NGSIv2 Overview for Developers That Already Know NGSIv1

FIWARE

OPEN APIs FOR OPEN MINDS

FIWARE Lab

Spark your imagination

FIWARE Ops

Easing your operations

www.fiware.org
@Fiware

Contact twitter
@fermingalan

Contact email
fermin.galanmarquez@telefonica.com

(Reference Orion Context Broker version: 1.4.0)

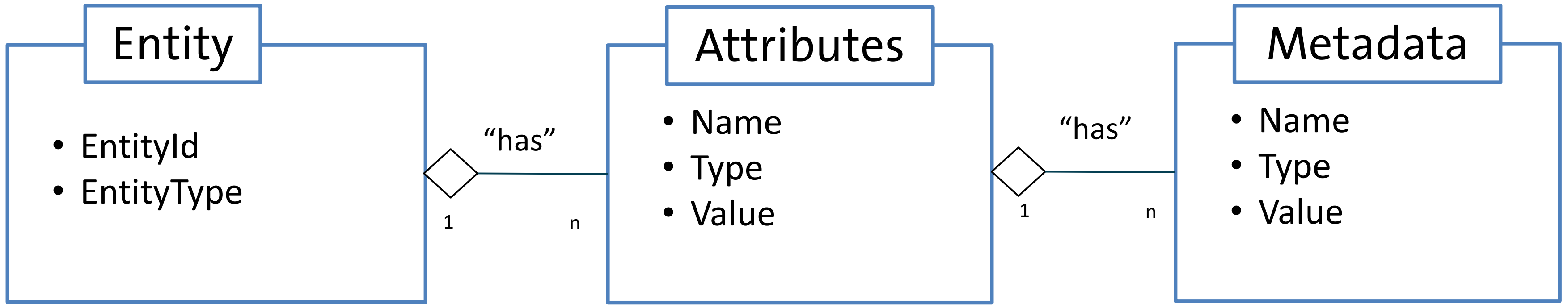
Outline

- Introduction to NGSIv2
- RESTful-ness
- URL & payload simplification
- Native JSON datatypes
- Text-based attribute value set/get
- Geolocation capabilities
- Filtering
- Datetime support
- Subscription improvements
- Batch operations
- Working with IDs
- Pagination
- Next things to come

Two "flavors" of NGSI API

- NGSIv1 (the one you already know ☺)
  - Original NGSI RESTful binding of OMA-NGSI
  - Implemented in 2013
  - Uses the **/v1** prefix in resource URL
- NGSIv2
  - A revamped, simplified binding of OMA-NGSI
    - Simple things must be easy
    - Complex things should be possible
    - Agile, implementation-driven approach
    - Make it as developer-friendly as possible (RESTful, JSON, …)
  - Enhanced functionality compared to NGSIv1 (e.g. filtering)
  - Stable, ready for production, version already available
    - Current NGSIv2 version is **Release Candidate 2016.05** http://telefonicaid.github.io/fiware-orion/api/v2/stable
    - New features coming (see last part of this presentation)
  - Uses the **/v2** prefix in resource URL
- Introduction to NGSIv2
  - https://docs.google.com/presentation/d/1_fv9dB5joCsOCHlb4Ld6A-QmeIYhDzHgFHUWreGmvKU/edit#slide=id.g53c31d7074fd7bc7_0

The NGSI model is kept as it is in NGSIv2



NGSIv2 status (AKA the “NGSIv2 disclaimer”)

- NGSIv2 is in “release candidate” status
  - By "release candidate" we mean that the specification is quite stable, but changes may occur with regard to new release candidates or the final version. In particular, changes may be of two types:
    - Extensions to the functionality currently specified by this document. Note that in this case there isn't any risk of breaking backward compatibility on existing software implementations.
    - Slight modifications in the functionality currently specified by this document, as a consequence of ongoing discussions. Backward compatibility will be taken into account in this case, trying to minimize the impact on existing software implementations. In particular, only completely justified changes impacting backward compatibility will be allowed and "matter of taste" changes will not be allowed.

So… when should I use NGSIv1 or NGSIv2?

- In general, **it is always preferable to use NGSIv2**
- However, you would need to use NGSIv1 if
  - You need register/discovery operations (context management availability functionality)
    - Not yet implemented in NGSIv2 (in roadmap)
  - Zero tolerance to changes in software interacting with Orion
- Even if you use NGSIv1, you can still use NGSIv2 functionality and requests
  - See "Considerations on NGSIv1 and NGSIv2 coexistence" section in the Orion manual

RESTful-ness

- In NGSIv1
  - Originally based on OMA-NGSI standard operations, not really RESTful
    - The URL doesn’t identify a resource, but an operation type
    - The verb is always POST
    - Actually, NGSIv1 is closer to HTTP-based RPC than to RESTful
  - An extended set of operations (convenience operations) were added in a later stage but with “legacy” from standard operations that make it hard to apply full RESTful principles
    - E.g. dual response code
- NGSIv2 has been designed from scratch with RESTful principles in mind
  - Batch operations (similar to standard operations in NGSIv1) are also provided, but without impacting the RESTful operations in the API

RESTful-ness

Example: get Car1 entity

NGSIv1

```
GET /v1/contextEntities/Car1
```

```
200 OK
...
{
  "contextElement" : {
    "type" : "",
    "isPattern" : "false",
    "id" : "Car1"
  },
  "statusCode" : {
    "code" : "404",
    "reasonPhrase" : "No context element found",
    "details" : "Entity id: /Car1/"
  }
}
```

both status codes have to be taken into account by the client in order to detect error conditions, which increases complexity

NGSIv2

```
GET /v2/entities/Car1
```

```
404 Not Found
...
{
  "error": "NotFound",
  "description": "The requested entity has not
been found. Check type and id“
}
```

only HTTP response code, following RESTful principles - simpler to process

URL & payload simplification

- In NGSIv1 the strict alignment with OMA NGSI involves complexity and ineffectiveness
  - Message payloads include a lot of structural overhead elements not actually needed
  - Response payload in methods that don’t really need a payload from a semantic point of view (e.g. update operation)
  - Unnecessarily long structural elements in URLs
- NGSIv2 simplifies URLs and payload, leading to a much more lean and less verbose API

URL & payload simplification

Example: create Car1 entity (type Car) with attribute *colour* set to "Red"

NGSIv1

```
POST /v1/contextEntities
...
{
  "id": "Car1",
  "type": "Car",
  "attributes": [
  {
    "name": "colour",
    "type": "Text",
    "value": "red"
  }
  ]
}
```

```
200 OK
...
{
  "contextResponses": [
    {
      "attributes": [
        {
          "name": "colour",
          "type": "float",
          "value": ""
        }
      ],
      "statusCode": {
        "code": "200",
        "reasonPhrase": "OK"
      }
    }
  ],
  "id": "Car1",
  "isPattern": "false",
  "type": "Car"
}
```

shorter URLs in NGSIv2

structural overhead

NGSIv2

```
POST /v2/entities
...
{
  "id": "Car1",
  "type": "Car",
  "colour": {
    "value": "red",
    "type": "Text",
  }
}
```

```
200 OK
Location: /v2/entities?type=Car
```

Most of the NGSIv1 response payload is useless: clients only need to know the status code. In NGSIv2 the response has no payload at all

URL & payload simplification

NGSIv1

```
GET /v1/contextEntities/type/Car/id/Car1/attributes/colour
```

```
200 OK
...
{
  "attributes": [
    {
      "name": "colour",
      "type": "Text",
      "value": " red"
    }
  ],
  "statusCode": {
    "code": "200",
    "reasonPhrase": "OK"
  }
}
```

NGSIv2

```
GET /v2/entities/Car1/attrs/colour?type=Car
```

```
200 OK
...
{
  "value": "red",
  "type": "Text",
  "metadata": {}
}
```

shorter URL in NGSIv2

structural overhead

redundant (already part of the request URL)

mostly useless

Example: get attribute *colour* at Car1 entity (type Car)

URL & payload simplification

- Moreover, NGSIv2 provides simplified data representations
  - *keyValues*: exclude attribute type and metadata
  - *values*: only attribute values (*attrs* is used to order the values in the resulting vector)
  - *unique:* like *values* which in addition removes duplicate values

```
GET /v2/entities/Car1/attrs?options=keyValues
```

```
200 OK
...
{
  "model": "Ford",
  "colour": "red",
  "temp": 22
}
```

```
GET /v2/entities/Car1/attrs?options=values&attrs=model,colour,temp
```

```
200 OK
...
["Ford", "red", 22]
```

Example: get attribute *colour* at Car1 entity (type Car)

- Not only for retrieval operations, also for creation/update operations
  - Default attribute types are used in that case

Native JSON datatypes

- In NGSIv1
  - All attribute values are string based to align with XML encoding
    - At the end, XML support was removed (in Orion 1.0.0), but it left an awful legacy ☹
  - Although creation/update operations can use numbers, bool, etc. at the end they are transformed to strings and stored in that way internally
  - Retrieve operation always provides string encoded values (*)
- NGSIv2 fully supports all the types described in the JSON specification (string, number, boolean, object, array and null) (**)

(*) Exception: entities created/updated with NGSIv2 (which support native types) and retrieved with NGSIv1 will render without stringification.
(**) Support for metadata vector and object values was added in Orion 1.3.0 (i.e. not included in IoTP v4.0)

Native JSON datatypes

Example: create Car1 entity (type Car) with attribute *speed* set to 98

NGSIv1

```
POST /v1/contextEntities
...
{
  "id": "Car1",
  "type": "Car",
  "attributes": [
  {
    "name": "speed",
    "type": "Number",
    "value": 98
  }
  ]
}
```

created as number but retrieved as string… weird!

```
GET /v1/contextEntities/Car1/attributes/speed
...
{
   "attributes": [
      {
         "name": "speed",
         "type": "Number",
         "value": "98"
      }
   ],
   "statusCode": { … }
}
```

NGSIv2

```
POST /v2/entities?options=keyValues
...
{
  "id": "Car1",
  "type": "Car",
  "speed": 98,
  "isActive": true
}
```

```
GET /v2/entities/Car1/attrs?options=keyValues
...
{
 "speed": 98,
 "isActive": true
}
```

coherent result

Text-based attribute value set/get

- In NGSIv1
  - There is no similar functionality
- NGSIv2 offers set/get attribute access directly without anything else than the attribute value itself in the request/response payload
  - In the set operation, attribute type and metadata are kept as they are

Example: set *speed* attribute value at Car1 entity

```
PUT /v2/entities/Car1/attrs/speed/value
Content-Type: text/plain
…

86
```

```
200 OK
…
```

Example: get *speed* attribute value at Car1 entity

```
GET /v2/entities/Car1/attrs/speed/value
```

```
200 OK
Content-Type: text/plain
…

86
```

Geolocation capabilities

- In NGSIv1
  - Entity location must be a point
  - Queries are based on an area specification (circle or polygon, inner or outer area)
  - Query as part of queryContext payload scope
- In NGSIv2
  - In addition to point, entity location can be a line, box, polygon or arbitrary GeoJSON
  - Queries are based on a spatial relationship and a geometry
    - Spatial relationships: near (max and min distance), coveredBy, intersect, equal and disjoint
    - Geometries: point, line, box, polygon
  - Query as part of URL (more user-friendly than payload-based approach)

Geolocation capabilities

Example: retrieve all entities of type "City" (no matter the id) whose distance to Madrid city center (GPS coordinates 40.418889,-3691944) is less than 13.5 km

NGSIv1

```
POST /v1/queryContext
…
{
 "entities": [
 {
  "type": "City",
  "isPattern": "true",
  "id": ".*"
 }
 ],
 "restriction": {
  "scopes": [
  {
   "type" : "FIWARE::Location",
   "value" : {
    "circle": {
     "centerLatitude": "40.418889",
     "centerLongitude": "-3.691944",
     "radius": "13500"
    }
   }
  }
  ]
 }
}
```

NGSIv2

```
GET /v2/entities?
  idPattern=.*&
  type=City&
  georel=near;maxDistance:13500&
  geometry=point&
  coords=40.418889,-3.691944
```

Much easier and more compact in NGSIv2

Geolocation capabilities

```
POST /v2/entities
{
  "id": "E",
  "type": "T",
  "location": {
    "type": "geo:point",
    "value": "40.41,-3.69"
  }
}
```

Point location
(the only one supported by NGSIv1)

```
POST /v2/entities
{
  "id": "E",
  "type": "T",
  "location": {
    "type": "geo:line",
    "value": [ "2, 2", "8, 7" ]
  }
}
```

Line location (e.g. a street)

```
POST /v2/entities
{
  "id": "E",
  "type": "T",
  "location": {
    "type": "geo:box",
    "value": [ "2, 2", "8, 7" ]
  }
}
```

Box location (e.g. a squared building)

```
POST /v2/entities
{
  "id": "E",
  "type": "T",
  "location": {
    "type": "geo:polygon",
    "value": [ "2, 2", "8, 7", "-1, 4", "2, 2"]
  }
}
```

Polygon location (e.g. a city district)

```
POST /v2/entities
{
  "id": "E",
  "type": "T",
  "location": {
    "type": "geo:json",
    "value": {
      "type": "Polygon",
      "coordinates": [ [ [2, 1], [4, 3], [5, -1], [2, 1] ] ]
  }   } }
```

GeoJSON geometry (full flexibility)

Filtering

- In NGSIv1
  - Limited filtering functionality, much of it based on queryContext complex scopes
  - Filters are not supported in subscriptions

```
POST /v1/queryContext
…
{
"restriction": {
   "scopes": [
   {
     "type" : "FIWARE::StringFilter",
     "value" : "temp<24"
…
}
```

Example: filtering entities which *temperature* is less than 24

This is the only interesting part, all the rest is structural overhead

- In NGSIv2 the mechanism
  - Is simpler (see next slide)
  - Can be applied to both queries and subscriptions (described in a later topic of this presentation)

Filtering

- For the **GET /v2/entities** operation, retrieve all entities…
  - … of a given **entity type**

```
GET /v2/entities?type=Room
```

  - … whose **entity id** is in a **list**

```
GET /v2/entities?id=Room1,Room2
```

  - .. whose **entity id** match a regex **pattern**
    - Example: the id starts with "Room" followed by a digit from 2 to 5

```
GET /v2/entities?idPattern=^Room[2-5]
```

  - … with an **attribute** that matches a given **expression**
    - Example: attribute temp is greater than 25

```
GET /v2/entities?q=temp>25
```

supported operators:

- *== (or :), equal*
- *!=, unequal*
- *>, greater than*
- *<, less than*
- *>=, greater than or equal*
- *<=, less than or equal*
- *A..B, range*
- *^=, pattern (regex)*
- *Existence/inexistence*

- Filters can be used simultaneously (i.e. like **AND** condition)

Datetime support

- In NGSIv1
  - There is no support for attributes meaning dates, they are treated as conventional strings
- NGSIv2 implements date support
  - Based on ISO ISO8601 format
  - Use reserved attribute type **DateTime** to express a date

```
POST /v2/entities
…
{
  "id": "John",
  "birthDate": {
    "type": "DateTime",
    "value": "1979-10-14T07:21:24.238Z"
  }
}
```

Example: create entity John, with *birthDate* attribute using type DateTime

  - Attribute value arithmetic filters can be used with dates as if they were numbers

```
GET /v2/entities?q=birthDate<1985-01-01T00:00:00
```

  - Entity **dateModified** and **dateCreated** special attributes, to get entity creation and last modification timestamps
    - They are shown in query responses using **options=dateModified,dateCreated** (*)

(*) This is probably going to change in a next version of the NGSIv2 API to use **attributes=dateModified,dateCreated** instead. However, Orion will keep backward compatibility with the way described here

Subscription improvements

- NGSIv1 context subscription API is very limited
  - There is no operation to list existing subscriptions
    - If a client loses the ID of created subscriptions, there is no way to retrieve them through the API
  - No support for permanent subscriptions
    - Creating absurdly long subscriptions (e.g. 100 years) is a dirty workaround
  - Fix notification structure
    - Difficult to integrate to arbitrary endpoints (e.g. public REST services)
  - No support for filters
- NGSIv2 subscription API solves all these limitations and introduces some additional enhancements
  - Notification attributes based on “blacklist” (in addition to the “whitelist” approach in NGSIv1)
  - Ability to pause/resume subscriptions
  - Extra fields: times sent, last notification and description

Anatomy of a NGSIv2 subscription

NGSIv1

```
POST /v1/subscribeContext
…
{
   "entities": [
      {
         "type": "Room",
         "isPattern": "false",
         "id": "Room1"
      }
   ],
   "attributes": [ "temp“ ],
   "reference": "http://<host>:<port>/publish",
   "duration": "P1M",
   "notifyConditions": [
      {
         "type": "ONCHANGE",
         "condValues": [ "temp" ]
      }
   ],
   "throttling": "PT5S"
}
```

Redundant

Simpler way of specifying expiration and throttling in NGSIv2

```
200 OK
…
{  "subscribeResponse": {
      "duration": "P1M",
      "subscriptionId": "51c0ac9ed714fb3b37d7d5a8",
      "throttling": "PT5S"
}}
```

NGSIv2

```
POST /v2/subscriptions
…
{
  "subject": {
    "entities": [
      {
        "id": "Room1",
        "type": "Room"
      }
    ],
  },
  "condition": {
    "attrs": [ "temp"  ]
  }
  },
  "notification": {
    "http": {
      "url": "http://<host>:<port>/publish"
    },
    "attrs": [  "temp"   ]
  },
  "expires": "2026-04-05T14:00:00.00Z“
  "throttling": 5
}
```

Example: subscribe to Room1 entity, so whenever a change occurs in the *temp* attribute a notification including only temp is sent

Simpler response (no payload)

```
201 Created
Location: /v2/subscriptions/51c0ac9ed714fb3b37d7d5a8
…
```

List subscriptions and special fields in NGSIv2

- List operations (not available in NGSIv1)
  - **GET /v2/subscriptions** lists all subscriptions
  - **GET /v2/subscriptions/<id>** retrieves information of a particular subscription
- Whitelist vs. blacklist (in the **notification** field)
  - Use **"attrs": ["A", "B"]** to "include A and B in the notification" (whitelist)
  - Use **"exceptAttrs": ["A", "B"]** to "include all the attributes except A and B" (blacklist)
  - Use **"attrs": [ ]** to include "all the attributes" (special case)
- Other informative fields (in the **notification** field)
  - **timesSent**: the number of times that the subscription has been triggered and a notification has been sent
  - **lastNotification**: datetime corresponding to the last notification
- Other informative fields (at root level)
  - description, free text descriptive text for user convenience

Permanent and paused subscriptions in NGSIv2

- The **status** attribute can be used to pause/resume subscriptions

To pause

```
PATCH /v2/subscriptions/<id>
…
{
  "status": "inactive"
}
```

To resume

```
PATCH /v2/subscriptions/<id>
…
{
  "status": "active"
}
```

- In GET operations, the **status** field can be
  - active: subscription is active (notifications will be sent)
  - inactive: subscription is inactive (notifications will not be sent)
  - expired: subscription is expired (notifications will not be sent)

Notification formats in NGSIv2

- The optional **attrsFormat** field can be used to choose between different notification flavors, aligned with the representation modes

normalized (default)

```
{
  "subscriptionId": "12345",
  "data": [
    {
      "id": "Room1",
      "type": "Room",
      "temperature": {
        "value": 23,
        "type": "Number",
        "metadata": {}
      }
    }
  ]
 }
```

keyValues

```
{
  "subscriptionId": "12345",
  "data": [
    {
      "id": "Room1",
      "type": "Room",
      "temperature": 23
    }
  ]
 }
```

values

```
{
  "subscriptionId": "12345",
  "data": [ [ 23 ] ]
}
```

Outer vector represent the list of entities, inner vector the values of the attribute of each entity (not too interesting in this single-entity single-attribute example)

- Notifications include the **NGSIv2-AttrsFormat** header to help the receiver identify the format
- **legacy** can be used as value for **attrsFormat** in order to send notifications in NGSIv1 format
  - Very useful when integrating legacy notification endpoints

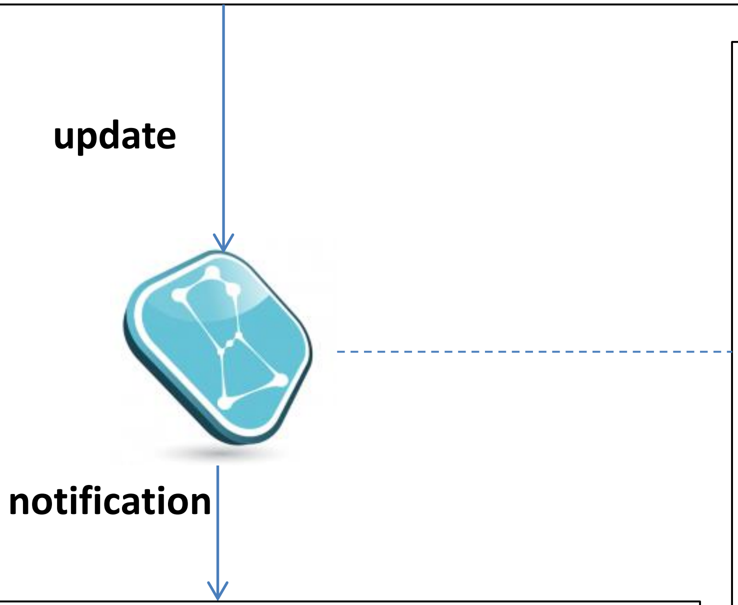
Custom notifications in NGSIv2

- Apart from the standard formats defined in the previous slide, NGSIv2 allows to re-define **all** the notification aspects
- **httpCustom** is used (instead of **http**) with the following subfields
  - URL query parameters
  - HTTP method
  - HTTP headers
  - Payload (not necessarily JSON!)
- A simple macro substitution language based on **${..}** syntax can be used to “fill the gaps” with entity data (id, type or attribute values)

Custom notifications in NGSIv2

Example: send a text notification (i.e. not JSON) with temperature value, using the entity id and type as part of the URL

```
PUT /v2/entities/DC_S1-D41/attrs/temp/value?type=Room
…
23.4
```



```
…
"httpCustom": {
 "url": "http://foo.com/entity/${id}",
 "headers": {
  "Content-Type": "text/plain"
  },
 "method": "PUT",
 "qs": {
  "type": "${type}"
 },
 "payload": "The temperature is ${temp} degrees"
 }
…
```

Custom notification configuration

```
PUT http://foo.com/entity/DC_S1-D41?type=Room
Content-Type: text/plain
Content-Length: 31

The temperature is 23.4 degrees
```

Subscription filters in NGSIv2

Example: subscribe to *speed* changes in entities with id Truck11 or Car2 to Car5 (both case of type RoadVehicle) whenever *speed* is greater than 90 and the vehicle distance to Madrid city center is less than 100 km

- Filters (described in previous slides) can be also used in subscriptions
  - id
  - type
  - id pattern
  - attribute values
  - geographical location (*)

```
POST /v2/subscriptions
…
{
 "subject": {
  "entities": [
   {
    "id": "Truck11",
    "type": "RoadVehicle"
   },
   {
    "idPattern": "^Car[2-5]",
    "type": "RoadVehicle"
   }
  ],
  "condition": {
   "attrs": [ "speed"  ],
   "expression":  {
     "q": "speed>90",
     "georel": "near;maxDistance:100000",
     "geometry": "point",
     "coords": "40.418889,-3.691944"
   }
  }
 },
 …
}
```

(*) Support for geo-filters in subscriptions was added in Orion 1.3.0 (i.e. not included in IoTP v4.0)

Batch operations

- In NGSIv1 we have standard operations
  - POST /v1/updateContext
  - POST /v1/queryContext
- Similar but more user-friendly operations have been included in NGSIv2
  - POST /v2/op/update
  - POST /v2/op/query

Batch operations

Example: create Room1 entity (type Room) with attribute *temp* set to 29

NGSIv1

```
POST /v1/updateContext
…
{
 "updateAction": "APPEND“,
 "contextElements": [
    {
       "type": "Room",
       "isPattern": "false",
       "id": "Room1",
       "attributes": [
       {
         "name": "temp",
         "type": "float",
         "value": "29"
       }
       ]
    }
 ]
}
```

structural overhead

```
200 OK
…
{
 "contextResponses" : [ … ],
 "statusCode" : {
   "code" : "200",
   "details" : "OK"
 }
}
```

lots of useless stuff here

NGSIv2

```
POST /v2/op/update
{
 "actionType": "APPEND",
  "entities": [
   {
     "type": "Room",
     "id": "Room1",
     "temperature":
      {
        "type": "Number",
        "value": 29
      }
   }
  ]
}
```

```
201 Created
```

NGSIv2 response doesn’t have any payload at all

Batch operations

Example: get *temp* attribute of all entities with type Room

NGSIv1

```
POST /v1/queryContext
…
{
   "entities": [
      {
         "type": "Room",
         "isPattern": "true",
         "id": ".*"
      } ,
      "attributes": [ "temp"  ]
   ]
}
```

```
200 OK
…
{
   "contextResponses": [
      {
         "contextElement": {
            "attributes": [
               {
                  "name": "temp",
                  "type": "Number",
                  "value": "25"
               }
            ],
            "id": "Room1",
            "isPattern": "false",
            "type": "Room"
         },
         "statusCode": { … }}
   ]
}
```

NGSIv2

```
POST /v2/op/query
…
{
  "entities": [
    {
      "idPattern": ".*",
      "type": "T"
    }
  ],
  "attributes": [ "temp" ]
}
```

```
200 OK
…
[
  {
    "id": "Room1",
    "type": "Room",
    "temp": {
      "type": "Number",
      "value": 25
    }
  }
]
```

Requests are more or less the same, but the simplicity of NGSIv2 becomes evident when comparing responses

Pagination

- In NGSIv1
  - based on **limit**, **offset** and **details**
  - Dirty workaround to fit count into NGSIv1 payloads, using an errorCode for something that actually is not an error and forcing to text based processing of the details field
  - Fixed order: always by creation date
- In NGSIv2
  - based on **limit**, **offset** and **options=count**
    - This part doesn't change too much
  - Cleaner and easier way of returning count, with the **Fiware-Total-Count** HTTP header in the response
  - Configurable ordering based on **orderBy** parameter
    - See details in the NGSIv2 specification

```
"errorCode": {
    "code": "200",
    "details": "Count: 322",
    "reasonPhrase": "OK"
 }
```

```
Fiware-Total-Count: 322
```

Working with IDs

- In NGSIv1
  - Fields such as entity id, attribute name, etc. may have any value (*)
  - This could cause a lot of problems as these fields use to act as IDs in many places when propagated through notifications
    - E.g. Cygnus MySQL sink may have problems when these fields are mapped to tables names, whose allowed charset is very strict
  - In addition, NGSIv1 allows ids or attribute names as "" (empty string) which is weird and typically an error condition in the client
- NGSIv2 establishes a set of restrictions to ensure sanity in the usage of ID fields. In particular:
  - Allowed characters are those in the plain ASCII set, except the following ones: control characters, whitespace, &, ?, / and #.
  - Maximum field length is 256 characters.
  - Minimum field length is 1 character.
  - The rules above apply to the following six fields (identified as ID fields): entity id, entity type, attribute name, attribute type, metadata name, metadata type

(*) Excluding the forbidden characters described in the Orion manual, which are general for all the fields in both NGSIv1 and NGSIv2 APIs

Next things to come

- Up to now, all the slides have described the stable version of the API corresponding to Release Candidate 2016.05 (http://fiware.github.io/specifications/ngsiv2/stable)
- There is also a *work in progress* (WIP) version, describing future functionally (http://fiware.github.io/specifications/ngsiv2/latest)
- Warning: the future functionality of the WIP version is subject to change and it may change before to be consolidated in a next Release Candidate stable version

New filtering features

- As an extension of the filters already described in slide 20
  - By **entity type pattern** (regex)

```
GET /v2/entities?typePattern=T[ABC]
```

attribute name

attribute sub-key (for compound attribute values only)

  - By **attribute value sub-key** (q)

```
GET /v2/entities?q=tirePressure.frontRight >130
```

  - By **metadata value** (mq)

attribute name

metadata name

```
GET /v2/entities?mq=temperature.avg>25
```

metadata sub-key (for compound metadata values only)

  - By **metadata value** sub-key (mq)

```
GET /v2/entities?mq=tirePressure.accuracy.frontRight >90
```

New filtering features

Example: subscribe to *speed* changes in any entities of any type ending with Vehicle (such as RoadVehicle, AirVehicle, etc.) whenever *speed* is greater than 90 its *average* metadata is between 80 and 90 and the vehicle distance to Madrid city center is less than 100 km

- They can be used also in subscriptions
  - type pattern
  - metadata value

```
POST /v2/subscriptions
…
{
 "subject": {
  "entities": [
   {
     "idPattern": ".*",
     "typePattern": ".*Vehicle"
   },
  ],
  "condition": {
   "attrs": [ "speed"  ],
   "expression":  {
     "q": "speed>90",
     "mq": "speed.average==80..100",
     "georel": "near;maxDistance:100000",
     "geometry": "point",
     "coords": "40.418889,-3.691944"
   }
   }
  },
  …
}
```

Metadata filtering and special metadata

- By default all attribute metadata are included in notifications
- The **metadata** field (in the **notification** field) can be used to specify a filtering list
- The **metadata** field can also be used to explicitly include some special metadata (not included by default)
  - **actionType**: whose value is the action type corresponding to the update triggering the notification: "update", "append" or "delete" (*)
  - **previousValue**: which provides the value of the attribute previous to processing the update that triggers the notification
- The "*" can be used as an alias of "all the normal metadata"
- Examples
  - **"metadata": ["MD1", "MD2"]** to "include only metadata M1 and M2"
  - **"metadata": ["previousValue"]** to "include previousValue and not any other attribute"
  - **"metadata": ["actionType", "*"] to** "include actionValue and all the other (regular) metadata"
  - **"metadata": ["*"]** to "include all metadata" (same effect than not using metadata, not very interesting)

(*) actionType "delete" not yet supported by Orion in 1.4.0.

Batch query scope

Example: get all entities of type T with the attribute temp as long as that attribute is greater than 40 and the entity distance to coordinates (40.31, -3.75) is less than 20 km

- This is the way of including q, mq and geo filters (typically used as URL param of a GET operation) in a batch query
- Like NGSIv1 scopes but much simpler (without the restriction structural overhead)

```
POST /v2/op/query
…
{
  "entities": [
    {
      "idPattern": ".*",
      "type": "T"
    }
  ],
  "attributes": [ "temp" ],
  "scopes": [
    {
      "type": "FIWARE::StringQuery",
      "value": "temp"
    },
    {
       "type" : "FIWARE::Location::NGSIv2",
       "value" : {
         "georel": [ "near", "maxDistance:20000" ],
         "geometry": "point",
         "coords": [ [40.31,-3.75] ]
       }
     }
   ]
}
```

Useful references

- Introduction to NGSI and Orion
  - http://bit.ly/fiware-orion
- Orion Manual
  - https://fiware-orion.readthedocs.io
- Orion page at FIWARE Catalogue
  - http://catalogue.fiware.org/enablers/publishsubscribe-context-broker-orion-context-broker
- NGSIv2 specs
  - http://fiware.github.io/specifications/ngsiv2/stable
  - http://fiware.github.io/specifications/ngsiv2/latest
- Orion support at StackOverflow
  - Look for existing questions at http://stackoverflow.com/questions/tagged/fiware-orion
  - Ask your questions using the "fiware-orion" tag
- FIWARE Tour Guide Application
  - https://github.com/fiware/tutorials.TourGuide-App

Thanks!

FIWARE

OPEN APIs FOR OPEN MINDS

FIWARE Lab

Spark your imagination

FIWARE Ops

Easing your operations

www.fiware.org

@Fiware