NGSI V2 API

FIWARE Mexico

Co-funded by the Horizon 2020 Framework
Programme of the European Union

Orion Context Broker

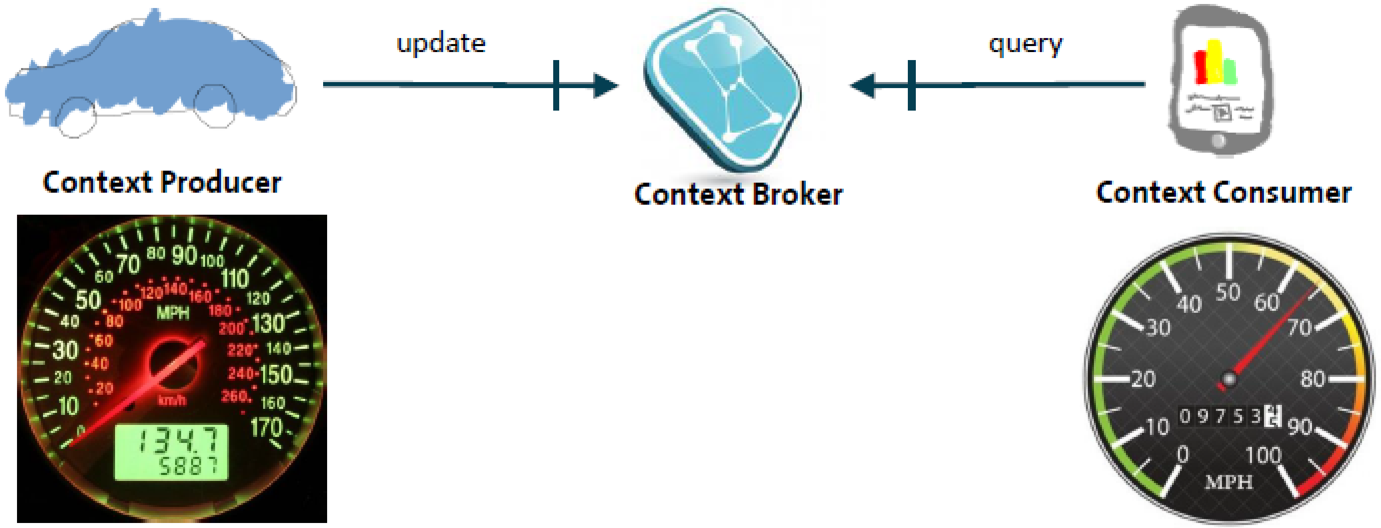
FIWARE
Mexico

Co-funded by the Horizon 2020 Framework Programme of
the European Union

Context Broker Operations

- **Context Producers** publish data/context elements by invoking the **update** operations on a Context Broker.

- **Context Consumers** can retrieve data/context elements by invoking the **query** operations on a Context Broker



update

query

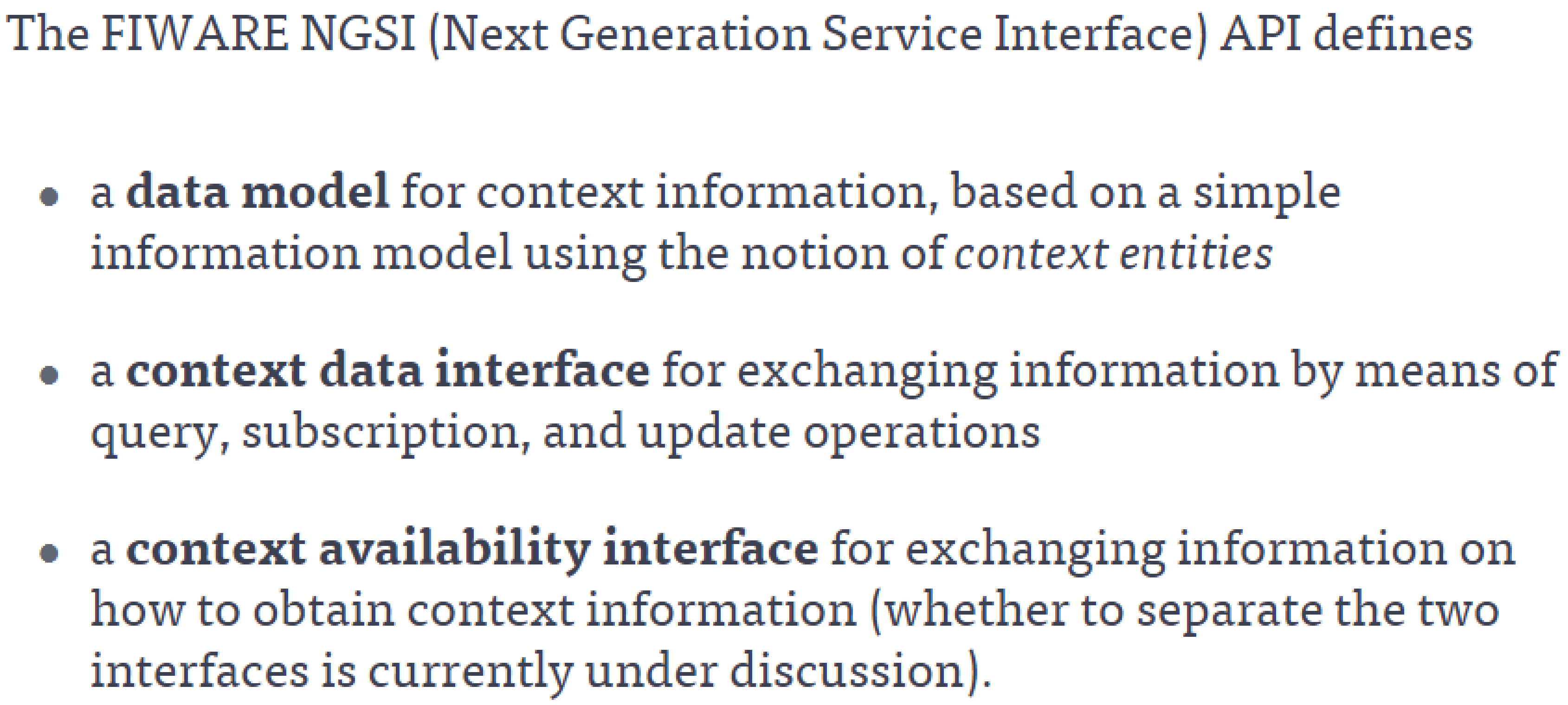
Context Producer

Context Broker

Context Consumer

Co-funded by the Horizon 2020 Framework Programme of
the European Union

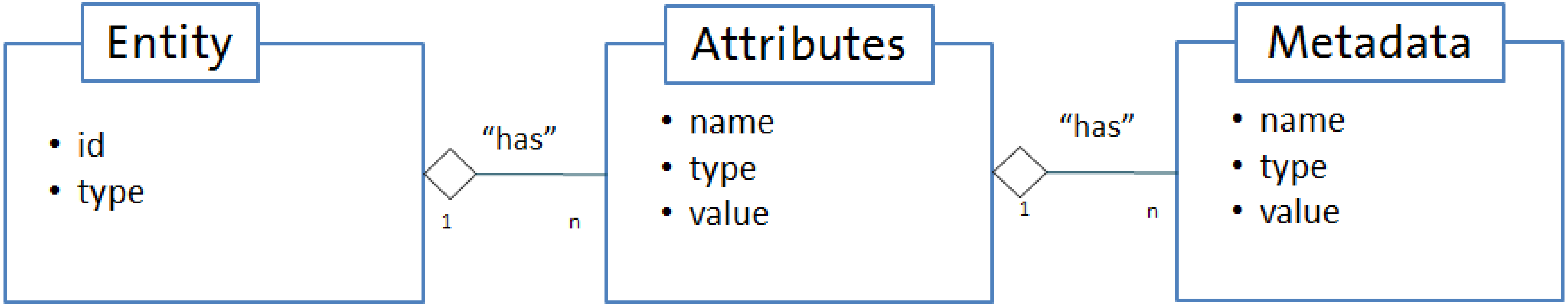
FIWARE NGSI



The FIWARE NGSI (Next Generation Service Interface) API defines

- a **data model** for context information, based on a simple information model using the notion of *context entities*
- a **context data interface** for exchanging information by means of query, subscription, and update operations
- a **context availability interface** for exchanging information on how to obtain context information (whether to separate the two interfaces is currently under discussion).

Context Data



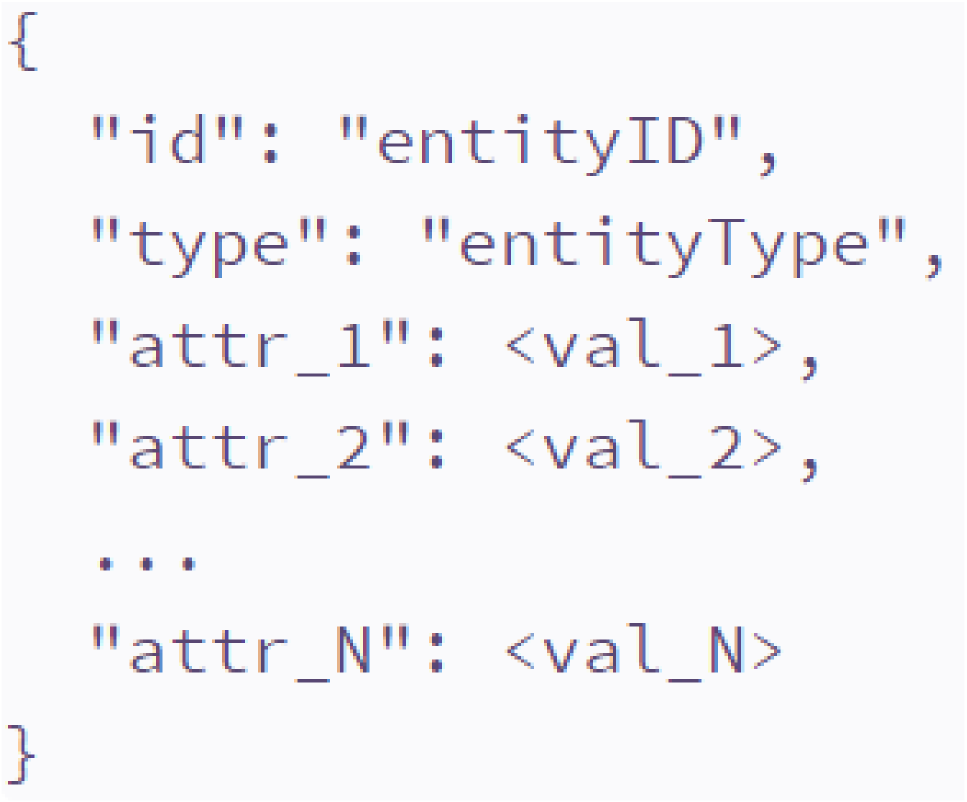
MIME Types

- The API response payloads in this specification are based on **application/json** and (for attribute value type operation) **text/plain** MIME types. Clients issuing HTTP requests with accept types different than those will get a **406 Not Acceptable** error.

Entity Representation

- The entity id is specified by the object's **id** property, whose value is a string containing the entity id.
- The entity type is specified by the object's **type** property, whose value is a string containing the entity's type name.
- Entity attributes are specified by additional properties, whose names are the **name** of the attribute and whose representation is described in the "JSON Attribute Representation" section below. Obviously, **id** and **type** are not allowed to be used as attribute names.

Entity Representation

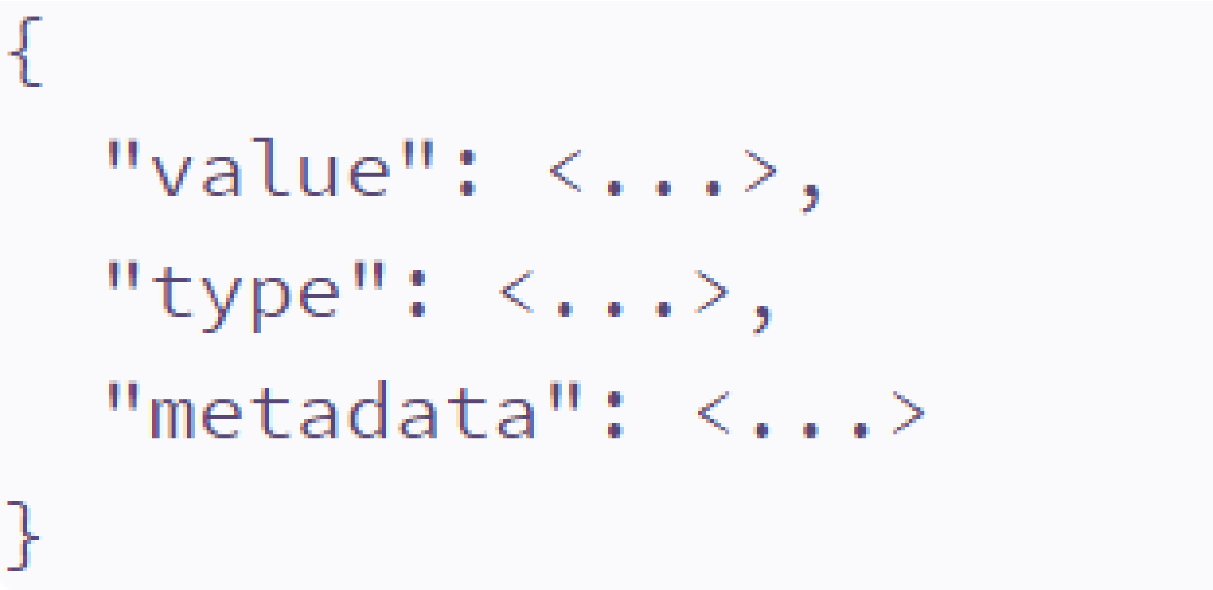


```
{
  "id": "entityID",
  "type": "entityType",
  "attr_1": <val_1>,
  "attr_2": <val_2>,
  ...
  "attr_N": <val_N>
}
```

JSON Attribute Representation

- The attribute **value** is specified by the value property, whose value may be any JSON datatype.
- The attribute NGSI **type** is specified by the type property, whose value is a string containing the NGSI type.
- The attribute metadata is specified by the **metadata** property. Its value is another JSON object which contains a property per metadata element defined (the name of the property is the name of the metadata element)

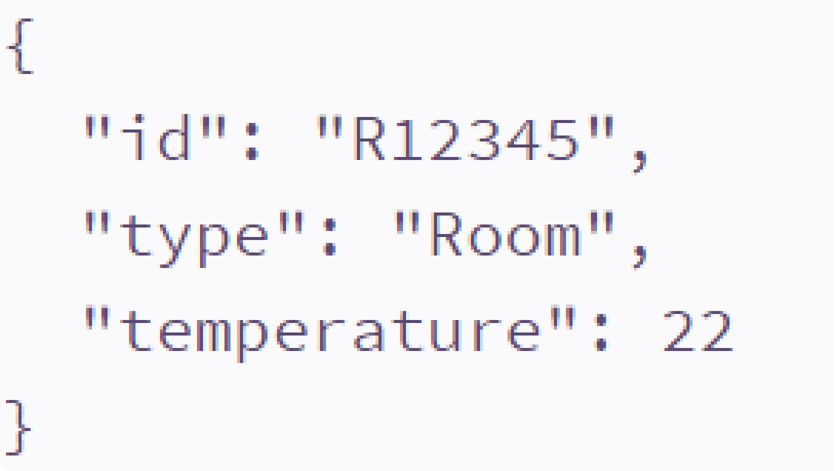
JSON Attribute Representation



```
{
  "value": <...>,
  "type": <...>,
  "metadata": <...>
}
```

Simplified Entity Representation

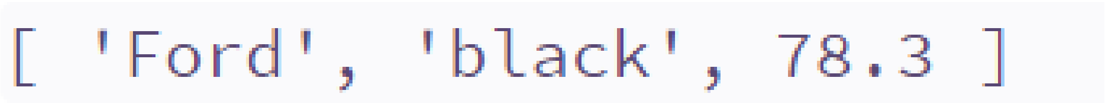
- **keyValues** mode. This mode represents the entity attributes by their values only, leaving out the information about type and metadata. See example below.



```
{
  "id": "R12345",
  "type": "Room",
  "temperature": 22
}
```

Simplified Entity Representation

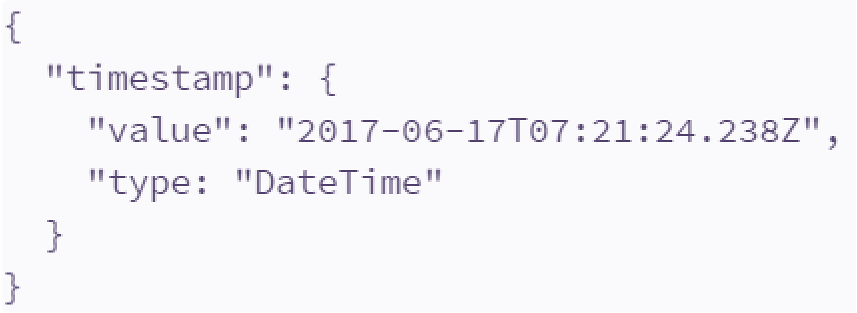
- **values mode**. This mode represents the entity as an array of attribute values.



```
[ 'Ford', 'black', 78.3 ]
```

Special Attribute Types

- **DateTime**: identifies dates, in ISO8601 format. These attributes can be used with the query operators greater-than, less-than, greater-or-equal, less-or-equal and range. For instance (only the referred entity attribute is shown):



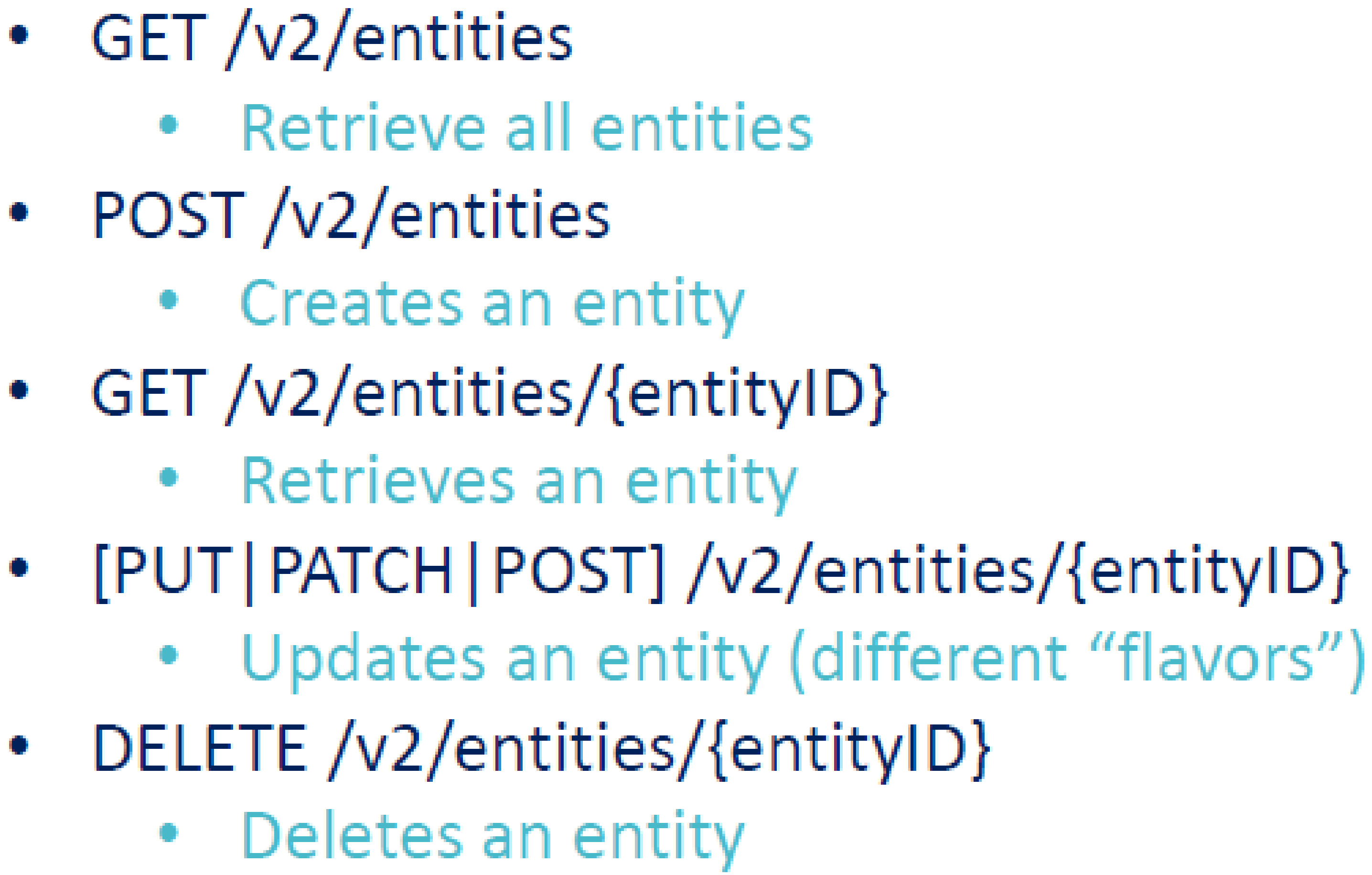
```
{
  "timestamp": {
    "value": "2017-06-17T07:21:24.238Z",
    "type: "DateTime"
  }
}
```

Virtual Attributes

- inside of **options:**
  - **dateCreated** (type:DateTime) ISO 8601.
  - **dateModified** (type:DateTime) ISO 8601.

- Like regular attributes, the can be used in **attrs, q** filters and **order by.**

Common Operations



- GET /v2/entities
  - Retrieve all entities
- POST /v2/entities
  - Creates an entity
- GET /v2/entities/{entityID}
  - Retrieves an entity
- [PUT|PATCH|POST] /v2/entities/{entityID}
  - Updates an entity (different “flavors”)
- DELETE /v2/entities/{entityID}
  - Deletes an entity

Common Operations

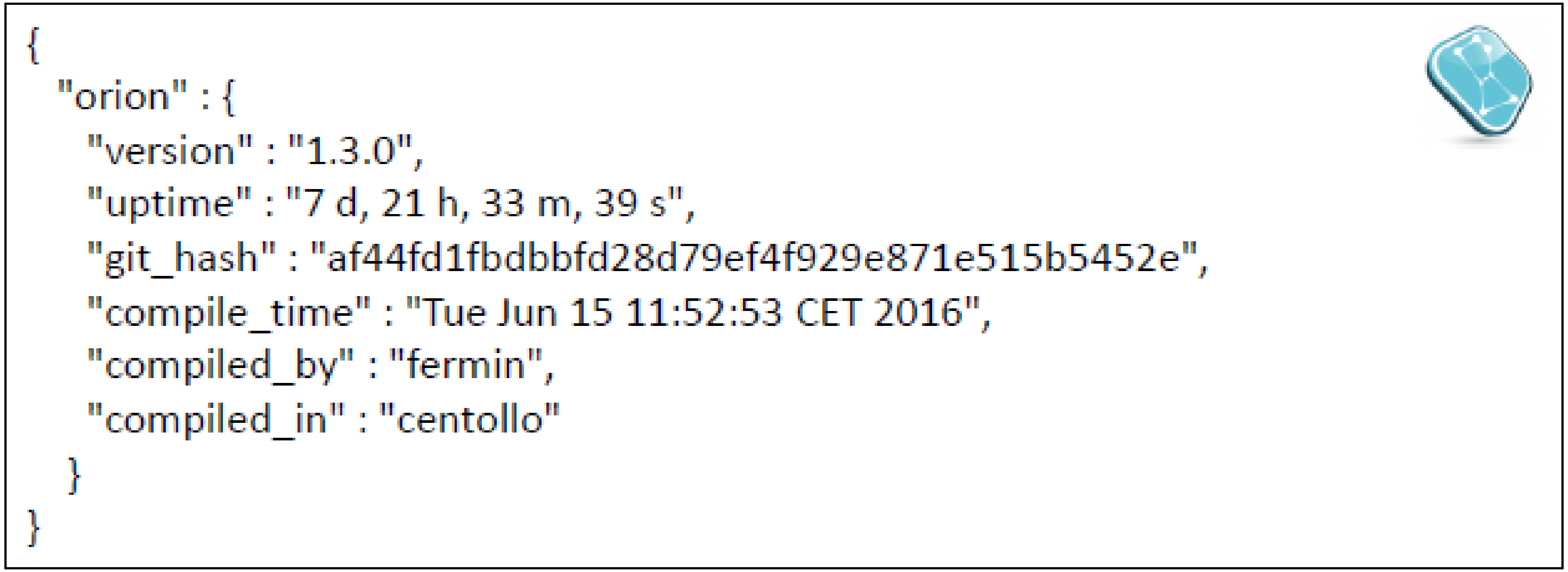
- GET /v2/entities/{entityID}/attrs/{attrName}
  - Retrieves an attribute’s data
- PUT /v2/entities/{entityID}/attrs/{attrName}
  - Updates an attribute’s data
- DELETE /v2/entities/{entityID}/attrs/{attrName}
  - Deletes an attribute
- GET /v2/entities/{entityID}/attrs/{attrName}/value
  - Retrieves an attribute’s value
- PUT /v2/entities/{entityID}/attrs/{attrName}/value
  - Updates an attribute’s value

Follow the steps at

https://codeshare.io/5X8egM

Check Health

```
GET <cb_host>:1026/version
```



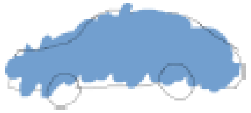
```
{
  "orion" : {
    "version" : "1.3.0",
    "uptime" : "7 d, 21 h, 33 m, 39 s",
    "git_hash" : "af44fd1fbdbbfd28d79ef4f929e871e515b5452e",
    "compile_time" : "Tue Jun 15 11:52:53 CET 2016",
    "compiled_by" : "fermin",
    "compiled_in" : "centollo"
  }
}
```

Create Entities

FIWARE Mexico

Add Entity

```
POST <cb_host>:1026/v2/entities
Content-Type: application/json
...

{
  "id": "Car1",
  "type": "Car",
  "speed": {
    "type": "Float",
    "value": 98
  }
}
```



```
201 Created
```

Get Entity

```
GET <cb_host>:1026/v2/entities/Car1/attrs/speed
```

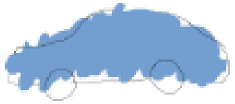
```
200 OK
Content-Type: application/json
...

{
   "type": "Float",
   "value": 110,
   "metadata": {}
}
```

You can get all the attributes of the entity using the entity URL:
   GET/v2/entities/Car1/attrs

Update Entities

FIWARE
Mexico

Update Entity



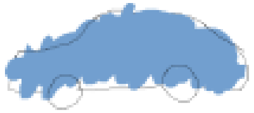
```
PUT <cb_host>:1026/v2/entities/Car1/attrs/speed
Content-Type: application/json
...

{
   "type": "Float",
   "value": 110
}
```

In the case of id ambiguity, you can use "?type=Car" to specify entity type

```
204 No Content
...
```

Update Entity

```
PUT <cb_host>:1026/v2/entities/Car1/attrs/speed/value
Content-Type: text/plain
...

115
```



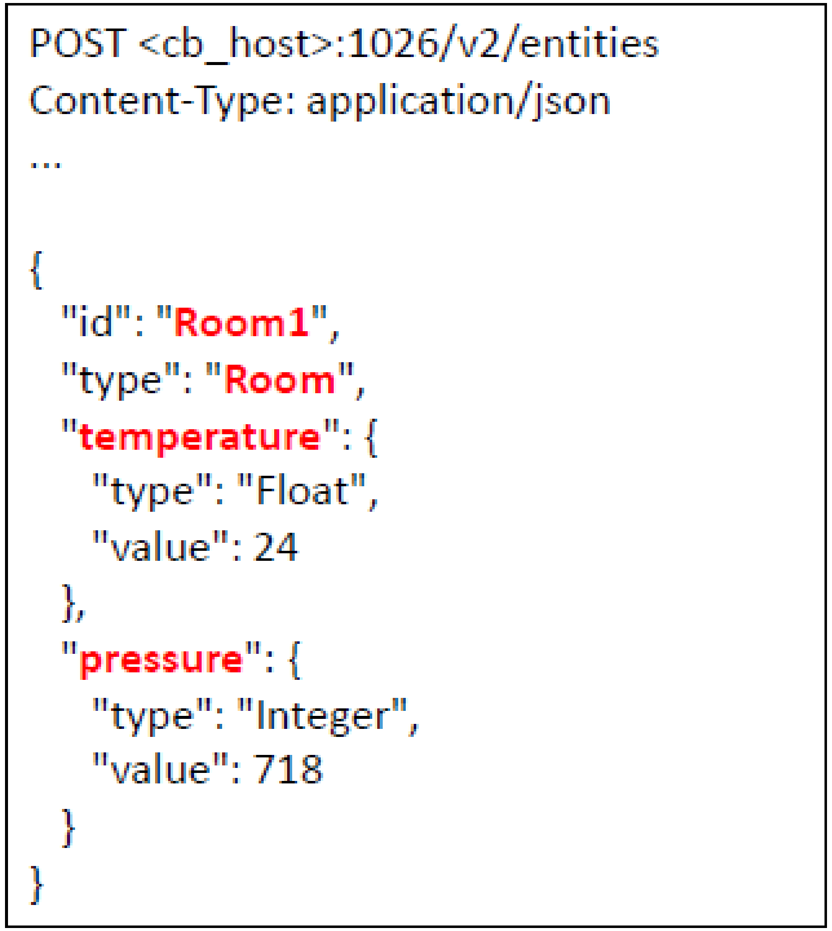
```
204 No Content
...
```

Update Entity – text/plain

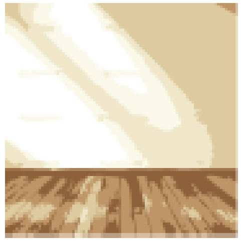
```
GET <cb_host>:1026/v2/entities/Car1/attrs/speed/value
Accept: text/plain
```

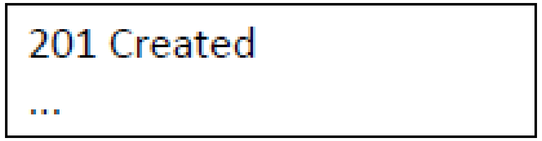
```
200 OK
Content-Type: text/plain
...

115.000000
```

Create Room



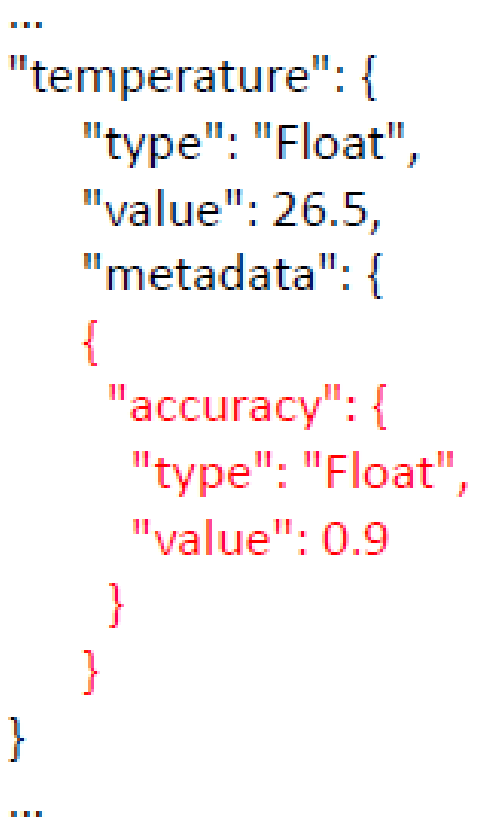
```
POST <cb_host>:1026/v2/entities
Content-Type: application/json
...

{
  "id": "Room1",
  "type": "Room",
  "temperature": {
    "type": "Float",
    "value": 24
  },
  "pressure": {
    "type": "Integer",
    "value": 718
  }
}
```



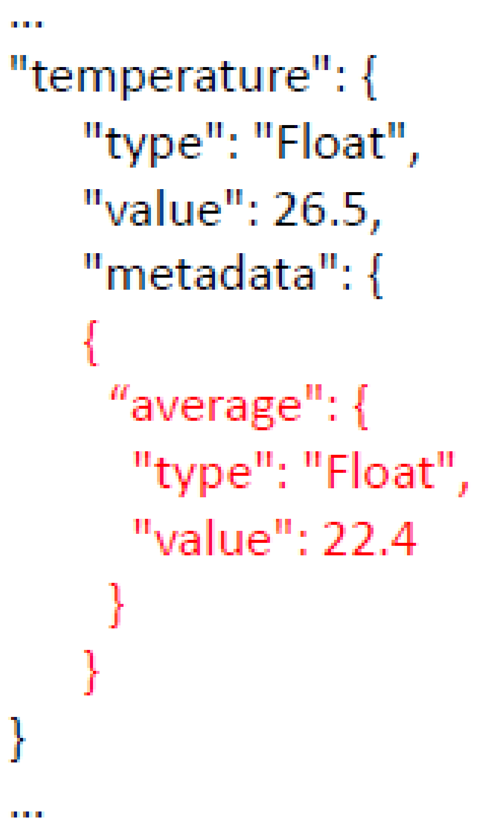


```
201 Created
...
```

Metadata



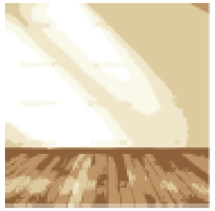
```
...
"temperature": {
    "type": "Float",
    "value": 26.5,
    "metadata": {
    {
     "accuracy": {
      "type": "Float",
      "value": 0.9
     }
    }
}
...
```



```
...
"temperature": {
    "type": "Float",
    "value": 26.5,
    "metadata": {
    {
     “average": {
      "type": "Float",
      "value": 22.4
     }
    }
}
...
```

Update Room

```
PATCH <cb_host>:1026/v2/entities/Room1/attrs
Content-Type: application/json
...

{
  "temperature“: {
    "type": "Float",
    "value": 25
  },
  "pressure": {
    "type": "Integer",
    "value":  720
  }
}
```



```
204 No Content
...
```

Query Room

```
GET <cb_host>:1026/v2/entities/Room1/attrs
```

```
200 OK
Content-Type: application/json
...

{
  "pressure": {
    "type": "Integer",
    "value": 720,
    "metadata": {}
  },
  "temperature": {
    "type": "Float",
    "value": 25,
    "metadata": {}
  }
}
```

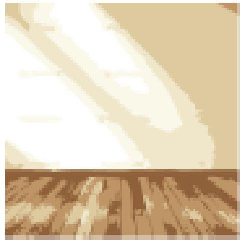
Query Room

```
GET <cb_host>:1026/v2/entities/Room1/attrs?options=keyValues
```

```
200 OK
Content-Type: application/json
...

{
   "pressure": 720,
   "temperature": 25
}
```

Query Room

```
POST <cb_host>:1026/v2/entities
Content-Type: application/json
...

{
  "id": "Room2",
  "type": "Room",
  "temperature": {
    "type": "Float",
    "value": 29
  },
  "pressure": {
    "type": "Integer",
    "value": 730
  }
}
```



```
201 Created
...
```

Filter Room

```
GET <cb_host>:1026/v2/entities?options=keyValues&q=temperature>27
```

```
200 OK
Content-Type: application/json
...

[
  {
    "id": "Room2",
    "pressure": 730,
    "temperature": 29,
    "type": "Room"
  }
]
```

Filter Room

```
GET <cb_host>:1026/v2/entities?options=keyValues&q=pressure==715..725
```

The full description of the Simple Query Language for filtering can be found in the NGSIv2 Specification document

```
200 OK
Content-Type: application/json
...

[
  {
    "id": "Room1",
    "pressure": 720,
    "temperature": 25,
    "type": "Room"
  }
]
```

Exercise

- Create the entities using the global instance.

Hands on – Create Entities

Entity	Entity Type
Bedroom1	Room
Bedroom2	Room
Kitchen	Room
Frontdoor	Door
Backdoor	Door

Entity Type	Attr. Name	Attr. Type	Example value
Room	Temperature	float	27.8
	Presence	boolean	true
	Status	string	OK
Door	Locked	boolean	false
	Closed	boolean	false

Hands on – Update Entities

Exercise

- Updates **Locked** attribute of **Frontdoor** entity using a valid input.
- Queries the entity and check the result.

Query Language

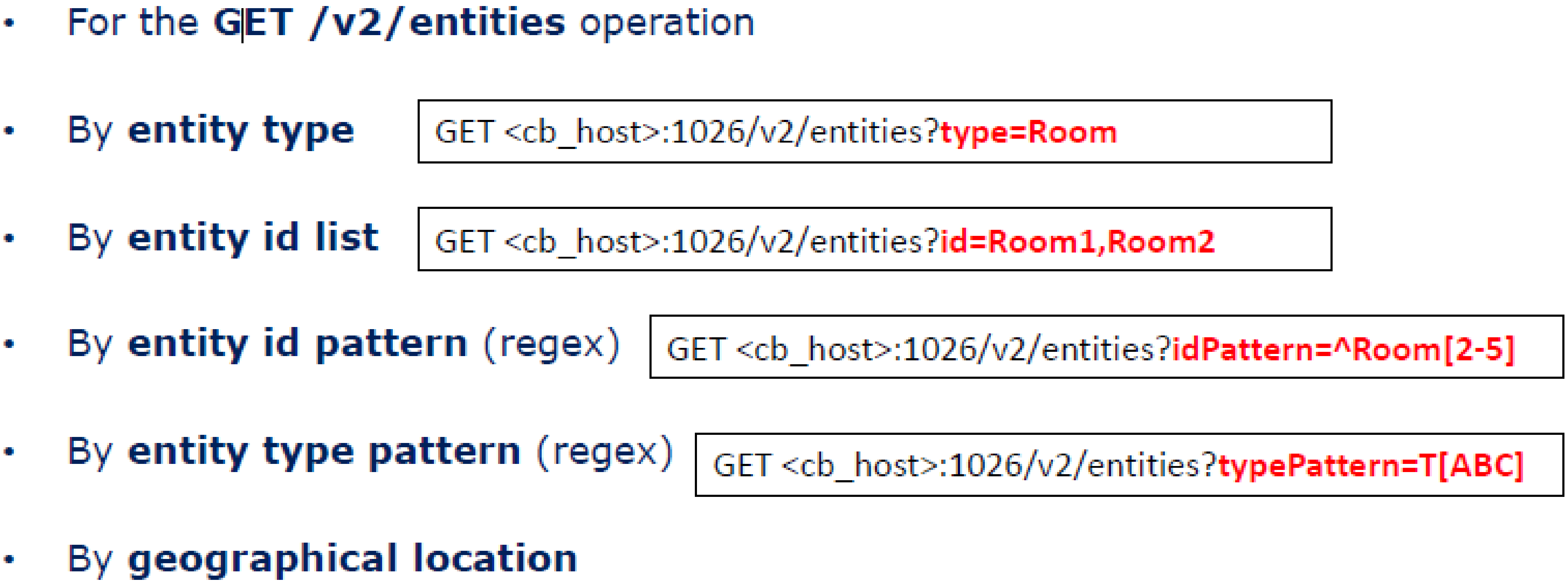
FIWARE
Mexico

Query Language

- inside of **options:**
  - **dateCreated** (type:DateTime) ISO 8601.
  - **dateModified** (type:DateTime) ISO 8601.

- Like regular attributes, the can be used in **attrs, q** filters and **order by.**

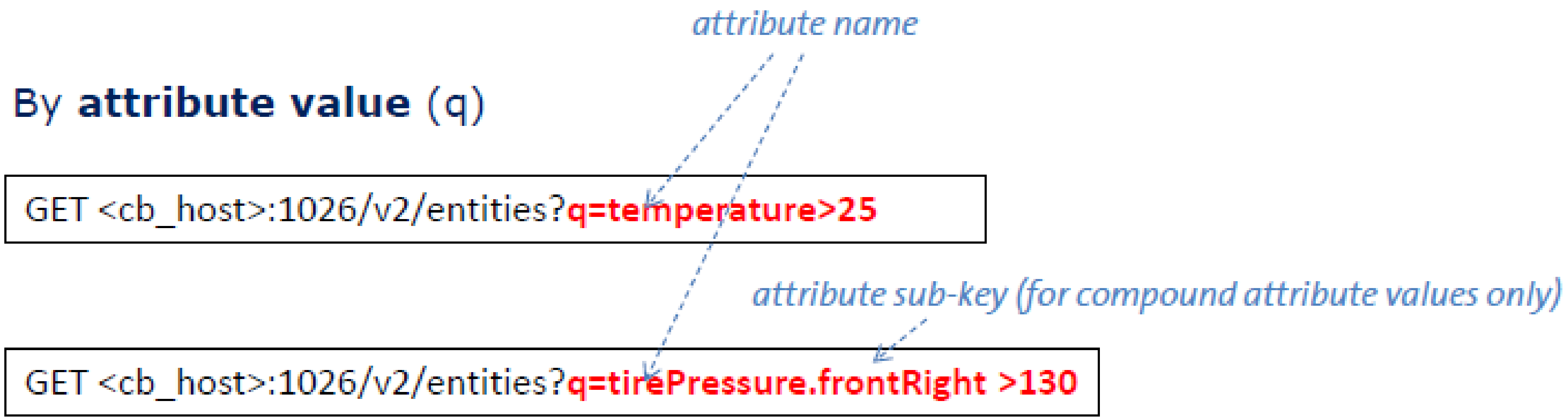
Query Filters



- For the **GET /v2/entities** operation
- By **entity type** `GET <cb_host>:1026/v2/entities?type=Room`
- By **entity id list** `GET <cb_host>:1026/v2/entities?id=Room1,Room2`
- By **entity id pattern** (regex) `GET <cb_host>:1026/v2/entities?idPattern=^Room[2-5]`
- By **entity type pattern** (regex) `GET <cb_host>:1026/v2/entities?typePattern=T[ABC]`
- By **geographical location**

Query Filters

- By **attribute value** (q)



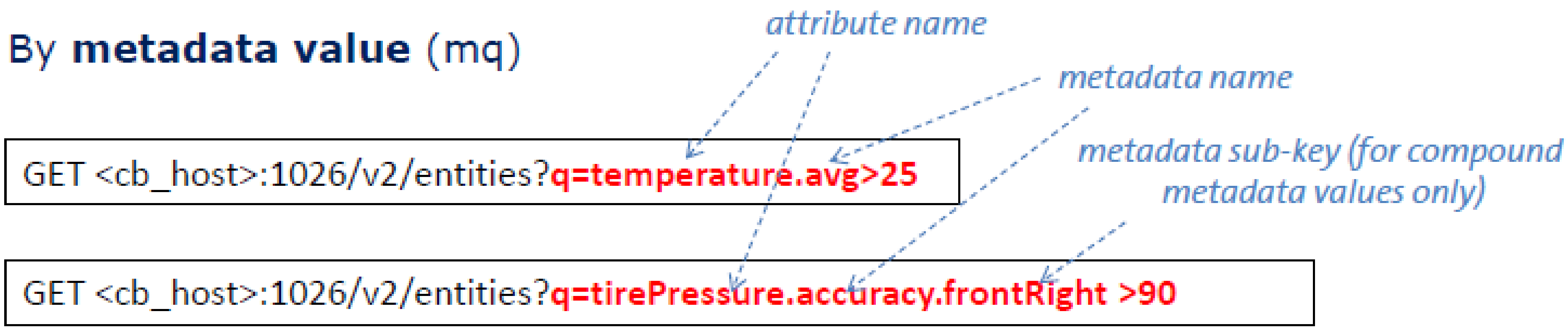
attribute name

```
GET <cb_host>:1026/v2/entities?q=temperature>25
```

attribute sub-key (for compound attribute values only)

```
GET <cb_host>:1026/v2/entities?q=tirePressure.frontRight >130
```

- By **metadata value** (mq)



attribute name

metadata name

metadata sub-key (for compound metadata values only)

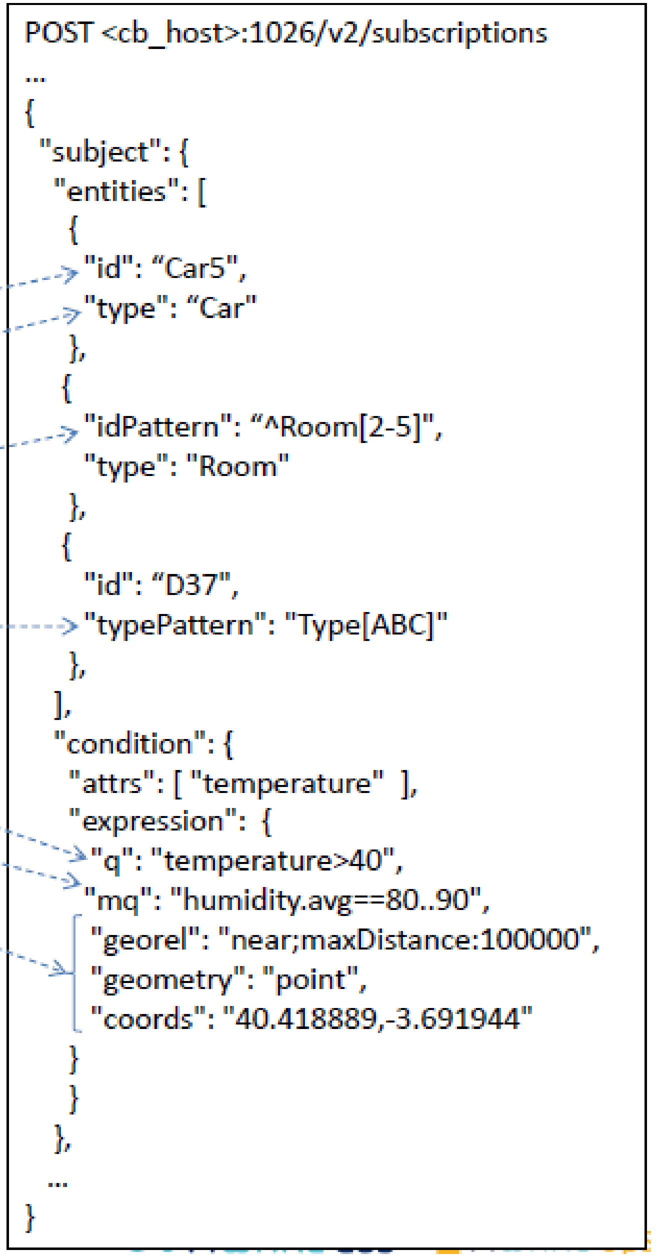
```
GET <cb_host>:1026/v2/entities?q=temperature.avg>25
```

```
GET <cb_host>:1026/v2/entities?q=tirePressure.accuracy.frontRight >90
```

- See full details about **q** and **mq** query language in NGSIv2 specification

Query filters

- Filters can be also used in subscriptions
  - id
  - type
  - id pattern
  - type pattern
  - attribute values
  - metadata value
  - geographical location



```
POST <cb_host>:1026/v2/subscriptions
...
{
 "subject": {
  "entities": [
   {
    "id": “Car5",
    "type": “Car"
   },
   {
    "idPattern": “^Room[2-5]",
    "type": "Room"
   },
   {
    "id": “D37",
    "typePattern": "Type[ABC]"
   },
  ],
  "condition": {
   "attrs": [ "temperature"  ],
   "expression": {
    "q": "temperature>40",
    "mq": "humidity.avg==80..90",
    "georel": "near;maxDistance:100000",
    "geometry": "point",
    "coords": "40.418889,-3.691944"
   }
  }
 },
 ...
}
```

Hands on – Query Entities

Exercise

- Obtain all attributes of **Bedroom1** entity.
- Obtain only the **Temperature** attribute of Kitchen entity.
- Obtain all attributes of **Kitchen** and **Bedroom2** entities in one query.
- Obtain all attributes of entities that match the pattern Bedroom.*

Exercise - Finally

- Find out whether the doors are closed using the pattern **.*door** and the **Closed** attribute

Suscriptions

FIWARE
Mexico

Hands on – Update Entities

Exercise

- Updates the **Temperature** attribute of **Beedroom1** and **Bedroom2** entities using that input with a single update operation.
- Queries the entities and check the result.

Hands on – Create Entities (Again)

Exercise

Entity	Entity Type
Garage	Room
Bathroom	Room
Light1	Light
Light2	Light
Light3	Light

Entity Type	Attr. Name	Attr. Type	Example value
Room	Temperature	float	27.8
	Presence	boolean	true
	Status	string	OK
Light	Intensity	Percent	0.25

Hands on – List Entity Types

Exercise

- Lists all entity types
- Provides detailed information of type Door

Geo Localization

FIWARE
Mexico

Geo-location

- Entities can have an attribute that specifies its location
- Several attribute types can be used
  - geo:point (for points)
  - geo:line (for lines)
  - geo:box (for boxes)
  - geo:polygon (for polygons)
  - geo:json (for arbitrary geometries, in GeoJson standard)
- Example: create an entity called Madrid

  …and create a couple more towns:
  - Leganés
  - Alcobendas

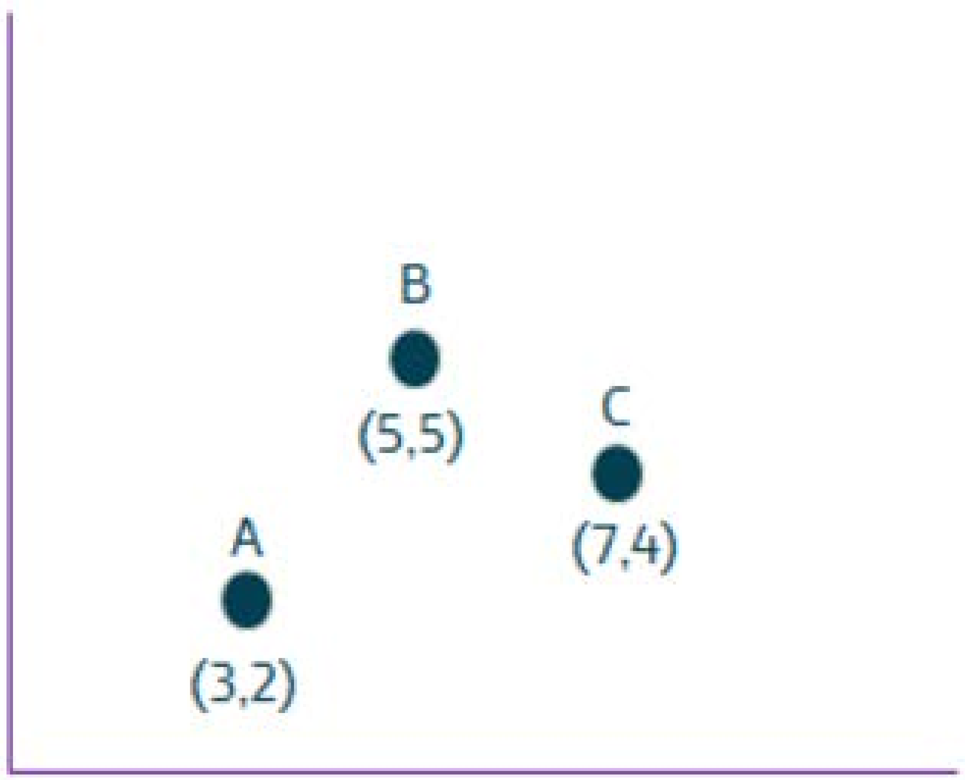
```
POST <cb_host>:1026/v2/entities
{
  "type": "City",
  "id": "Madrid",
  "position": {
    "type": "geo:point",
    "value": "40.418889, -3.691944"
  }
}
```

More geo-relationships

- Apart from near, the following georel can be used
  - georel=coveredBy
  - georel=intersects
  - georel=equals
  - georel=disjoint

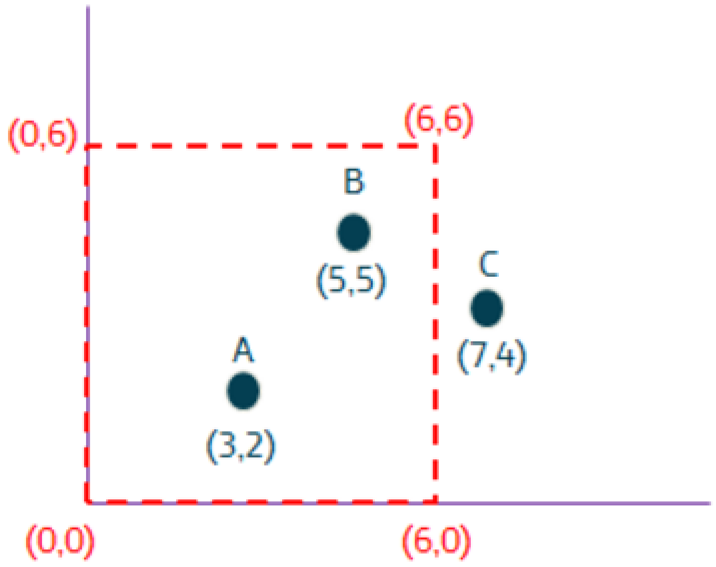
Example

- Let consider the next elementos:



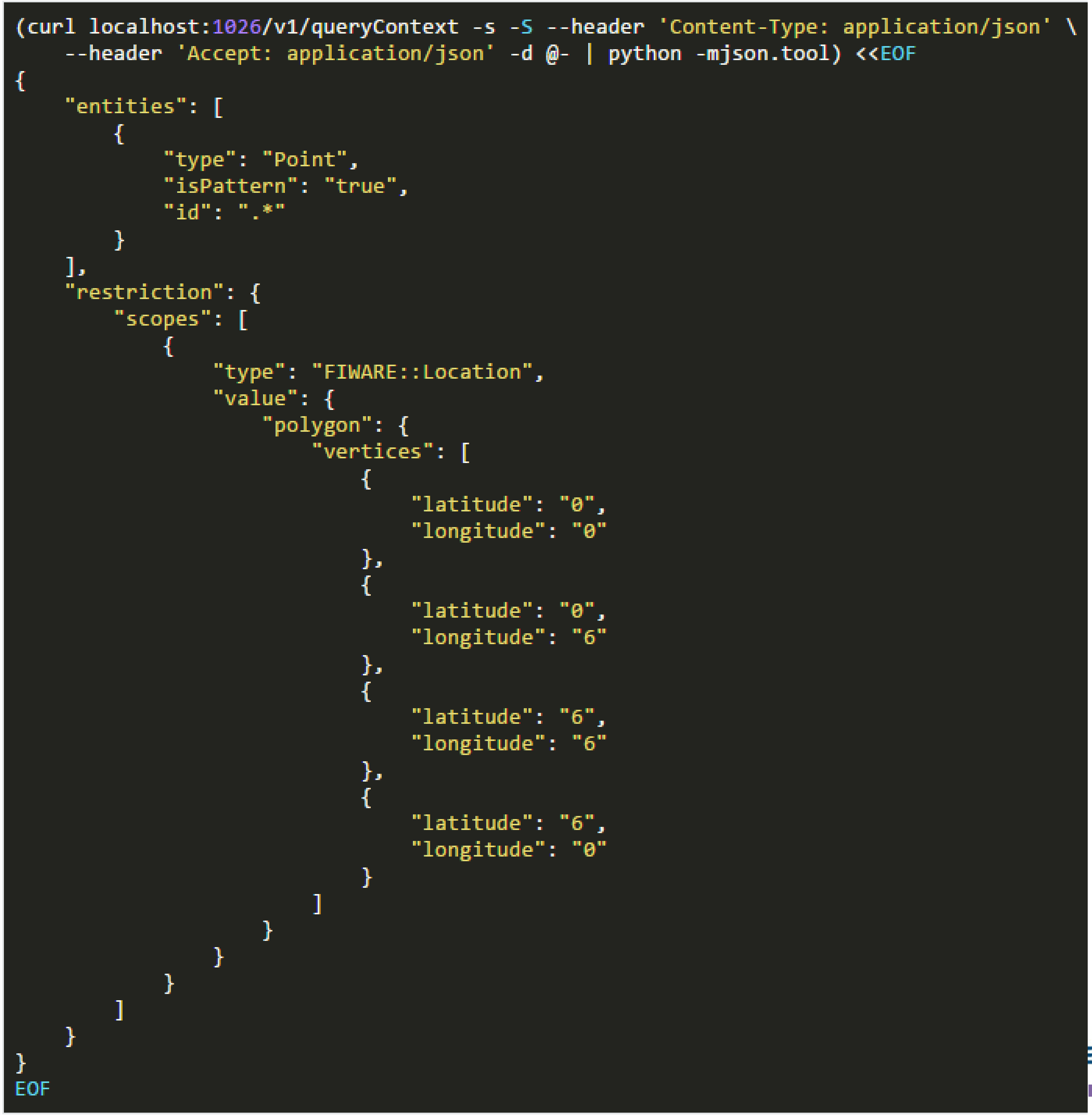
Example

- Let´s consider a query whose scope is the internal area to the square defined by coordinates **(0,0) , (0,6)** , **(6,6)** and **(6,0)**



The result of the query is

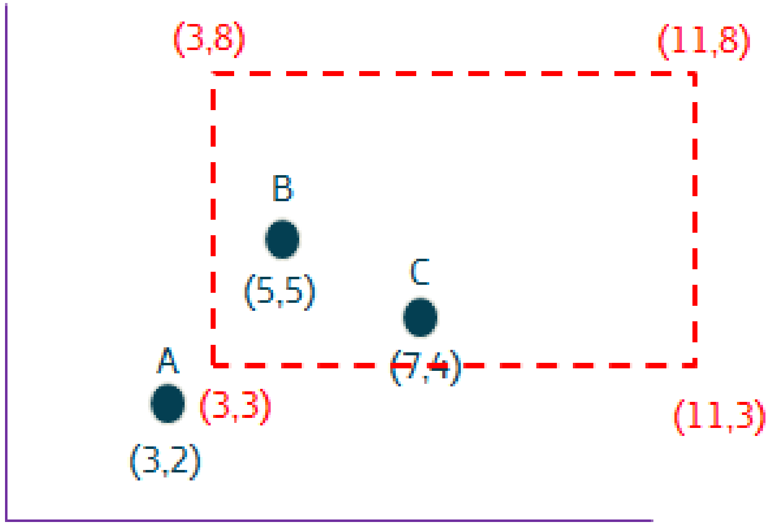
A and **B**



```
(curl localhost:1026/v1/queryContext -s -S --header 'Content-Type: application/json' \
    --header 'Accept: application/json' -d @- | python -mjson.tool) <<EOF
{
    "entities": [
        {
            "type": "Point",
            "isPattern": "true",
            "id": ".*"
        }
    ],
    "restriction": {
        "scopes": [
            {
                "type": "FIWARE::Location",
                "value": {
                    "polygon": {
                        "vertices": [
                            {
                                "latitude": "0",
                                "longitude": "0"
                            },
                            {
                                "latitude": "0",
                                "longitude": "6"
                            },
                            {
                                "latitude": "6",
                                "longitude": "6"
                            },
                            {
                                "latitude": "6",
                                "longitude": "0"
                            }
                        ]
                    }
                }
            }
        ]
    }
}
EOF
```

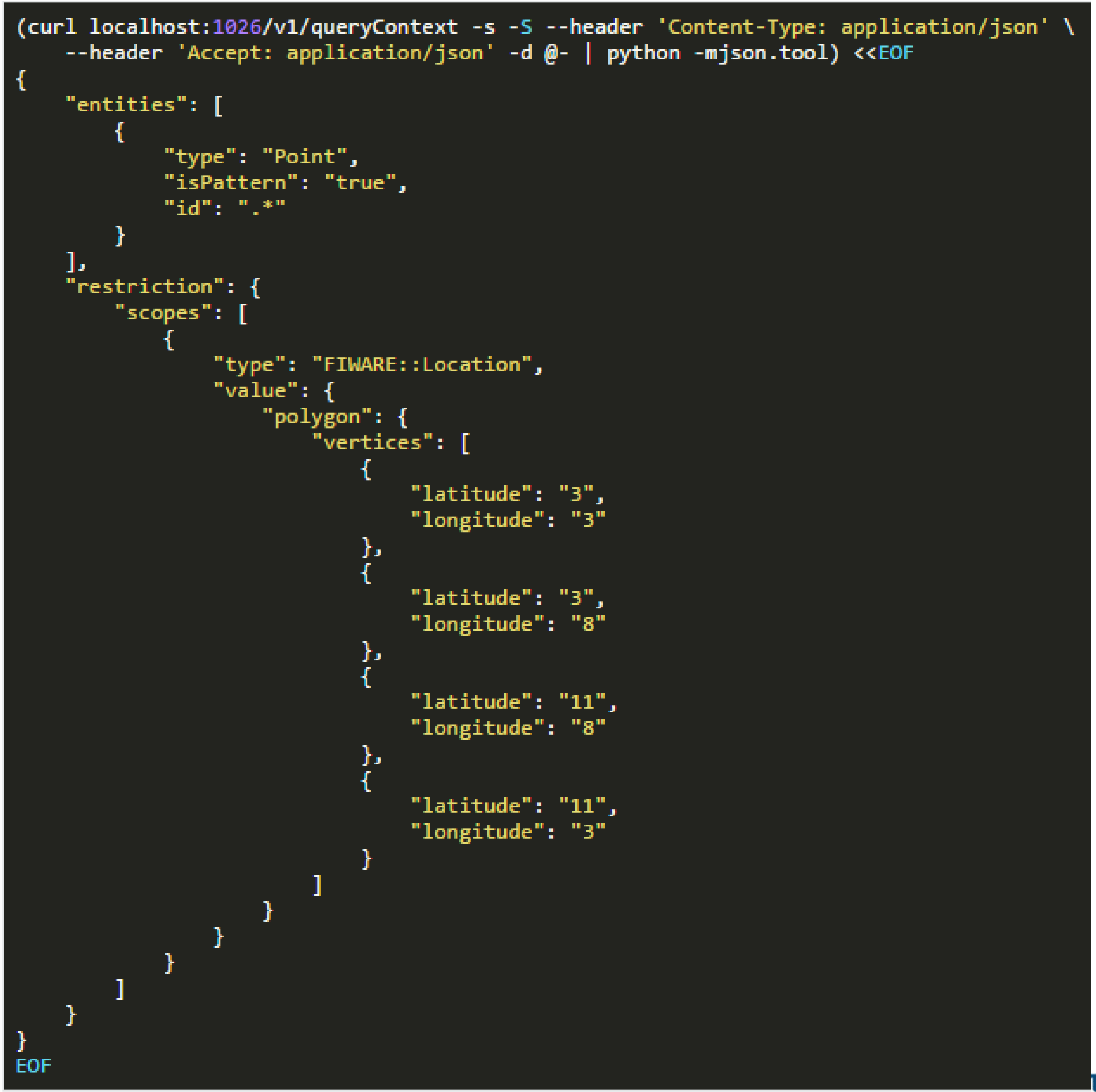
Example

- Let´s consider a query whose scope is the internal area to the rectangle defined by coordinates **(3,3) , (3,8)** , **(11,8)** and **(11,3)**



The result of the query is

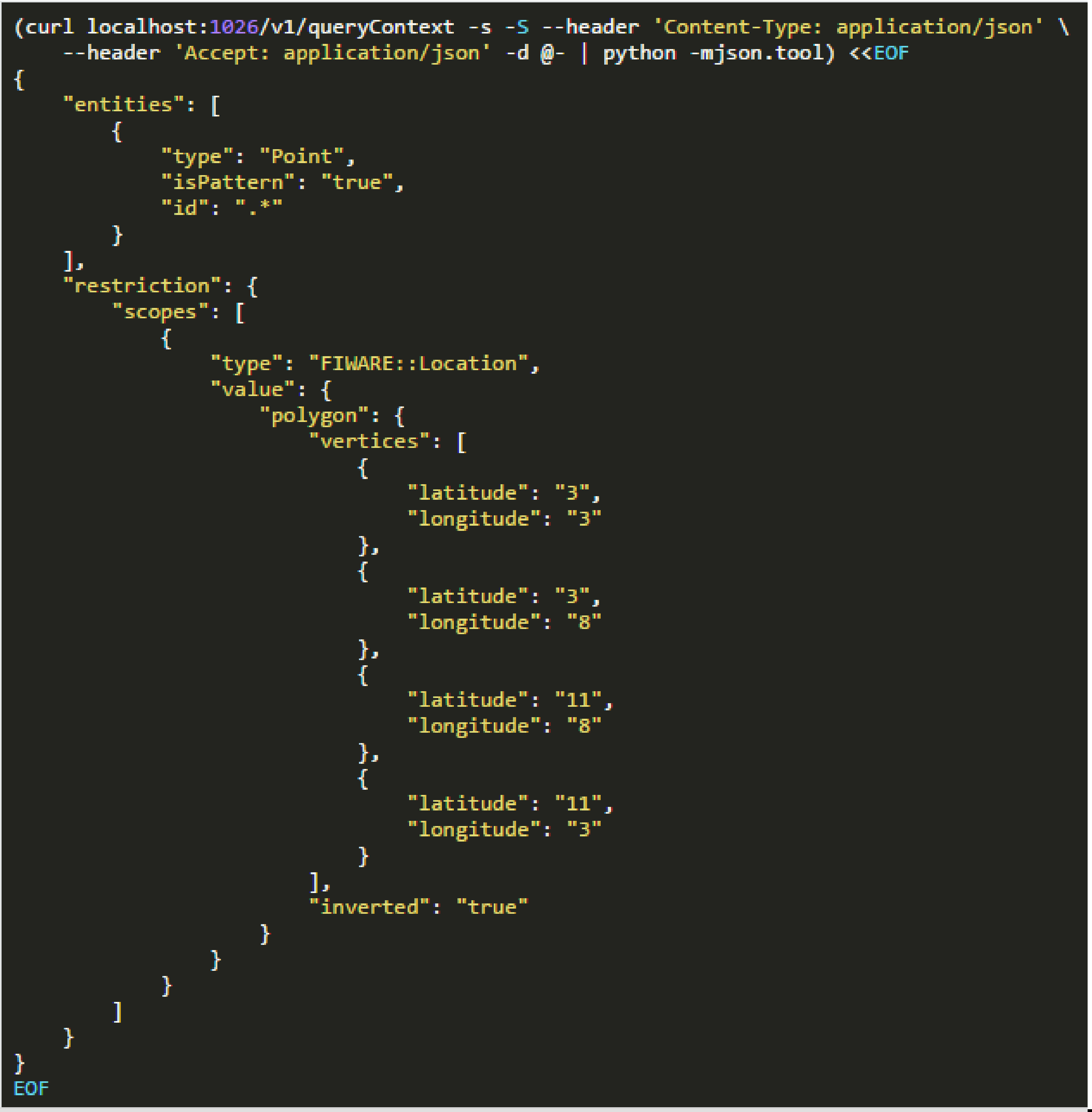
B and **C**



```
(curl localhost:1026/v1/queryContext -s -S --header 'Content-Type: application/json' \
    --header 'Accept: application/json' -d @- | python -mjson.tool) <<EOF
{
    "entities": [
        {
            "type": "Point",
            "isPattern": "true",
            "id": ".*"
        }
    ],
    "restriction": {
        "scopes": [
            {
                "type": "FIWARE::Location",
                "value": {
                    "polygon": {
                        "vertices": [
                            {
                                "latitude": "3",
                                "longitude": "3"
                            },
                            {
                                "latitude": "3",
                                "longitude": "8"
                            },
                            {
                                "latitude": "11",
                                "longitude": "8"
                            },
                            {
                                "latitude": "11",
                                "longitude": "3"
                            }
                        ]
                    }
                }
            }
        ]
    }
}
EOF
```

Example

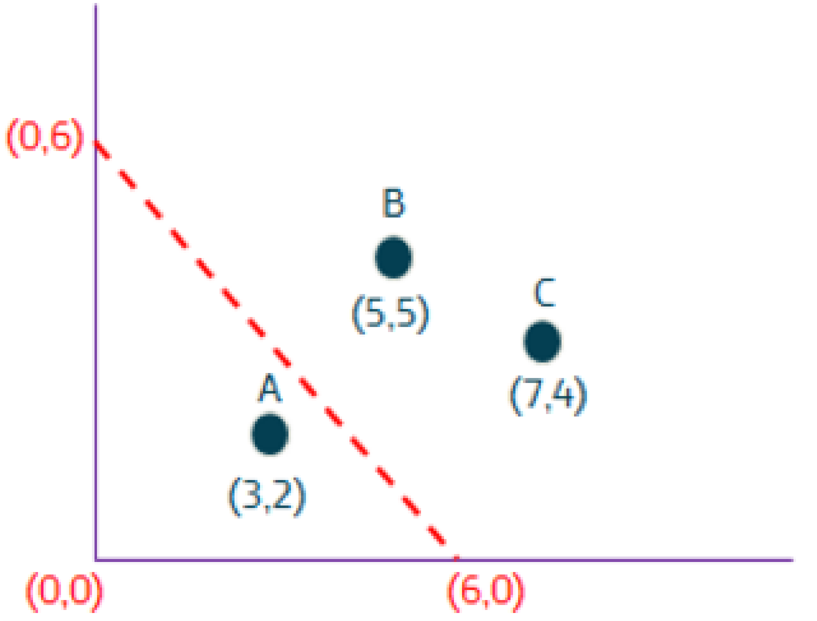
- if we consider the query to the external area to that rectangle, the result of the query would be **A.** To specify that, we refer to the area external to the polygon we include the inverted element set to "**true**"

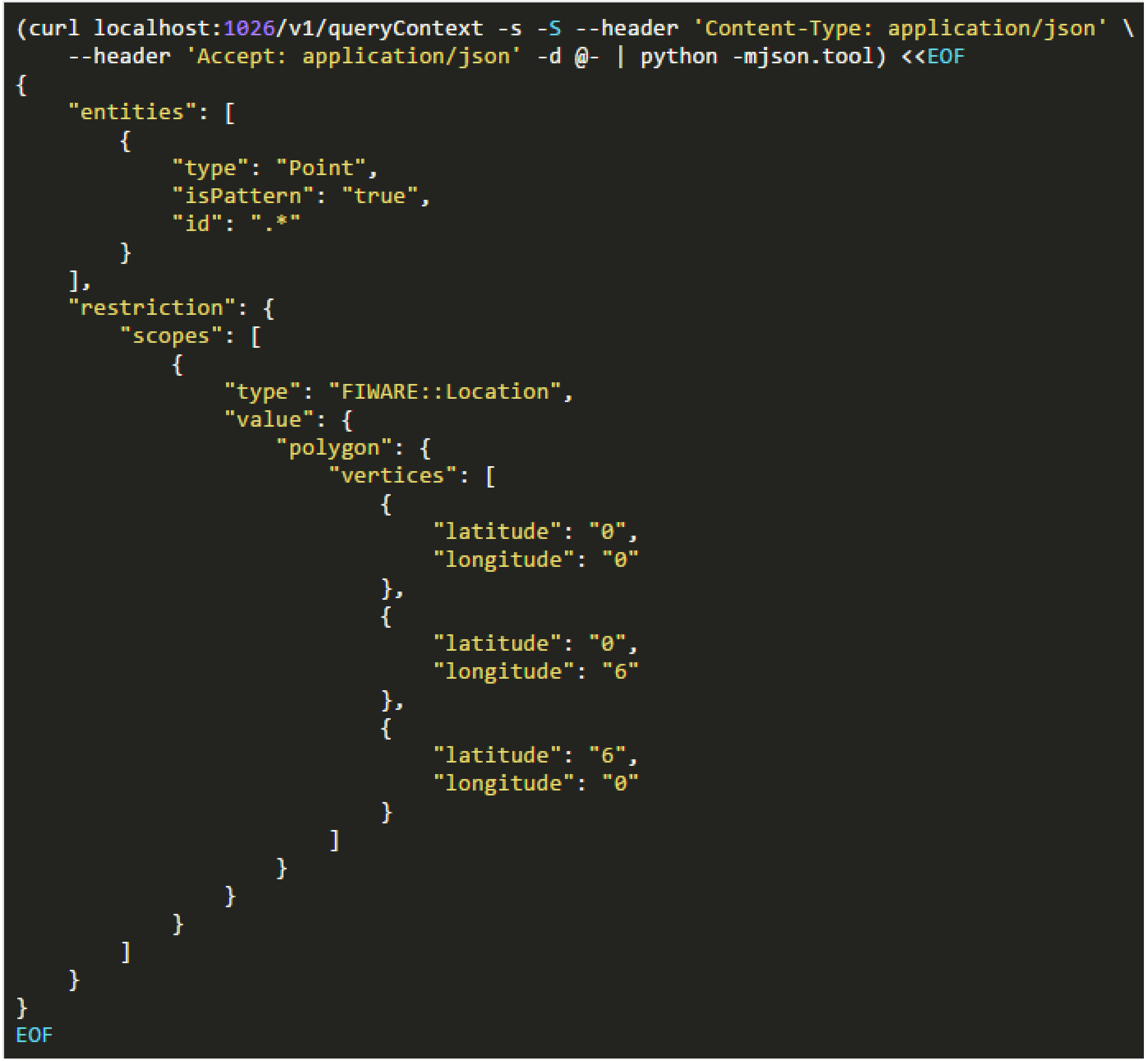


```
(curl localhost:1026/v1/queryContext -s -S --header 'Content-Type: application/json' \
    --header 'Accept: application/json' -d @- | python -mjson.tool) <<EOF
{
    "entities": [
        {
            "type": "Point",
            "isPattern": "true",
            "id": ".*"
        }
    ],
    "restriction": {
        "scopes": [
            {
                "type": "FIWARE::Location",
                "value": {
                    "polygon": {
                        "vertices": [
                            {
                                "latitude": "3",
                                "longitude": "3"
                            },
                            {
                                "latitude": "3",
                                "longitude": "8"
                            },
                            {
                                "latitude": "11",
                                "longitude": "8"
                            },
                            {
                                "latitude": "11",
                                "longitude": "3"
                            }
                        ],
                        "inverted": "true"
                    }
                }
            }
        ]
    }
}
EOF
```

Example

- Let´s consider a query whose scope is the internal area to the triangle defined by coordinates (0,0), (0,6), (6,0).

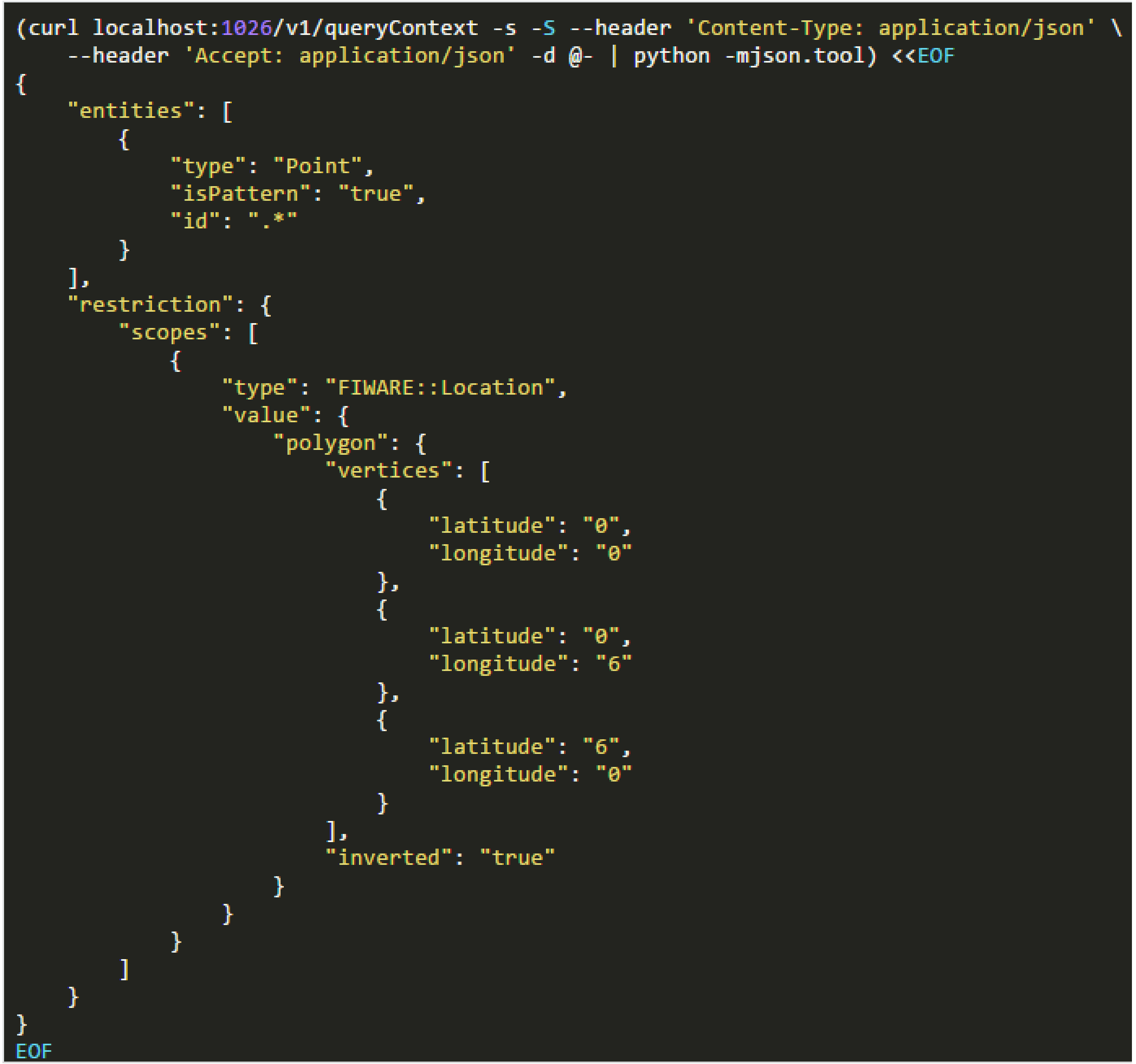




```
(curl localhost:1026/v1/queryContext -s -S --header 'Content-Type: application/json' \
    --header 'Accept: application/json' -d @- | python -mjson.tool) <<EOF
{
    "entities": [
        {
            "type": "Point",
            "isPattern": "true",
            "id": ".*"
        }
    ],
    "restriction": {
        "scopes": [
            {
                "type": "FIWARE::Location",
                "value": {
                    "polygon": {
                        "vertices": [
                            {
                                "latitude": "0",
                                "longitude": "0"
                            },
                            {
                                "latitude": "0",
                                "longitude": "6"
                            },
                            {
                                "latitude": "6",
                                "longitude": "0"
                            }
                        ]
                    }
                }
            }
        ]
    }
}
EOF
```

Example

- However, if we consider the query to the external area to that triangle (using the inverted element set to “true”), the result of the query would be **B** and **C.**



```
(curl localhost:1026/v1/queryContext -s -S --header 'Content-Type: application/json' \
    --header 'Accept: application/json' -d @- | python -mjson.tool) <<EOF
{
    "entities": [
        {
            "type": "Point",
            "isPattern": "true",
            "id": ".*"
        }
    ],
    "restriction": {
        "scopes": [
            {
                "type": "FIWARE::Location",
                "value": {
                    "polygon": {
                        "vertices": [
                            {
                                "latitude": "0",
                                "longitude": "0"
                            },
                            {
                                "latitude": "0",
                                "longitude": "6"
                            },
                            {
                                "latitude": "6",
                                "longitude": "0"
                            }
                        ],
                        "inverted": "true"
                    }
                }
            }
        ]
    }
}
EOF
```

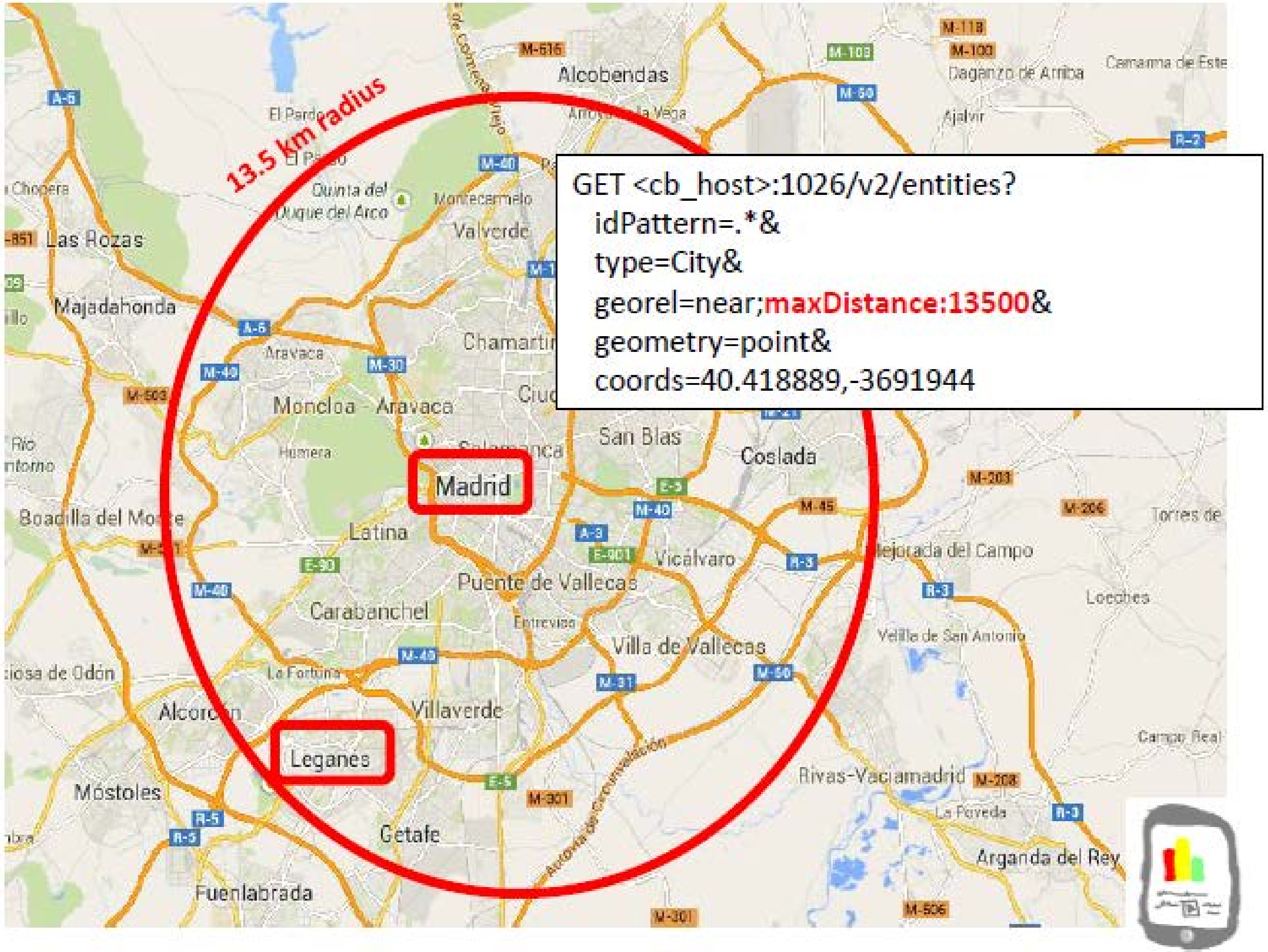
Real Use Case

- **Three entities** (representing the cities of **Madrid**, **Alcobendas** and **Leganes**) have been created in **Orion Context Broker.**
- The coordinates for **Madrid** are **(40.418889, -3.691944)**; the coordinates for **Alcobendas** are **(40.533333, -3.633333)** and the coordinates for **Leganes** are **(40.316667, -3.75).**

Real Use Case

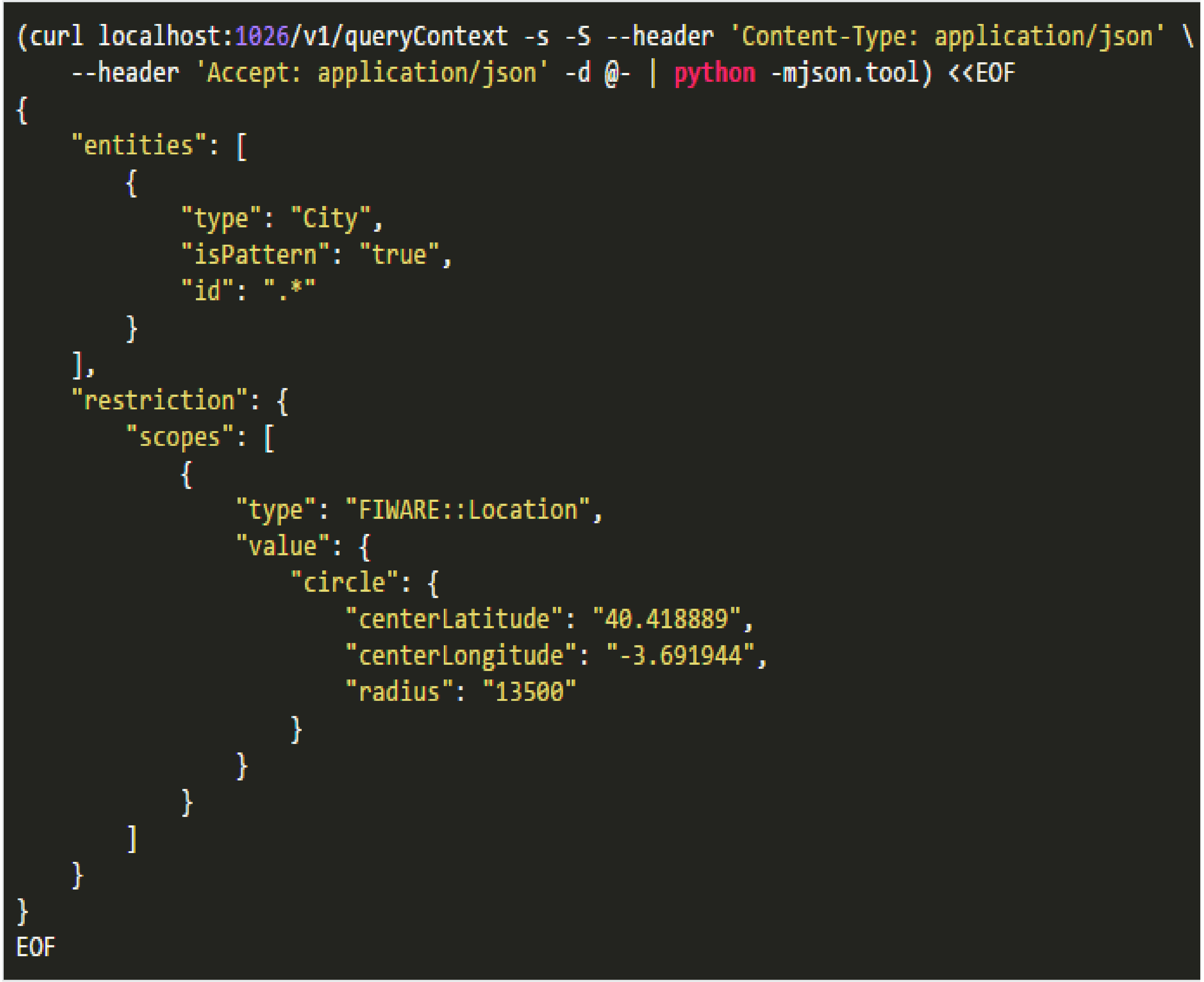
- Let's consider a query whose scope is inside a radius of 13.5 km (13500 meters) centred in Madrid.

Geo-location- Max distance



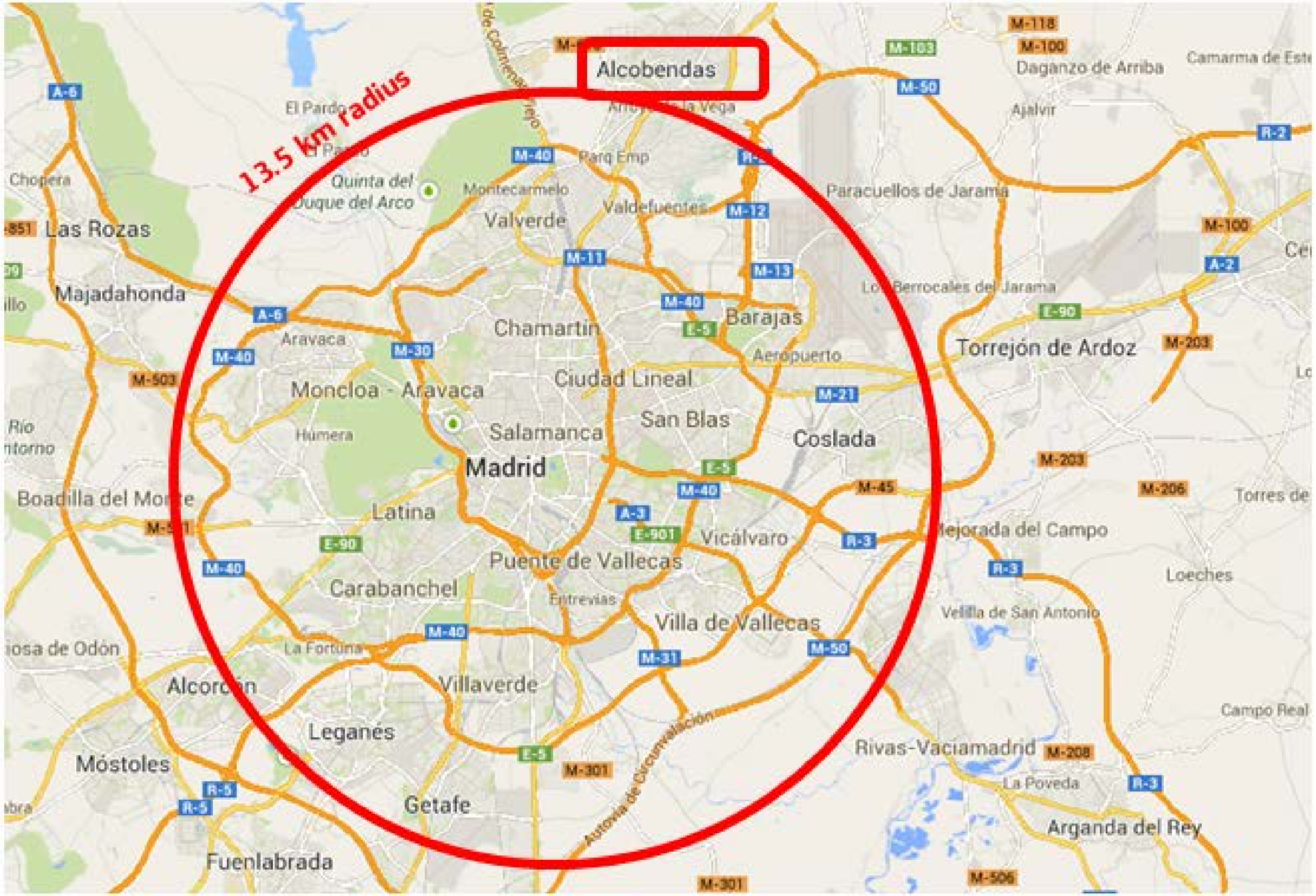
```
GET <cb_host>:1026/v2/entities?
  idPattern=.*&
  type=City&
  georel=near;maxDistance:13500&
  geometry=point&
  coords=40.418889,-3691944
```

The query is:

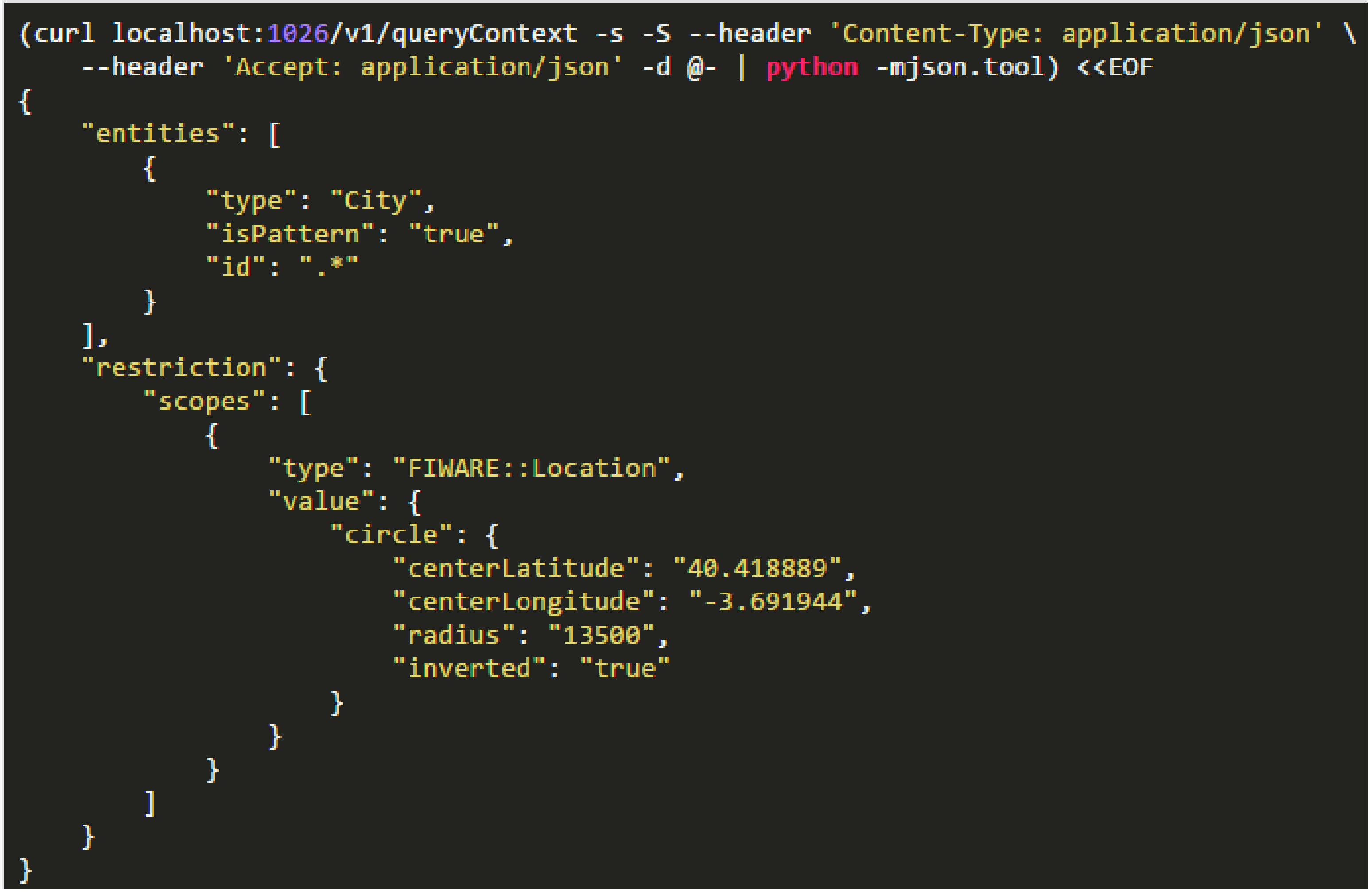


```
(curl localhost:1026/v1/queryContext -s -S --header 'Content-Type: application/json' \
    --header 'Accept: application/json' -d @- | python -mjson.tool) <<EOF
{
    "entities": [
        {
            "type": "City",
            "isPattern": "true",
            "id": ".*"
        }
    ],
    "restriction": {
        "scopes": [
            {
                "type": "FIWARE::Location",
                "value": {
                    "circle": {
                        "centerLatitude": "40.418889",
                        "centerLongitude": "-3.691944",
                        "radius": "13500"
                    }
                }
            }
        ]
    }
}
EOF
```

Geo-location- Inverted



Geo-location- Min distance



```
(curl localhost:1026/v1/queryContext -s -S --header 'Content-Type: application/json' \
    --header 'Accept: application/json' -d @- | python -mjson.tool) <<EOF
{
    "entities": [
        {
            "type": "City",
            "isPattern": "true",
            "id": ".*"
        }
    ],
    "restriction": {
        "scopes": [
            {
                "type": "FIWARE::Location",
                "value": {
                    "circle": {
                        "centerLatitude": "40.418889",
                        "centerLongitude": "-3.691944",
                        "radius": "13500",
                        "inverted": "true"
                    }
                }
            }
        ]
    }
}
```

Hands on – Geo Location

Batch Operations

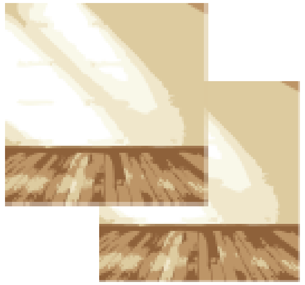
FIWARE

Mexico

Batch Operations

```
POST <cb_host>:1026/v2/op/update
Conten-Type: application/json
...

{
 "actionType": "APPEND",
 "entities": [
  {
   "type": "Room",
   "id": "Room3",
   "temperature": {
    "value": 21.2,
    "type": "Float"
   },
   "pressure": {
    "value": 722,
    "type": "Integer"
   }
  },
...
```

```
...
  {
   "type": "Room",
   "id": "Room4",
   "temperature": {
    "value": 31.8,
    "type": "Float"
   },
   "pressure": {
    "value": 712,
    "type": "Integer"
   }
  }
 ]
}
```



```
201 Created
...
```

Pagination

FIWARE Mexico

Elements

- **limit**: Number of elements per page (default: 20, max: 1000)
- **offset**: Number of elements to skip (default: 0)
- **count** (optional): Returns total elements (default: not return)

Example

- GET <orion_host>:1026/v2/entities?limit=5
- GET <orion_host>:1026/v2/entities?offset=5&limit=5
- GET <orion_host>:1026/v2/entities?offset=10&limit=5
- GET <orion_host>:1026/v2/entities?offset=15&limit=5

Considerations:

- By default, results are ordered by entity creation date
- This behavior can be overridden using orderBy URI parameter
- Example: get the first 10 entities ordered by temp in ascending order, then humidity in descending order

GET <orion_host>:1026/v2/entities?limit=20&offset=0&orderBy=temp,!humidity

Considerations (Continues..)

- **dateCreated** and **dateModified** can be used to ordering by entity creation and modification date, respectively

The end

FIWARE
Mexico