

CoAP via serial – guideline for running example sequence

Prerequisites:

- linux (tested on Ubuntu 18.04 and Raspbian Wheezy)
- GCC compiler (to be honest, almost every “modern” version should work :))
- socat
- curl

I. Before we run everything...

In order to run this application, we need to have devices that will receive and transmit data. We can plug the real ones in (for example: ttyUSB converters) or emulate them in our system. Since the first option requires some modifications in our code (ttyUSB converters are not perfect and different models could require different settings), let’s simply emulate the devices.

We can do it via terminal by calling:

```
socat -d -d pty,raw,echo=0 pty,raw,echo=0
```

The command will create two devices (/dev/pts/...) and print their names (below: /dev/pts/1 and /dev/pts/2).

```
2019/03/18 18:36:10 socat[3086] N PTY is /dev/pts/1
2019/03/18 18:36:10 socat[3086] N PTY is /dev/pts/2
2019/03/18 18:36:10 socat[3086] N starting data transfer loop with FDs [5,5] and [7,7]
```

One of them will be assigned by us to CoAP client, another one will be used as the second endpoint of communication where CoAP server would write its requests. Let’s assume that the lower device number is the one used by server and the higher one will be bound to client. We will need that information later. Do not close this or any other terminal that we will open later!

II. Running CoAP server and CoAP client

1) First of all, let’s build a CoAP server application which should be turned on at the beginning of the whole process. In order to do it:

- a) in a separate terminal go to server directory
- b) build a project by calling

```
make
```

if the build failed, create folder called “build” inside of server

- c) edit devices.txt file that contains the list of all devices – leave the name “virtual.uart” and just edit third column – device address. Change it from /dev/pts/3 to the lower address generated by socat in chapter I or leave it like this if socat generated identical address.

d) run server by calling

```
./build/server raw 8001 0.0.0.0 debug
```

“8001” is the port that we will run HTTP server on (in next steps), “0.0.0.0” is just a dummy address that will be used to fill CoAP message. “raw” means that we don’t use ser2net connection – it is not fully implemented, but left in code in case someone would want to implement it correctly. For now we’re focusing on raw mode which means sending data directly to a device and not to a mapped ser2net port.

2) When server is up, it’s time to build and run a client

a) in a separate terminal go to client directory

b) build a project by calling

```
make
```

if the build failed, create folder called “build” inside of client

c) run client by calling

```
./build/client /dev/pts/X debug
```

where X is the upper device number generated by socat in chapter I.

III. Running HTTP server and sending requests to it

Okay, just for the record – at this point we should have 3 terminals running: one with socat, one with CoAP server and one with CoAP client. Now it’s time to add the layer that user will communicate with – a HTTP server.

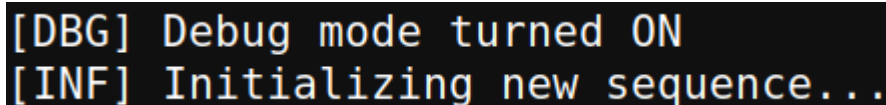
1) In order to run an example HTTP server we have to:

a) in separate terminal go to examples

b) run the server (no compiling since it’s Python) by typing

```
./http_server
```

Look at CoAP server terminal – there should be a log about connecting of HTTP server.



```
[DBG] Debug mode turned ON  
[INF] Initializing new sequence...
```

By default the HTTP server is running on port 8000 and port 8001 is used for communication with CoAP server (that’s why we passed this port to CoAP server in step II.1.d). When request is sent by client, it goes to HTTP server first and then it passes the request to CoAP server – that’s why we need this layer.

2) At this point we have 4 terminals. Now let’s open a fifth one and type:

```
curl -o - 0.0.0.0:8000/virtual.uart/temperature
```

We should get the response “23”. You can look at CoAP client and server terminals – there should be a lot of debug prints.

```
[INF] Received:
161 31 128 0 129 0 66 2 214 13 25 189 60 118 105 114 116 117 97 108 46 117 97 11
4 116 139 116 101 109 112 101 114 97 116 117 114 101 0 0 0
)DBG] Re[4]:25( message: [5]:189(0) [6]:60(<) [7]:118(v) [8]:105(
i) [9]:114(r) [10]:116(t) [11]:117(u) [12]:97(a) [13]:108
(l) [14]:46(.) [15]:117(u) [16]:97(a) [17]:114(r) [18]:116
(t) [19]:139(0) [20]:116(t) [21]:101(e) [22]:109(m) [23]:112
(p) [24]:101(e) [25]:114(r) [26]:97(a) [27]:116(t) [28]:117
(u) [29]:114(r) [30]:101(e)
[INF] Source: 128.0
Destination: 129.0
[INF] Sending temperature: "23"...
[INF] Bytes written: 16
```

```
[INF] Successfully received message
[DBG] Sending:
161 31 128 0 129 0 66 2 214 13 25 189 60 118 105 114 116 117 97 108 46 117 97 11
4 116 139 116 101 109 112 101 114 97 116 117 114 101 0 0 0
[INF] Device /dev/pts/1 opened successfully
[INF] Bytes written: 38
[DBG] Received CoAP response:
[0]:161(0) [1]:9( ) [2]:128(0) [3]:0( ) [4]:129(0) [5]:0( ) [
6]:62(>) [7]:25( ) [8]:69(E) [9]:189(0) [10]:214( ) [11]:255(0) [12]:13( ) [13]:50(2) [14]:51(
3) [15]:0( ) [16]:0( ) [17]:0( ) [18]:0( ) [19]:0( )
[20]:0( ) [21]:0( ) [22]:0( ) [23]:0( ) [24]:0( ) [25]:0( ) [26]:0( ) [27]:0( ) [28]:0( ) [29]:0( ) [30]:0( )
```

As you can see, request needs to have a specific path to the device and its resources. Why the device is called “virtual.uart”? Because that’s the alias defined in devices.txt in CoAP server! You can obviously change it however you want and also set different address/alias instead of 0.0.0.0.

In order to perform POST operation, simply type:

```
curl -o - -d 99 0.0.0.0:8000/virtual.uart/temperature
```

You will get typed value as a response – meaning that value has been set correctly. You can read it again with previous command to find out that it is changed now.