON there is excellent helping web arduinojson.org with detail explanation from creator of this library.
OK, so go to ESP32forth code for time retrieving /on next page/. Program is simple, with comments, so self explanatory. Last word *retriveJSONdata* pulls out key values from doc variable.

Program runs well, biggest problem is web api WorldTimeAPI itself. It looks overloaded with GET requests from whole world, so it is to be patient and wait for returned payload. It could take even few minutes of waiting.

```
defined? --http 0<> [if] forget --http [then]
create --http
decimal forth only wifi

create payload 1024 allot       \ buffer for getString and deserialize
    payload 1024 erase
create errorbuf 120 allot       \ buffer for deserialize error
    errorbuf 120 erase
create stringbuf 120 allot      \ buffer for getString
    stringbuf 120 erase

: wificonnection    ( -- )          \ connect to my wifi
    z" TUKKUK1" z" SHX3U21a" login
;

: retriveJSON   ( --- z-payload) \ retrive payload from WorldTime
    z" http://worldtimeapi.org/api/timezone/Europe/Prague" HTTP.begin
    if cr ." connected to WorldTime"
        else
        cr ." no connection to WorldTime"
    then
    begin
    200 ms
    HTTP.get    \ send GET
        dup cr ." GET response: " .
    0 > until
    payload 1024 erase      \ clear payload buffer
    payload 1024 HTTP.getString ( --- z-payload)
    dup cr ." obtained payload:" cr z>s type
;
: retriveJSONdata ( z-payload--- )
    errorbuf JSON.deserialize
    cr ." JSON deserialize error: " z>s type
    stringbuf 120 z" datetime" JSON.getString
    cr ." retrived datetime: " z>s type
    stringbuf 120 z" timezone" JSON.getString
    cr ." retrived timezone: " z>s type
    stringbuf 120 z" client_ip" JSON.getString
    cr ." my IP address: " z>s type
    z" day_of_week" JSON.getNum
    cr ." day of week: " .
;
```

And next typical session from MyFORTHshell terminal program. At first connection to local WiFi, at second retrieving of JSON data. Here it was short waiting, but it could take also minutes. And at last decoding parts of JSON.

```
 ok
--> wificonnection
192.168.1.47
MDNS started
 ok
--> retriveJSON

connected to WorldTime
GET response: -5
GET response: -5
GET response: -5
GET response: -5
GET response: -5
GET response: -5
GET response: -5
GET response: -5
GET response: -5
GET response: -5
GET response: -5
GET response: -5
GET response: -5
GET response: -5
GET response: -5
GET response: 200
obtained payload:
{"utc_offset":"+01:00","timezone":"Europe/Prague","day_of_week":5,"day_
of_year":355,"datetime":"2024-12-
20T21:37:23.377622+01:00","utc_datetime":"2024-12-
20T20:37:23.377622+00:00","unixtime":1734727043,"raw_offset":3600,"week
_number":51,"dst":false,"abbreviation":"CET","dst_offset":0,"dst_from":
null,"dst_until":null,"client_ip":"109.81.167.145"} ok
1073669032 --> retriveJSONdata

JSON deserialize error: Ok
retrived datetime: 2024-12-20T21:37:23.377622+01:00
retrived timezone: Europe/Prague
my IP address: 109.81.167.145
day of week: 5  ok
-->
```

This programming create question if this is good way - use external ready made C libs. Of course more experienced forth user can do all in pure forth. This has advantage one has full control on each byte of code. But usage of external C libs is quicker, also ESP32forth itself is mixture of forth and C libraries. It is possible to little exaggerate and tell " One can do everything in ESP32forth - if there is C library for it".

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 git:
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.