INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE

INNOVATIVE SMART SYSTEMS

Middleware Partie 2

oneM2M standard

by JAULHIAC Paul
VASSEUR Cyril

19/12/2024

Year 2024/2025

5A Innovative Smart Systems

1/Objectives

The objective of this lab is to explore and manipulate the service layer of the oneM2M standard and its associated resources using the ACME oneM2M Common Service Entity (CSE). Building on the foundational knowledge acquired through the MOOC and the Eclipse OM2M implementation, this lab focuses on hands-on experience with the ACME stack. By performing tasks such as installation, configuration, resource management, and the development of simulated devices, participants will deepen their understanding of oneM2M's capabilities and interoperability in IoT systems.

2/ First tests with ACME

To begin, the ACME project repository on GitHub provides detailed installation instructions necessary for setting up the IN-CSE. The installation process requires Python version 3.8 or higher, with the latest version being recommended for optimal compatibility and performance.

Initially, the installation and configuration of the ACME oneM2M Common Service Entity (CSE) presented several challenges. On my personal Mac environment, despite multiple attempts spanning approximately 45 minutes, I was unable to achieve a successful setup. Consequently, I transitioned to the dedicated lab machine equipped with `sudo` privileges for the TP sessions.

Even in this environment, I encountered additional hurdles, requiring approximately 15 minutes of troubleshooting and adjustments to resolve configuration issues. Ultimately, I managed to successfully start the ACME CSE. This marked a crucial step in enabling further progress in the lab exercises.

The configuration and running instructions for the IN-CSE can be found within the provided documentation. Using a terminal, the IN-CSE is launched on the laptop. After starting the service, commands are executed to inspect the resource tree, and the web interface is utilized to view and interact with the resources. Once the exploration is complete, the IN-CSE can be stopped.

For reference :

- **Screenshot 1**: Window of ACME CSE on localhost.
- **Screenshot 2**: Resource tree and help display on terminal.

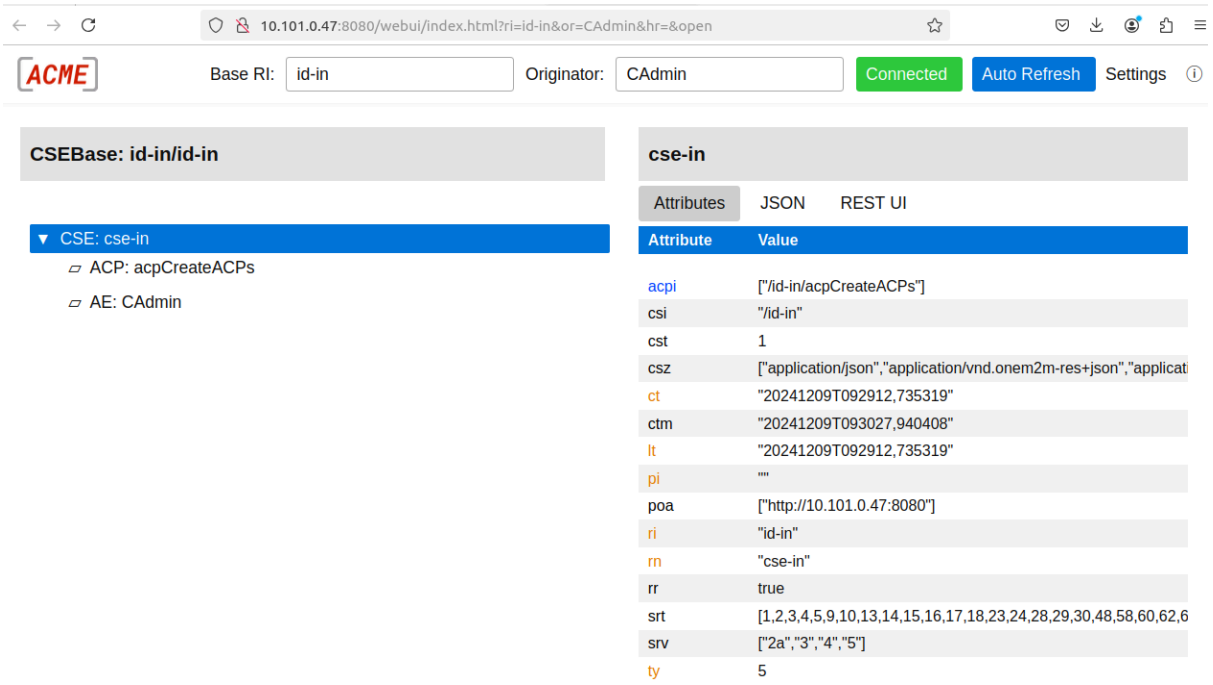


Figure 1 : Window of ACME CSE on localhost

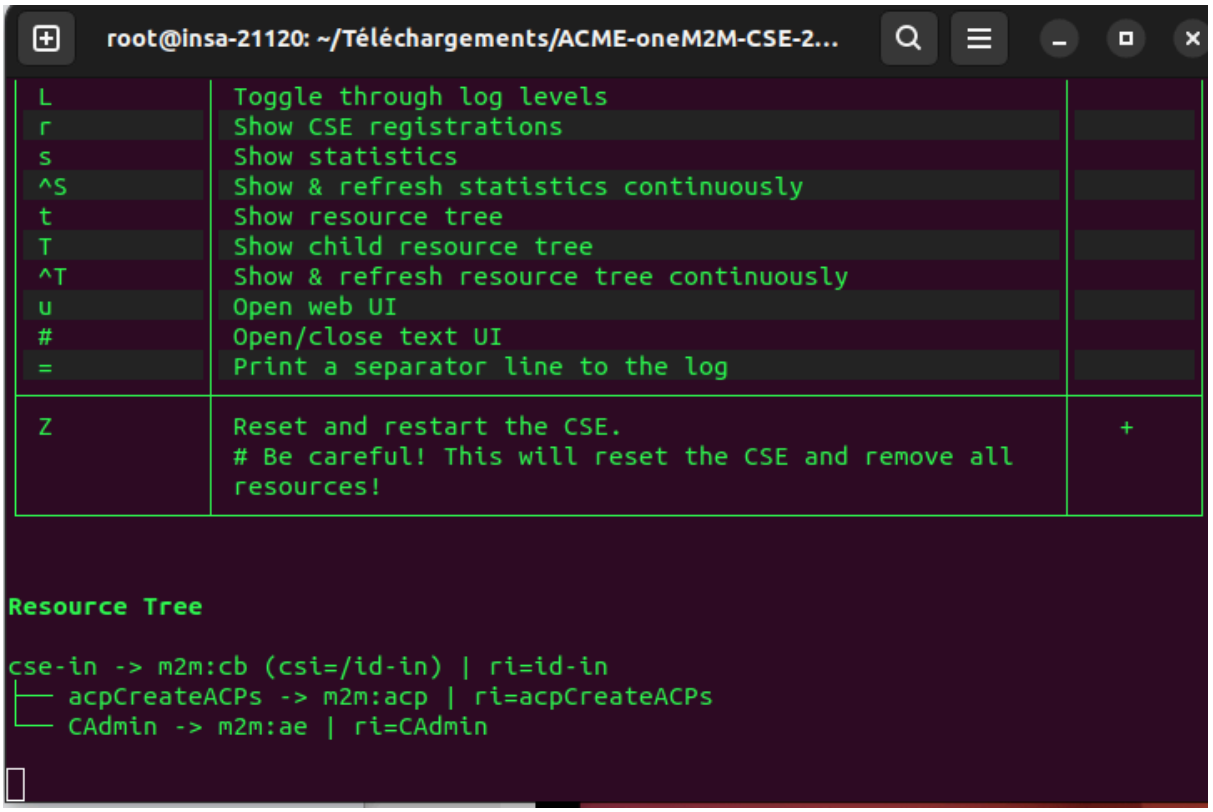


Figure 2 : Resource tree and help display on terminal

3/Manipulation of an acme CSE with a Jupitrer Notebook

During these first steps, we run the code through the Jupyter Notebook to start the CSE and the notification server. As we can see in the two screenshots below, once we get the return messages confirming they are both up and running, we can proceed to test the Web User Interface.

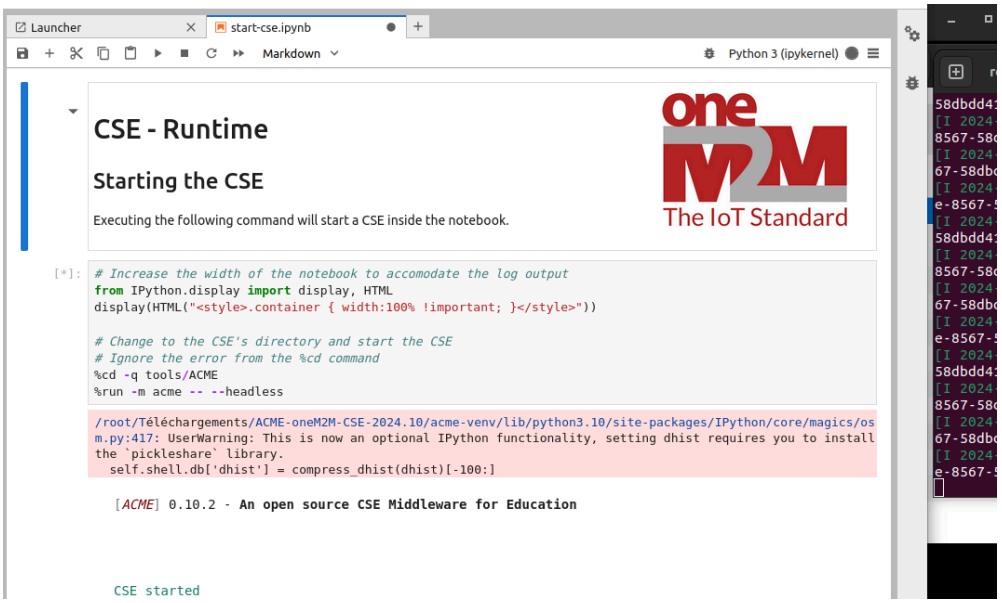


Figure 3 : Screenshot showing the CSE was successfully started

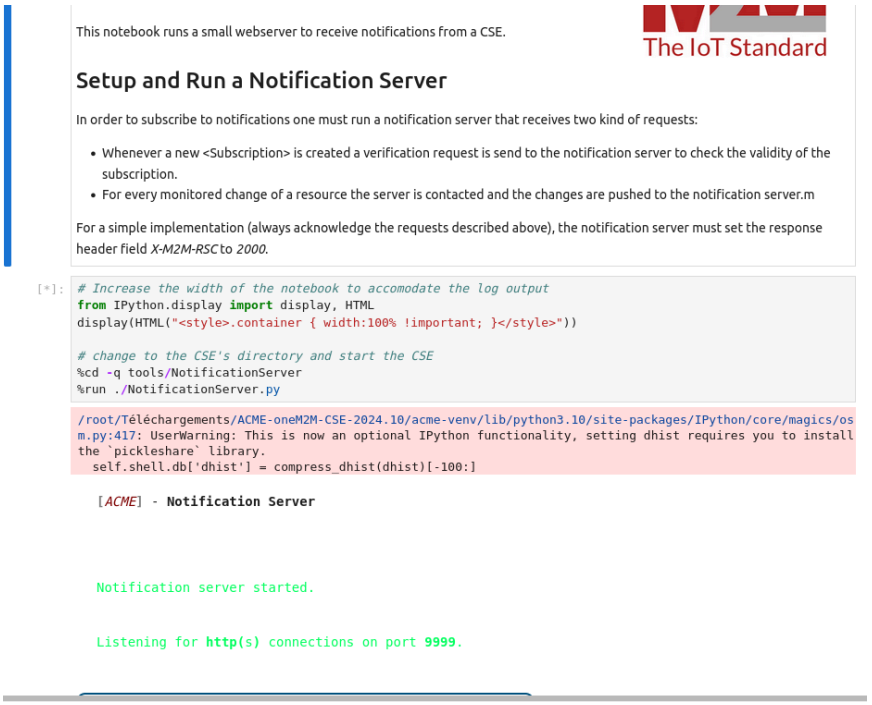


Figure 4 : Screenshot showing the notification server was successfully started

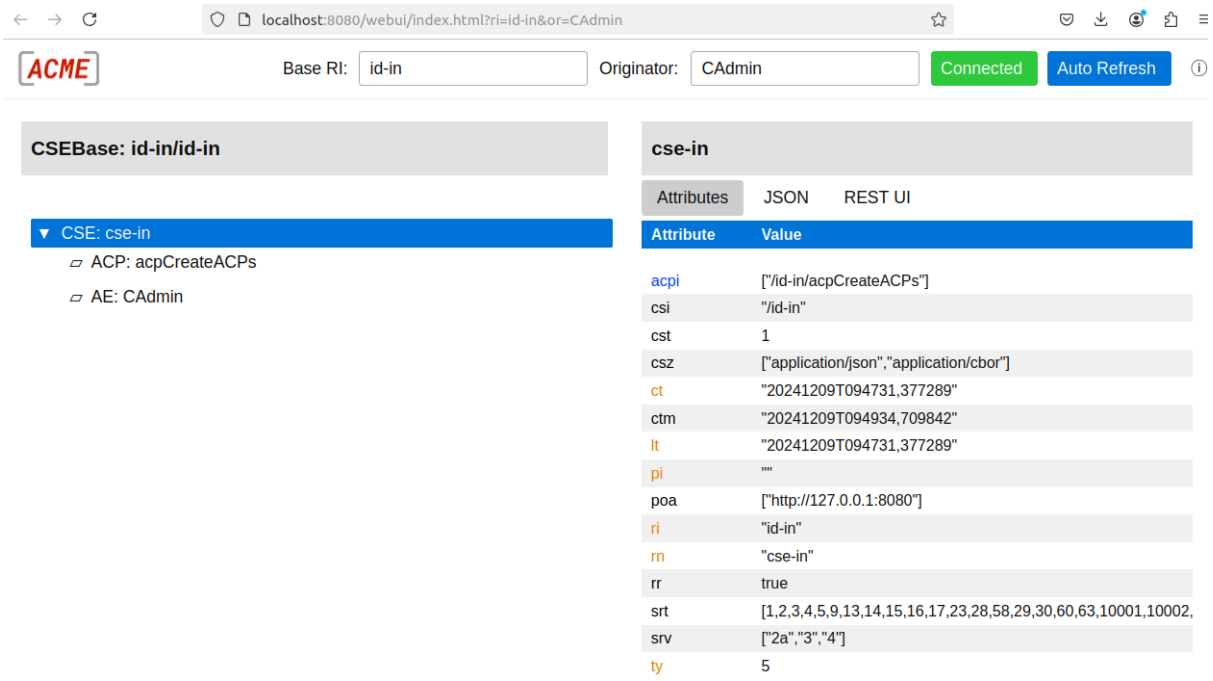
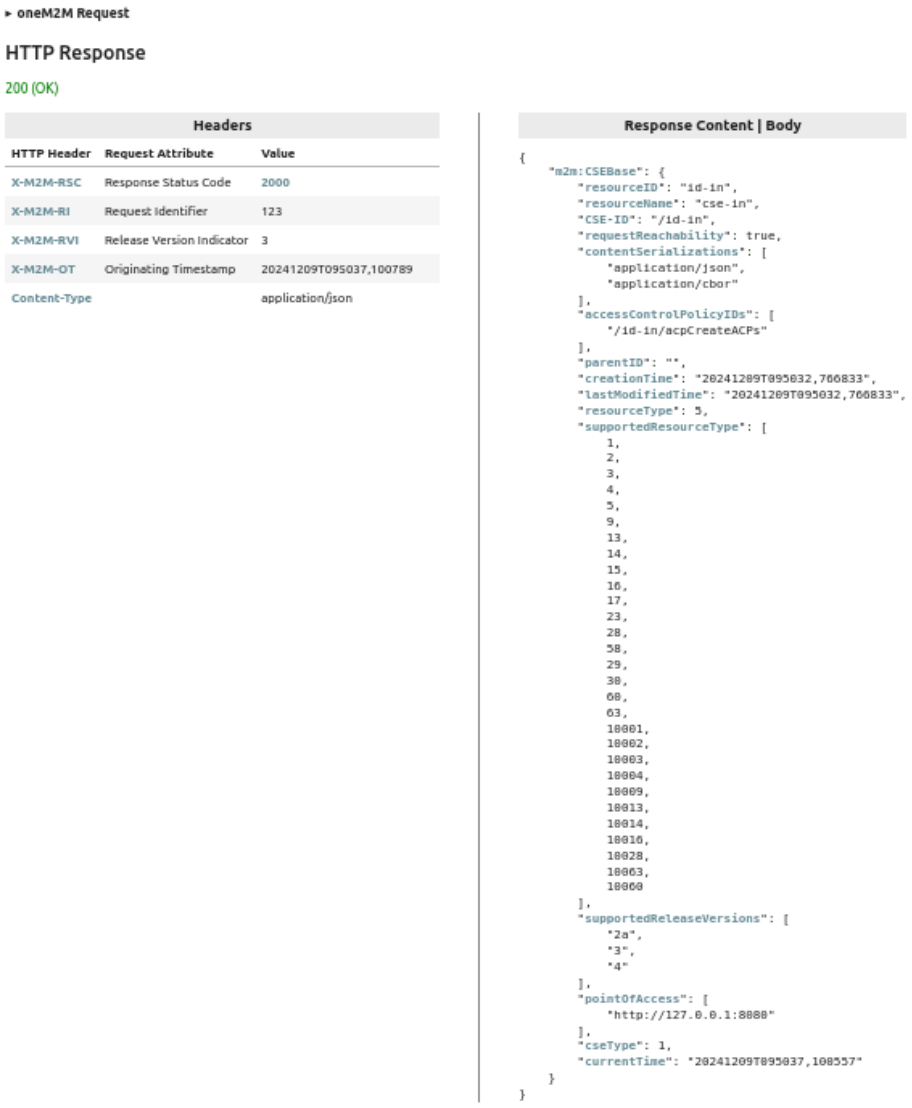


Figure 5 : Functional Web UI

4/Manipulation oneM2M resources

This section describes the practical exploration and manipulation of oneM2M resources using the ACME implementation. Each step builds on the concepts introduced in the accompanying notebook exercises, with Python scripts provided to perform the required operations and screenshots used to highlight specific outcomes.

The first exercise involved initializing the oneM2M environment and retrieving information about the CSEBase resource. This resource serves as the root of the oneM2M resource tree, providing essential metadata about the system. Interacting with the CSEBase confirmed its role in organizing the hierarchical structure of oneM2M resources.



▸ oneM2M Request

HTTP Response

200 (OK)

Headers

HTTP Header	Request Attribute	Value
X-M2M-RSC	Response Status Code	2000
X-M2M-RI	Request Identifier	123
X-M2M-RVI	Release Version Indicator	3
X-M2M-OT	Originating Timestamp	20241209T095037,100789
Content-Type		application/json

Response Content | Body


```
{
    "m2m:CSEBase": {
        "resourceID": "id-in",
        "resourceName": "cse-in",
        "CSE-ID": "/id-in",
        "requestReachability": true,
        "contentSerializations": [
            "application/json",
            "application/cbor"
        ],
        "accessControlPolicyIDs": [
            "/id-in/acpCreateACPs"
        ],
        "parentID": "",
        "creationTime": "20241209T095032,766833",
        "lastModifiedTime": "20241209T095032,766833",
        "resourceType": 5,
        "supportedResourceType": [
            1,
            2,
            3,
            4,
            5,
            9,
            13,
            14,
            15,
            16,
            17,
            23,
            28,
            58,
            29,
            30,
            60,
            63,
            10001,
            10002,
            10003,
            10004,
            10009,
            10013,
            10014,
            10016,
            10028,
            10063,
            10060
        ],
        "supportedReleaseVersions": [
            "2a",
            "3",
            "4"
        ],
        "pointOfAccess": [
            "http://127.0.0.1:8888"
        ],
        "cseType": 1,
        "currentTime": "20241209T095037,108557"
    }
}
```


Figure 6 : CSEBase ressource

An Application Entity (AE) was created to act as a client or device interacting with the oneM2M system. The AE serves as the primary interface for managing and accessing resources within the service layer.

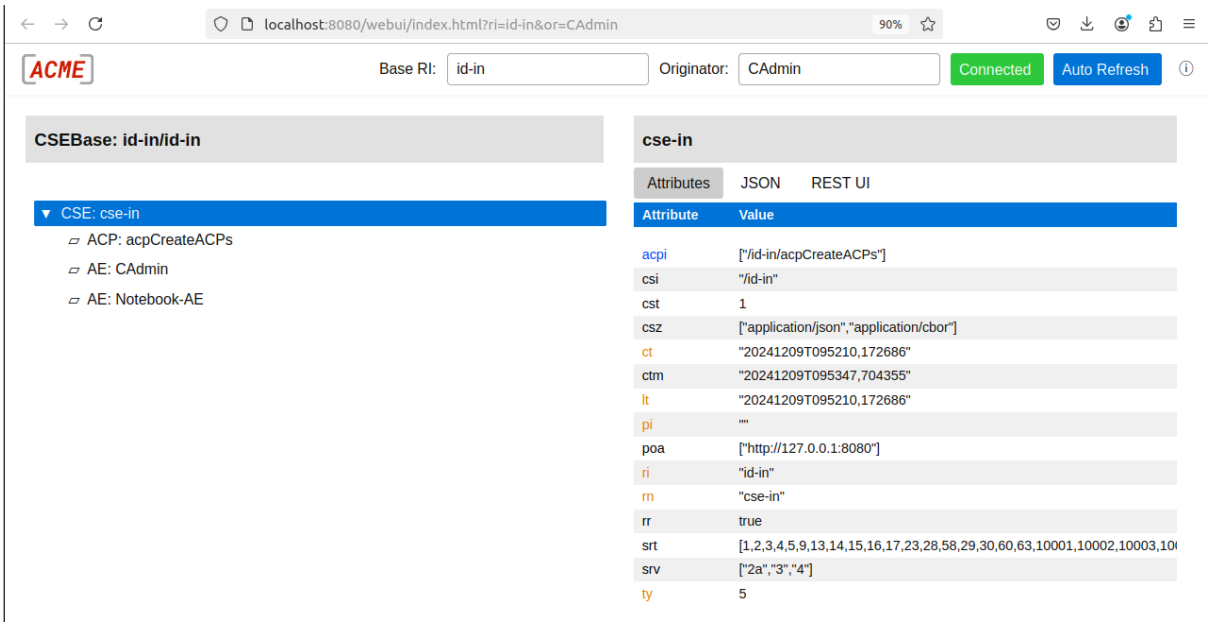


Figure 7 : Application Entity AE

Within the AE, a Container was created to organize resources and manage stored information. Containers are essential for structuring data in oneM2M systems, and their creation is a critical step in resource management.

A Content Instance (CIN) was added to the container to demonstrate how oneM2M stores and retrieves data. The content instance carried specific information ("Hello, World!") to represent dynamic data being uploaded to the IoT system.

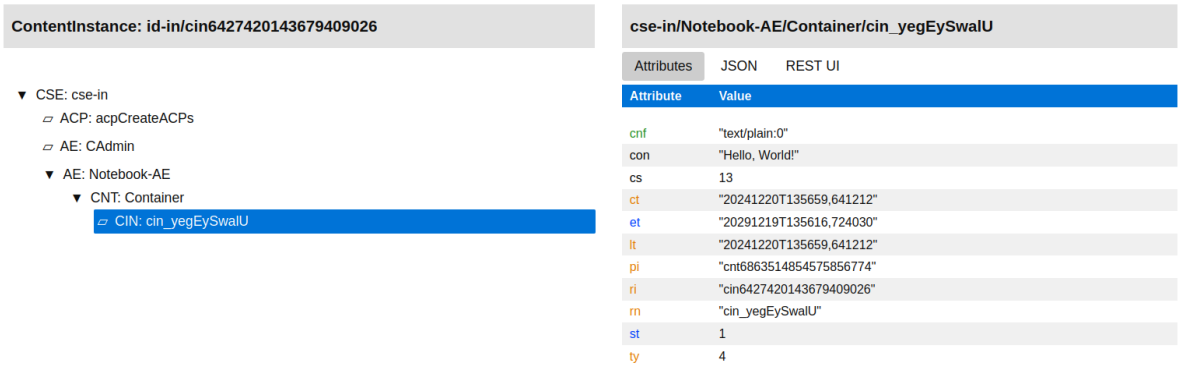


Figure 8 : Container and Content Instance

We write our own program to add a new content instance on the AE: Notebook-AE inside the container:

```
import sys
import requests
import json

# Function to handle the response from the request
def handleResponse(r):
        print(f"Status Code: {r.status_code}")
        print(f"Headers: {r.headers}")
        print(f"Response Text: {r.text}")
        return

# Function to create a new ContentInstance (CIN)
def createCIN():
        # Define the payload for creating the ContentInstance (CIN)
        payload = {
        "m2m:cin": {
        "rn": "Notebook-AE",
        "con": "Joyeux anniversaire PAUL!"
        }
        }

        # Convert the Python dictionary to a JSON string
        payload_json = json.dumps(payload)

        # Define the headers for the request
        _headers = {
        "X-M2M-Origin": "Cmyself",  # Originator
        "X-M2M-RI": "123",  # Request Identifier (unique)
        "X-M2M-RVI": "3",  # Release Version Indicator
        "Content-Type": "application/json;ty=4",  # Content type of the request
        "Accept": "application/json"  # Expected response type
        }

        # Define the resource URL (replace with your actual URL)
        resource_url = "http://localhost:8080/cse-in/Notebook-AE/Container"  # Change as needed

        # Send the POST request to create the ContentInstance
        r = requests.post(resource_url, data=payload_json, headers=_headers)

        # Handle the response from the request
        handleResponse(r)

# Call the createCIN function to create the new ContentInstance
createCIN()
```

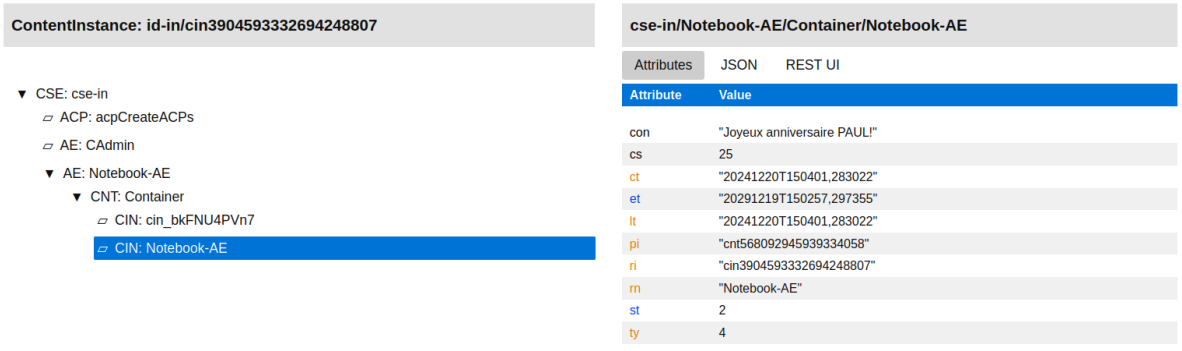


Figure 9 : New Content Instance with python script

Then we define multiple functions in python to create an application entity, a container and to get the last content instance.

```
def createENT() :

        # Define the payload for creating the entity (ENT)
        payload = {
        "m2m:ae": {
        "rn": "Entity-AE",
        "api": "NEntityAPIAE",
        "rr":True,
        "srv":['4']
        }
        }

        # Convert the Python dictionary to a JSON string
        payload_json = json.dumps(payload)

        # Define the headers for the request
        _headers = {
        "X-M2M-Origin": "Cyourself",  # Originator
        "X-M2M-RI": "123",  # Request Identifier (unique)
        "X-M2M-RVI": "3",  # Release Version Indicator
        "Content-Type": "application/json;ty=2",  # Content type of the request
        "Accept": "application/json"  # Expected response type
        }

        # Define the resource URL (replace with your actual URL)
        resource_url = "http://localhost:8080/cse-in"  # Change as needed

        # Send the POST request to create the entity
        r = requests.post(resource_url, data=payload_json, headers=_headers)

        # Handle the response from the request
        handleResponse(r)

def createCNT() :

        # Define the payload for creating the Container (CNT)
        payload = {
        "m2m:cnt": {
        "rn": "container"
        }
        }

        # Convert the Python dictionary to a JSON string
        payload_json = json.dumps(payload)

        # Define the headers for the request
        _headers = {
        "X-M2M-Origin": "Cyourself",  # Originator
        "X-M2M-RI": "123",  # Request Identifier (unique)
        "X-M2M-RVI": "3",  # Release Version Indicator
        "Content-Type": "application/json;ty=3",  # Content type of the request
        "Accept": "application/json"  # Expected response type
        }

        # Define the resource URL (replace with your actual URL)
        resource_url = "http://localhost:8080/cse-in/Entity-AE"  # Change as needed

        # Send the POST request to create the ContentInstance
        r = requests.post(resource_url, data=payload_json, headers=_headers)

        # Handle the response from the request
        handleResponse(r)


# Call the createCIN function to create the new ContentInstance
createCIN()
createENT()
createCNT()
```

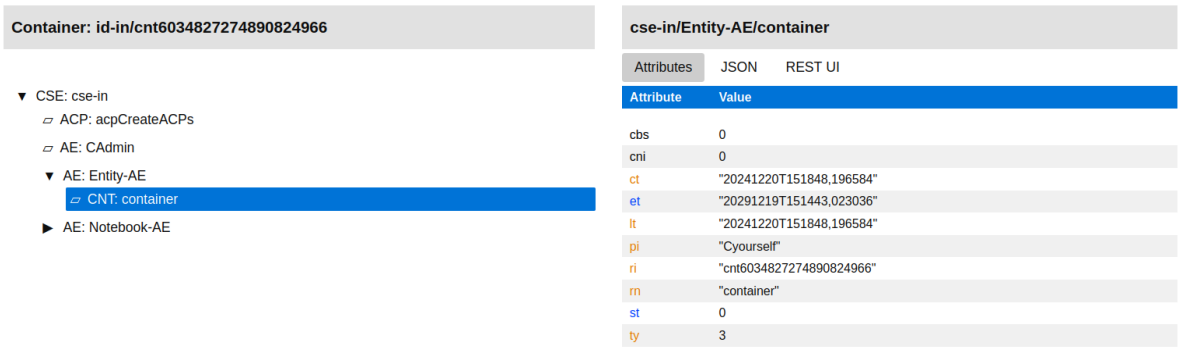


Figure 10 : Container, Instance and Entity creation with python script

5/ Discovery and group

1) Practicing Chapter 3 – Discovery

In this practical work, we explored how the oneM2M framework facilitates resource discovery for managing and retrieving data, specifically <ContentInstance> resources within a <container>. We implemented filtering mechanisms based on criteria like resourceType to retrieve either references or full details of resources.

We also worked with labels to add metadata to resources, enabling categorization and keyword-based searches. During the implementation, we created resources with specific labels and then performed targeted queries based on this. It allowed for conditional and precise searches, demonstrating how resource organization and accessibility can be significantly enhanced in IoT environments.

Throughout the exercise, we utilised key attributes such as filterUsage for conditional queries, filterCriteria for setting filters, and resultContent to control the format of the output. The experience highlighted how oneM2M supports scalable and flexible data management, proving to be a powerful framework for real-time IoT applications.

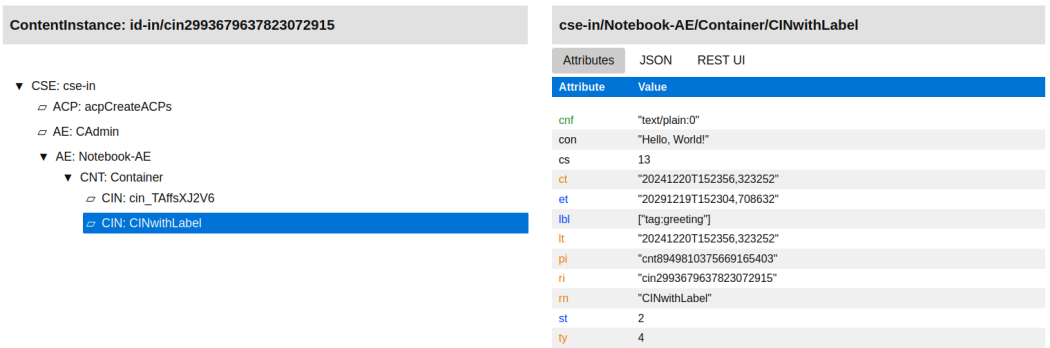


Figure 11 : Content Instances inside the “Container” container

2) Practicing Chapter 4 – Groups

In this practical work, we explored how the oneM2M framework organizes resources into groups using <group> resources, which act as fan-out points to address multiple resources simultaneously (broadcasting). We began by creating a <group> resource containing two <container> resources as members, demonstrating how groups can include resources of different types. The creation process required specifying member IDs and a maximum number of members, while the framework handled internal validation and consistency checks.

Next, we retrieved all <container> resources belonging to the group through the virtual fan-out point. The responses aggregated individual results from each member resource, showcasing the framework's ability to streamline interactions with multiple devices or containers. We then added <contentInstance> resources to each member of the group by sending a single request to the virtual fan-out point. The framework automatically replicated this request for all group members, simplifying the process of simultaneously updating multiple resources with identical data.

And finally, we retrieved the latest <contentInstance> resources from the group's containers by appending /la to the fan-out point path. This demonstrated how the framework supports dynamic queries to fetch the most recent data from all group members. Throughout this exercise, we used attributes such as memberIDs for specifying group members, aggregatedResponse for combining results, and consistencyStrategy for handling validation errors. These features highlighted the scalability and efficiency of oneM2M in managing grouped resources, making it ideal for IoT applications requiring synchronized updates and batch data retrieval.

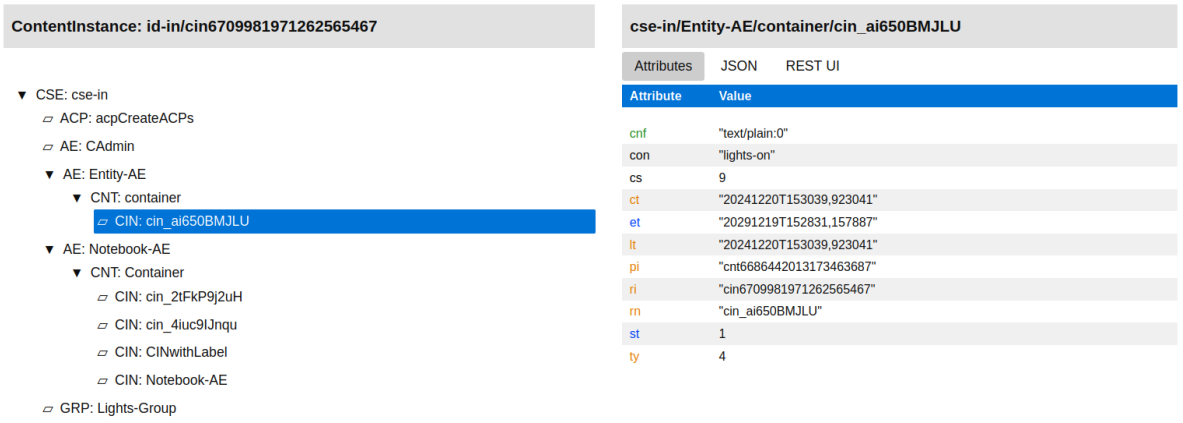


Figure 12 : Visualisation of the “Lights-Group” group on the Web UI

6/ More advanced resources

This section explores how access control is implemented and enforced within the oneM2M framework, focusing on the definition and management of Access Control Policies (ACPs) to regulate resource access. A streetlight scenario is used to demonstrate the practical application of these concepts.

The objective of the experiment was to investigate the default access control behavior in oneM2M and extend it by defining and assigning ACPs to resources. The scenarios tested how access could be granted or restricted for different originators based on the defined policies.

By default, access to resources in oneM2M is limited to the resource creator unless an Access Control Policy (ACP) is applied. ACPs offer fine-grained control by specifying privileges for different originators, such as CREATE, RETRIEVE, UPDATE, or DELETE, using a bitfield representation. Additionally, ACPs can be dynamically created and assigned, providing flexibility for secure collaboration among multiple originators in IoT systems.

The experiment demonstrated the robustness of oneM2M's access control mechanism. Through ACPs, oneM2M ensures that resources are accessible only to authorized originators, enhancing security while supporting scalable multi-user collaboration. This flexibility positions oneM2M as a powerful choice for IoT applications that require controlled resource sharing.

Figure 13 : Example of no authorization access

Due to time constraints, we were unable to cover the section "notification". As a result, it was not included in this experiment. However, this part would have allowed for a deeper exploration of the notification mechanisms within an IoT system implementation using oneM2M.

7/ Simulated device

We were unable to complete this task. However, with more time, we believe we would have been able to develop the application to simulate the behavior of the device created during Labs 1 and 2 on the ESP8266. We could have leveraged the Python scripts already written in previous sections to make REST requests and simulate the device behavior effectively.

8/ Conclusion

In conclusion, this lab provided a comprehensive hands-on experience with the oneM2M standard, allowing us to explore its core functionalities, including resource management, access control, and group handling. Despite some initial challenges with installation and configuration, we successfully navigated through the setup and tested various features, such as creating and manipulating Application Entities, Containers, and Content Instances. The exploration of resource discovery and grouping demonstrated the flexibility and scalability of oneM2M in managing IoT data.

Furthermore, the lab allowed us to delve into security mechanisms, particularly Access Control Policies (ACPs), which offer fine-grained access management for IoT resources. This is crucial for ensuring that only authorized entities can access specific data, enhancing the overall security of IoT systems.

Although due to time constraints, certain aspects like the notification system and device simulation were not completed, we are confident that with more time, we would have been able to integrate these features using the Python scripts developed earlier in the lab. Overall, the lab has greatly enhanced our understanding of oneM2M, equipping us with valuable skills to work with IoT systems in real-world applications.