Table of Contents

1. Advanced IoT Hardware 2
1.1. Getting familiar with your hardware 2
1.1.1. Target group 2
1.1.2. Prerequisites 2
1.1.3. Scenario 2
1.1.4. Result 3
1.1.5. Start 3
1.1.6. Steps 3
1.1.7. Result validation 5
1.1.8. FAQ 5

1. Advanced IoT Hardware

1.1. Getting familiar with your hardware

Here you will learn technical details on the hardware, you interface via distant laboratory.

1.1.1. Target group

Undergraduate / Bachelor / Engineering Students

1.1.2. Prerequisites

We assume you already know how to:

- handle LCD screen to present information (refer to the scenarios B1 and B2 when in doubt),
- connect to the existing WiFi network: `internal.IOT`,
- additionally, we will ask you to install and use a CoAP client of your choice. We suggest using chrome extension Cooper4Cr, chrome plugin, but any CoAP client will work here. You connect your solution to the private IP of the NodeRED server but the same information is available on the public IP as well, so you can check it from your home.

NodeRED CoAP server is present in the `internal.IOT` network and it is also visible under public address. Refer to the node documentation for the latest information.



Note - information present in source code samples can be not up-to-date - remember to refer to the VREL node documentation for the latest information on IPs, users and passwords for both `internal.IOT` network access and for the NodeRED access.

CoAP server data is:

- IP address (implemented with NodeRED) is: `192.168.90.5` as seen from the `internal.IOT` network, or `157.158.56.54` when accessing from the public space;
- The only implemented URI is **“”** (simply root resource) - for other URIs you will get 4.04 (Not Found) error;
- CoAP port is 5683 (default);

1.1.3. Scenario

I this scenario, once you get connected to the WiFi as AP and then you will make a UDP request using CoAP protocol to the NodeRED server acting as CoAP server. Please note, we do not implement here discovery services for CoAP, for simplicity and because of constrained resources, so you won't be able to use clients' CoAP discover feature. You will obtain a secret code from the server (it varies over

time). As a bonus, you may try to discover the secret code generation algorithm 🙂. You're expected to present the results of the CoAP query on the LCD screen. To implement a CoAP client feature you will use a dedicated library `ESP-CoAP simple library`, available via Platformio library management system.

1.1.4. Result

You should be able to obtain a UDP message in CoAP, with the payload containing an ASCII text with secret code, and present it on the LCD screen.

1.1.5. Start

Define some identifiers to separate and update AP's SSID and passphrase easily. To simplify conversion, use `IPAddress` class to create IP address of the CoAP server. To format lines for the LCD, we suggest using a char buffer of 20 characters (one full line) and some 2-3 integers for iterators. Remember to declare the LCD control class in your code. You do not need to instantiate WiFi communication class - as you have only one interface here, it is singleton class you can refer directly using WiFi. Reading analogue input brings you an integer value. Note, you're supposed to instantiate your MCU as AP even if using UDP communication.

1.1.6. Steps

Following steps do not present full code - you need to supply missing parts on your own! We do not present here how to connect to the WiFi AP. If you're in doubt, rever to the U1 scenario. Please refer to scenario B1, if you need a recall on how to handle LCD screen.

1.1.6.1. Step 1

Include all necessary libraries. We use ESP-CoAP simple library to contact CoAP server as a client. Note, this library also implements a CoAP server features but we do not use it in our scenario here. The minimum set here is:

```
#include <Arduino.h>
#include <ESP8266WiFi.h>
#include "coap_client.h"
#include <LiquidCrystal_I2C.h>
...
```

Declare some identifiers and variables/class instances to let you easier handle necessary modifications and keep code clear:

```
#define wifi_ssid "internal.IOT"
#define wifi_password "IoTlab32768"

IPAddress ip(192,168,90,5);//take NodeRed Address
```

```
int port = 5683;

...
```

1.1.6.2. Step 2

Declare `coapClient` and WiFi client, to implement CoAP client instance, PubSubClient, initialise, instantiate and connect to the network. If in doubt, refer to the scenario U3 on how to prepare networking code for your solution:

```
...

coapClient coap;

...
```

1.1.6.3. Step 3

Prepare CoAP client response asynchronous handler.



CoAP uses UDP for communication, so you will be informed asynchronously, when response to your request appears. That is fully handled by asynchronous response handler and mechanism is implemented by the library.



As messages may come quickly in the queue (many replies) due to the heavy use or UDP nature, you are required to make a copy of the payload ASAP, best using `memcpy`, as first operation in your asynchronous response handler.

```
void callback_response(coapPacket &packet, IPAddress ip, int port);
...
// coap client response callback
void callback_response(coapPacket &packet, IPAddress ip, int port) {
    char p[packet.payloadlen + 1];                  //declare local buffer
for the payload
    memcpy(p, packet.payload, packet.payloadlen);  //quickly copy payload
contents
    p[packet.payloadlen] = NULL;

    //response from coap server - check if this is regular one (response to
your request), or is it a ping request?
    if(packet.type==3 && packet.code==0){
      //that means you've obtained a ping request so handle it (i.e. show on
LCD)
      ...
    }
```

```
    ... //handle payload.
}
...
```

1.1.6.4. Step 4

Start your CoAP client in the `setup()` function:

```
... // remember to initialise it AFTER you make a WiFi connection, not
before
  coap.respinse(callback_response); // add response handler
  coap.start(); //start coap client
```

1.1.6.5. Step 5

Your `loop()` function should include call to the CoAP client methods. Here we use only CoAP GET, as this is the only one implemented by the NodeRED server, we provide to you:

```
void loop()
{
  ...
  int msgid = coap.get(ip,port,"");
  bool state = coap.loop();
  ...
  delay(10000);
}
```



Mind to keep a `delay(…)`, not to saturate NodeRED server with issuing dozens of UDP requests. Minimum reasonable delay between issuing consecutive requests is about 500ms.

1.1.7. Result validation

Observe connection progress on the LCD via video stream. Once WiFi is connected, you should be able to see secret code as a return from CoAP server. Note, you may want to implement presentation of the communication status, i.e. present `msgid` and state of the `coap.loop()`.

1.1.8. FAQ

Secret changes over time so do not be surprised that you will obtain different numbers when returning to this exercise or executing in loop for a time. Note, to distinguish from random data, there is always some extra, textual message to let you be sure that server returned a value.

From:
https://dokuwiki.zmitac.aei.polsl.pl/ - **RAU6 Dokuwiki**

Permanent link:
https://dokuwiki.zmitac.aei.polsl.pl/en:edu:zmitac:s2:iot:ait

Last update: **2021/01/05 17:50**