ESP32forth and Arduino C++ - notes about ESP32 preferences.

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. First part was in article ESP32forth and Arduino C.pdf saved on this facebook

ESP32 chip has on-board non-volatile memory (NVS) to store data. This data are retained across restarts and loss of power events to the system. Technicaly data are stored to SPI Flash memory, so access is not so easy, but there is special library for it *preferences.h* usable in Arduino C++ code. Data are organized in so called namespaces, each namespace can contain pairs key-value. For usage of this system in Arduino C there are prepared C functions in this library. I have used for my tests C examples from original Espessif documentation, located in tutorial https://docs.espressif.com/projects/arduino-esp32/en/latest/tutorials/preferences.html
and API https://docs.espressif.com/projects/arduino-esp32/en/latest/api/preferences.html. So I don't copy Arduino C code here, everything is in mentioned documentation. Preferences storage is suitable for saving of small amount of data as for example SSID, password, simple configuration data. For longer parts as program modules or bigger measurement values there is better use file system as SPIFFS.
For usage in ESP32forth is necessary create new words using this functions from C library. I use simple method creating file *userwords.h* with X/Y macros, which is compiled together with ESP32forth.ino file in Arduino. This method is described on Brad Nelsons web page for ESP32forth https://esp32forth.appspot.com/ESP32forth.html in the end of text. For testing of preferences I created new forth words only for part of functions from *preferences.h*. Instead of ESP32forth words the names for namespaces and keys are case sensitive. True value is represented by 1, false 0. So here are new words from file *userwords.h*:

```
#include <Preferences.h>
Preferences preferences; // global instance of Preferences

#define USER_WORDS \
   X("preferences.begin", pre_begin, {  n0 = preferences.begin(c1, b0); NIP;} ) \
   X("preferences.end", pre_end, preferences.end();) \
   X("preferences.clear", pre_clear, { PUSH preferences.clear(); } ) \
   X("preferences.isKey", pre_isKey, { n0 = preferences.isKey(c0); } ) \
   X("preferences.remove", pre_remove, { n0 = preferences.remove(c0);} ) \
   X("preferences.putInt", pre_putInt, { n0 = preferences.putInt(c1, n0); NIP;} ) \
   X("preferences.getInt", pre_getInt, { n0 = preferences.getInt(c0);} ) \
   X("preferences.putString", pre_putString, { n0 = preferences.putString(c1, c0); NIP; } ) \
   X("preferences.getString", pre_getString, { n0 = preferences.getString(c2, c1, n0); NIPn(2);} ) \
   X("preferences.putBytes", pre_putBytes, { n0= preferences.putBytes(c2, b1, n0); NIPn(2);} ) \
   X("preferences.getBytes", pre_getBytes, { n0= preferences.getBytes(c2, b1, n0); NIPn(2);} ) \
   X("preferences.getBytesLength", pre_getBytesLength, { n0= preferences.getBytesLength(c0);} )
```

And necessary explanation of usage with stack diagrams:

```
/*
* preferences.begin ( name-z false --- t/f ) open namespace name-z, false is R/W, true is R only
*   returns true- success false- error, true here is 1
*   namespace and key can have max 15 chars, only one namespace can be open at a time
* preferences.end ( --- ) close current namespace
* preferences.clear ( --- t/f ) clears all keys in current namespace
* preferences.isKey ( key-z --- t/f ) test existence of key in current namespace
* preferences.remove ( key-z --- t/f) remove key from current namespace
* preferences.putInt ( key-z n --- 4/0 ) store a value to key, 4- success, 0 error
* preferences.getInt ( key-z --- n ) retrive value from key
* preferences.putString ( key-z addr -- len/0 ) store null term. string to key, return saved len
* preferences.getString ( key-z addr len -- len/0 ) retrive C-style string into buffer with addr and len
* preferences.putBytes ( key-z addr len --- len/0 ) store len bytes starting on addr to key, returns 0 if
error
* preferences.getBytes ( key-z addr len --- len/0 ) retrive len bytes to addr from key, returns 0 if error
* preferences.getBytesLength ( key-z --- len/0 ) get no of stored bytes from key of type byte
*/
```

Names of namespaces and keys are strings in C-type with trailing 0. ESP32forth has necessary support for it with words z" , Z>S , S>Z . After successful compilation in Arduino environment there is possible to start testing new words interactively.

```
 ok
--> 1 constant RO_MODE
 ok
--> 0 constant RW_MODE
 ok
--> z" NAMESPACE" RW_MODE preferences.begin
 ok
1 -->
```

Here is opened/created namespace NAMESPACE responding with 1 – true success.

```
 ok
--> z" INT1" 123654 preferences.putInt
 ok
4 --> z" INT2" 7896 preferences.putInt
 ok
4 4 -->
```

Storage of 2 integer numbers, respond 4 means success, 4 bytes integer stored.

```
 ok
--> preferences.end
 ok
--> z" INT1"  preferences.getInt
 ok
0 -->
```

Close namespace and try to read back – error, namespace not opened. So once more better.

```
 ok
--> z" NAMESPACE" RO_MODE preferences.begin
 ok
1 --> z" INT1"  preferences.getInt
 ok
1 123654 --> z" INT2"  preferences.getInt
 ok
1 123654 7896 -->
```

Now it is correct, namespace opened in RO mode. Next usage of strings after reopening namespace in RW mode..

```
 ok
--> create MyBUFF 32 allot MyBUFF 32 255 fill \ helping 32 bytes buffer
 ok
--> z" STRING1" z" I am STRING1" preferences.putString
 ok
12 --> z" STRING1" MyBUFF 32 preferences.getString
 ok
12 13 --> MyBUFF 32 dump


--addr---  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  ------chars-----
3FFE-E4A0  49 20 61 6D 20 53 54 52 49 4E 47 31 00 FF FF FF  I am STRING1....
3FFE-E4B0  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ................
3FFE-E4C0  53 54 52 49 4E 47 31 00 49 20 61 6D 20 53 54 52  STRING1.I am STR

 ok
12 13 -->
```

MyBUFF is starting at address E4A0, on E4AC there is visible trailing 0 of C type string.
Last example is about saving of bytes array. For this we create MyBYTES 7 bytes array, save it to key MyBYTES1 with respond 7 as successful save.

```
 ok
--> MyBUFF 32 254 fill
 ok
--> create MyBYTES 65 c, 66 c, 67 c, 68 c, 220 c, 230 c, 240 c,
 ok
--> z" BYTES1" MyBYTES 7 preferences.putBytes
 ok
7 -->
```

And finaly read BYTES1 back into MyBUFF.

```
 ok
--> z" BYTES1" MyBUFF 7 preferences.getBytes
 ok
7 --> mybuff 32 dump


--addr---  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  ------chars-----
3FFE-E4A0  41 42 43 44 DC E6 F0 FE FE FE FE FE FE FE FE FE  ABCD............
3FFE-E4B0  FE FE FE FE FE FE FE FE FE FE FE FE FE FE FE FE  ................
3FFE-E4C0  53 54 52 49 4E 47 31 00 49 20 61 6D 20 53 54 52  STRING1.I am STR

 ok
```

Of course data in namespaces are stored permanently, possible to test it with power off.
This is end of my explanation focused to beginners as me, if it helps somebody I will be pleased, if there are errors in my explanation I will be also pleased to be corrected.

Final note: tested on ESP32forth 7.0.7.20, Arduino IDE2.3.2, ESP32 lib 2.0.14, ESP32 Dev Module.