International Institute of Information Technology

Introduction to IoT

Lab 7(OneM2M)

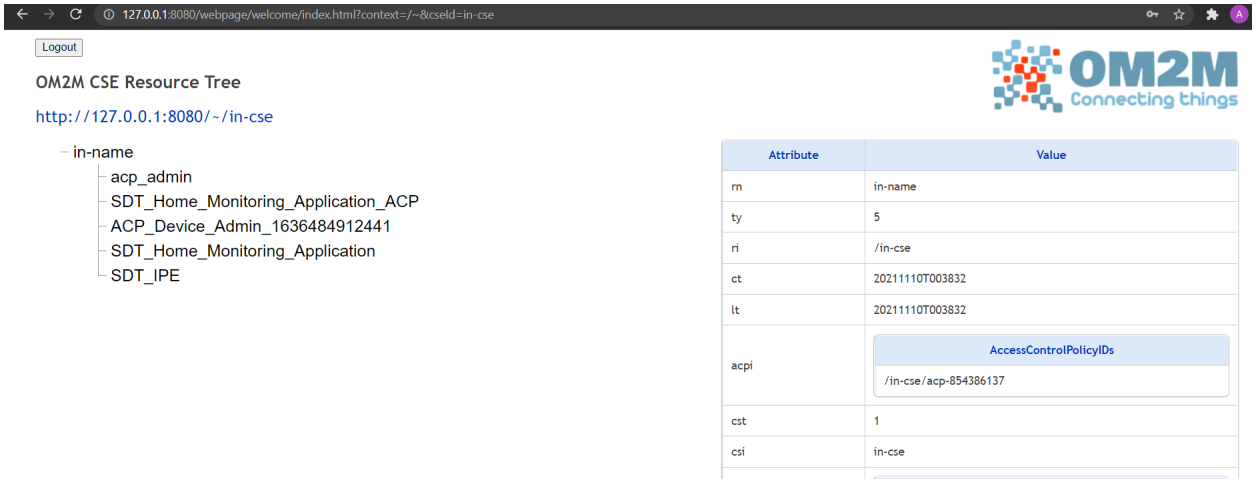
INSTALLATION PROCESS

1. onem2m prerequisites(java)

- as onem2m server is written on java, you need to install java(jdk/jvm/jre) to be able to host your onem2m server.
- go to https://www.java.com/en/download/manual.jsp
- download 64bit version for your respective OS.

2. onem2m platform

- go to https://wiki.eclipse.org/OM2M/Download
- download the latest version of Eclipse Om2m, and unzip the file
- go inside in-cse folder and run start.bat file if on windows, or start.sh file if on linux to start the server.
- Check your java installation if errors arise.
- Go to 127.0.0.1:8080/webpage to ensure your server is hosted.
- type admin as both your username and password and your resource tree will appear



Expected result

TYPES OF RESOURCES in this lab

1. **Application Entity (AE)**: child-resource can be container
2. **Container (CNT)**: child-resource can be contentInstance, or further container
3. **Content Instance (CIN)**: stores the data acutally

HEADERS AND PAYLOADS

- We can create resources in our tree by sending post requests to the parent resource.
- Post request requires header and payload.
- Each type has a different header and payload

AE

```
 headers = {
        'X-M2M-Origin': 'admin:admin',
        'Content-type': 'application/json;ty=2'}

    payload = {
        "m2m:ae": {
            "rn": "{}".format(ae_name),     # RESOURCE NAME
            "api": "Application ID",        # APPLICATION ID
            "rr": "false",}
    }
```

CNT

```
headers = {
        'X-M2M-Origin': 'admin:admin',
        'Content-type': 'application/json;ty=3'}

    payload = {
        "m2m:cnt": {
            "rn": "{}".format(cnt_name),     # RESOURCE NAME
            "mni": 120,}                     # MAX. NO OF INSTANCES
    }
```

CIN

```
headers = {
        'X-M2M-Origin': 'admin:admin',
        'Content-type': 'application/json;ty=4'}

    payload = {
        "m2m:cin": {
            "con": "{}".format(value) }      # CONTENT
     }
```

Experiment Part 1 - Build Resource Tree

provided onem2m.py file have functions that can send POST request to your hosted onem2m server for creating various resources.

```
from onem2m import *
uri_cse = "http://127.0.0.1:8080/~/in-cse/in-name"
ae = "Your_SENSOR_NAME"
cnt = "node1"
```

Import all the functions from onem2m.py and declare the following variable.

a) Create Application Entity
   - Create an AE for your favourite sensor
   - Call function create_ae(uri_cse, ae)

b) Create Container
   - Declare variable as uri_ae = uri_cse + “/” + ae
   - Call function create_cnt(uri_ae, cnt)

c) Create ContentInstance
   - Declare variable as uri_cnt = uri_ae + “/” + cnt
   - Call function create_data_cin(uri_cnt, “random_value”)

Go to 127.0.0.1:8080/webpage to see your Resource Tree. Click of content instance id to view your random_value.

Fig: Supposed output

Experiment Part 2 - create CIN in your resource tree from esp32.

Note:- your laptop and esp32 should be connected to same network/hotspot/ use ifconfig/ipconfig commands to get cse_ip.

a) Connect any sensor to your esp32 board(DHT11, ultrasonic, bmp180, etc), and read its output value.
Now the below pseudo-code describes the data publishing part to OM2M.

Step 1- Including the required header-file and declaring required variables

```
#include "WiFi.h"
#include "HTTPClient.h"
#include "time.h"
#include <ArduinoJson.h>

char* wifi_ssid = "YOUR HOTSPOT/ROUTER NAME";
char* wifi_pwd = "YOUR PASSWORD";
String cse_ip = "192.168.1.2";   // YOUR IP from ipconfig/ifconfig
String cse_port = "8080";
String server = "http://" + cse_ip + ":" + cse_port + "/~/in-cse/in-name/";
String ae = "YOUR_SENSOR_NAME"
String cnt = "Node1"
```

Step- 2 - Create a function createCI, calling which will send the post request to onem2m server.

```
void createCI(String& val){
    // add the lines in step 3-6 inside this function
}
```

Step-3: Constructing the URL based on above parameters inside createCI function

```
HTTPClient http;
http.begin(server + ae + "/" + cnt + "/");
```

Step-4: Attaching the headers before sending the request

```
http.addHeader("X-M2M-Origin", "admin:admin");
http.addHeader("Content-Type", "application/json;ty=4");
```

Step-5: Send the request with the sensor values.

```
      int code = http.POST("{\"m2m:cin\": {\"cnf\":
\"application/json\",\"con\": " + String(val) + "}}");
```

Step-6: Check if the request has been sent and close the connection

```
Serial.println(code);
if (code == -1) {
    Serial.println("UNABLE TO CONNECT TO THE SERVER");
}
http.end();
```

In Setup Function

a. Use Serial.begin() to begin the serial communication
b. Call connect_to_wifi() function, similar to previous labs to connect with wifi.

In Loop Function

a. Read value from your sensor and store it in a String variable val.
b. Call function createCI(val);
c. use appropriate delay

Fig: Supposed output

Experiment Part 3 - Fetch data and create graph

We will be sending get requests to our onem2m resource tree from a python script to get the data Content instances.
For eg.
Lets say uri_cnt = “http:127.0.0.1:8080/ae/cnt”

Then get request on

- uri_cnt + “/la” will give latest content instance under cnt
- uri_cnt + “/?rcn=4” will give for all content instances under cnt
- Uri_cnt + “/ol” will give oldest data content under cnt

Step 1. Define header of your get request

```
headers = {
        'X-M2M-Origin': 'admin:admin',
        'Content-type': 'application/json'
}
```

Step2. Send get request and store the response

```
   response = requests.get(uri, headers=headers)
```

Step3. Parse the response to store content(con) of every content instance in a list variable y
Step 4. Pare the response to store creation time(ct) of every content instance in a list variable x

Step5. Use Matplotlib and x and y list to print a graph. Label your graph accurately.

List of tree parameters for your understanding:

- ae: Application Entity(Sensor/actuators)
- cnt: Container(For holding various kinds of data under the same AE)
- cin: Content Instance(For holding various instances of the same data type)
- sub: Subscription
- rn: Resource Name

- ty: Type
- ri: Resource ID
- pi: Parent Id
- Acpi: Access Control Policies IDs
- uril: URI List
- ct: Creation Time
- et: Expiration Time
- lt: Last Modified Time
- lbl: Label
- cnf: Content Format
- con: Content
- mni: Maximum Number of Instance
- api: Application Id
- poa: Point of Access
- rr: Request Reachability
- sur: Subscription URI

END