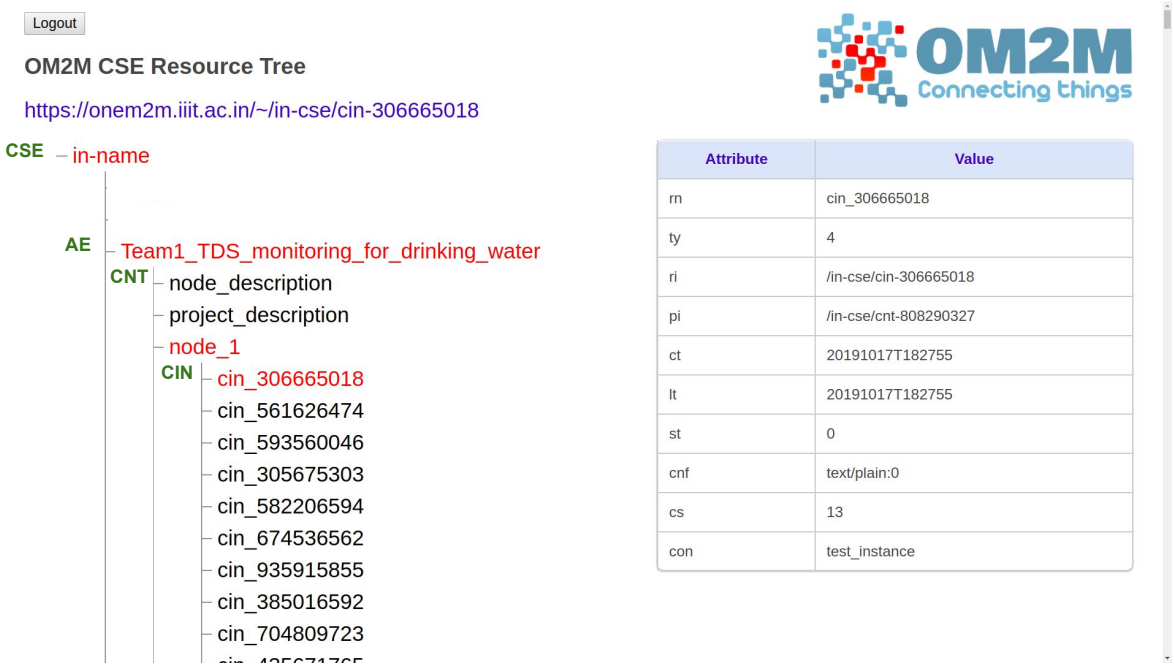
OneM2M Demo

Suraj, IIIT-H

OneM2M Resource tree

CSE : Common service entity
AE : Application entity
CNT : Container
CIN : Container Instance



- 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

Protocols to communicate with OneM2M

- **HTTP**
- MQTT
- CoAP

HTTP Messages

The hardware reads the sensor data and encapsulates a HTTP message by forming the *header* and the *payload.*

- Header:
  - Consists of meta information, credentials, etc…
- Payload:
  - Actual data and other relevant parameters
  - Can specify payload in either XML or **JSON** format
- This is then sent to the IoT server using appropriate HTTP methods.

HTTP Request can be made from any device

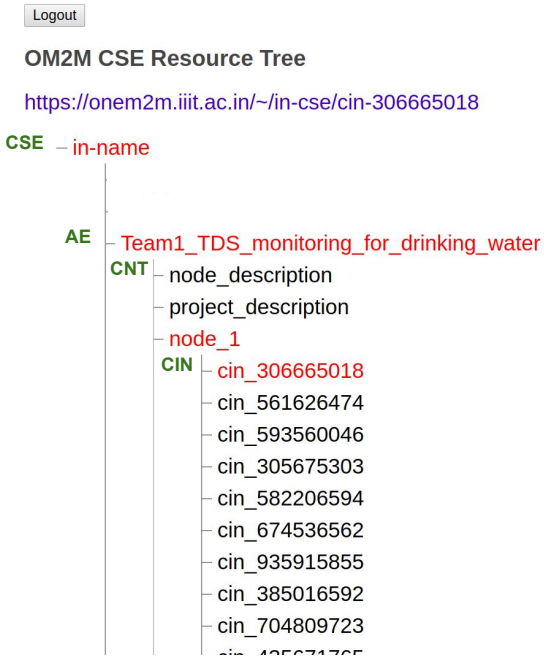
Condition : Hardware should have a TCP/IP stack

- A plain arduino UNO or Mega doesn’t have TCP/IP stack
  (should use a WiFi shield)
- RaspberryPi, BeagleBone, etc.. have the TCP/IP stack.
- Considering the power consumption and footprint, ESP has been a good choice
  (ESP32, ESP8266, NodeMCU)

For the purpose of this demo...

Build the header and payload using Python libraries on my local system.

- Registering AE
- Creating Container
- Create content instances
- Retrieve data



OM2M Connecting things

Attribute	Value
rn	cin_306665018
ty	4
ri	/in-cse/cin-306665018
pi	/in-cse/cnt-808290327
ct	20191017T182755
lt	20191017T182755
st	0
cnf	text/plain:0
cs	13
con	test_instance

For the purpose of this demo...

- **Registering AE**
- Create Container
- Create content instances
- Retrieve data

Registering AE

Things to note:

- Run once per AE
- Type:2 (specifies we are creating an AE)
- rn : resource name (name of the AE)
- lbl : labels (used to filter)
- Header + payload = HTTP message
- Then execute a HTTP POST method (sends the HTTP message to the server)

```
def register_ae(uri_cse, ae_name, labels="", fmt_ex="json"):
    """
        Method description:
        Registers an application entity(AE) to the OneM2M framework/tree
        under the specified CSE

        Parameters:
        uri_cse : [str] URI of parent CSE
        ae_name : [str] name of the AE
        labels : [str] labels for the AE
        fmt_ex : [str] payload format
    """

    headers = {
        'X-M2M-Origin': 'admin:admin',
        'Content-type': 'application/{};ty=2'.format(fmt_ex)}

    payload = {
        "m2m:ae": {
            "rn": "{}".format(ae_name),
            "api": "tap",
            "rr": "true",
            "lbl": labels
        }
    }

    response = requests.post(uri_cse, json=payload, headers=headers)
    print('Return code : {}'.format(response.status_code))
    print('Return Content : {}'.format(response.text))
```

For the purpose of this demo...

- Registering AE
- **Creating Container**
- Create content instances
- Retrieve data

Creating Container

Things to note:

- Run once per CNT
- Type:3 (specifies we are creating an CNT)
- rn : resource name (name of the CNT)
- mni : Maximum number of Instances
- Header + payload = HTTP message
- Then execute a HTTP POST method (sends the HTTP message to the server)

```
def create_cnt(uri_ae, cnt_name="", fmt_ex="json"):
    """
        Method description:
        Creates a container(CON) in the OneM2M framework/tree
        under the specified AE

        Parameters:
        uri_ae : [str] URI for the parent AE
        cnt_name : [str] name of the container (DESCRIPTOR/DATA)
        fmt_ex : [str] payload format
    """

    headers = {
        'X-M2M-Origin': 'admin:admin',
        'Content-type': 'application/{};ty=3'.format(fmt_ex)}

    payload = {
        "m2m:cnt": {
            "rn": "{}".format(cnt_name),
            "mni": -1
        }
    }

    response = requests.post(uri_ae, json=payload, headers=headers)
    print('Return code : {}'.format(response.status_code))
    print('Return Content : {}'.format(response.text))
```

For the purpose of this demo...

- Registering AE
- Creating Container
- **Create content instances**
- Retrieve data

Creating Content instances

Things to note:

- Type:4 (specifies we are creating an CIN)
- con : content (sensor/actuator state)
- Header + payload = HTTP message
- Then execute a HTTP POST method (sends the HTTP message to the server)

```
def create_data_cin(uri_cnt, value, fmt_ex="json"):
    """
        Method description:
        Creates a data content instance(data_CIN) in the OneM2M framework/tree
        under the specified DATA CON

        Parameters:
        uri_cnt : [str] URI for the parent DATA CON
        fmt_ex : [str] payload format (json/XML)
    """

    headers = {
        'X-M2M-Origin': 'admin:admin',
        'Content-type': 'application/{};ty=4'.format(fmt_ex)}

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

    response = requests.post(uri_cnt, json=payload, headers=headers)
    print('Return code : {}'.format(response.status_code))
    print('Return Content : {}'.format(response.text))
```

For the purpose of this demo...

- Registering AE
- Creating Container
- Create content instances
- **Retrieve data**

Retrieve

- Using the GET method by specifying the URI of the resource

Things to note:

- Type: not necessary to specify
- No payload
- Just Header = HTTP message
- Then execute a HTTP GET method (get the data that we want)
- Append “/la” to the URI to get the latest content instance from the specified container

```
def get_data(uri, format="json"):
    """
        Method description:
        Gets data from the specified container(data_CIN)
        in the OneM2M framework/tree

        Parameters:
        uri : [str] URI for the parent DATA CON appended by "la" or "ol"
        fmt_ex : [str] payload format (json/XML)
    """
    headers = {
        'X-M2M-Origin': 'admin:admin',
        'Content-type': 'application/json'}

    response = requests.get(uri, headers=headers)
    print('Return code : {}'.format(response.status_code))
    print('Return Content : {}'.format(response.text))
    _resp = json.loads(response.text)
    return response.status_code, _resp["m2m:cin"]["con"]
```

Kinds of URI

- Direct URI
- Indirect URI
- Other kinds can be found in the OneM2M documentation

Advanced Features

- Grouping
- **Filtering**
- Subscription
- Security and permissions

- Can pass various parameters for attaining flavours of functionality
  - Eg: Get data after a particular date, get data between 2 time intervals, etc..

To summarize

- IoT devices need to make HTTP requests to communicate with the OneM2M resource tree:
  - To send data, we use the POST method using HTTP request
  - To get data, we use the GET method
  - There are other methods like UPDATE and DELETE

- HTTP Requests can be made from any kind of device given it has TCP/IP hardware stack.
  - Arduino boards with wifi shields
  - ESP32, ESP8266
  - Raspberry Pi

Advice for the upcoming hackathon

- Documentation available on onem2m.org
- Try out an example setup from hackster.io
- Python libraries available on github. Will share mine too.
- JAVA 8 is required to run the OneM2M server if downloaded from eclipse.org

References

- http://www.onem2m.org/getting-started/onem2m-overview/introduction/service-layer
- http://www.onem2m.org/technical/published-drafts/release-3
- http://www.onem2m.org/developer-guides