ONTOLOGY GOVERNANCE

June 2016

Version 4

indra

1 INDEX

2 INTRODUCTION

The Sofia2 platforms offers all the mechanisms needed to interconnect any kind of sensor networks, offering several mechanisms to do so.

The current document, attempting to establish a common foundation for the work in all the applications and verticals, establishes a number of **suggestions** about rules and procedures to use these.

The objective is that the developers and information-exploiting system know **a priori** the data structures and types handled by the platform and that they can **homogenously use these, independently of the particular and concrete implementation of any integration development.**

We strongly recommend reading the reference documentation beforehand to understand all the concepts used in the current document.

The platform gets, registers and manages all the sensory information (measurements, commands, alerts, events, etc…) using **json ontologies** to represent the data.

As the mechanisms that Sofia2 offers to manage ontologies are very reliant and adaptable, the governance rules explained in the current documents were established to **ensure that all the information in the Platform has a homogenous structure among the different applications and verticals**, and that it **allows for a horizontal use of all the generated information.**

3 SOFIA2 ENTITIES

The Sofia2 Platform manages two different types of entities:

- **Assets:** An asset is a physical or virtual element that can generate or consume sensory information and manage it through the Sofia2 Platform.
- **Ontologies:** The assets generate information and this information is modelled in the Platform using (JSON) ontologies.

3.1 Assets

Depending on their nature and on how the assets are connected to the Platform, they can be classified as:

- **Inventory Assets:** These assets are directly managed by the Platform. Due to this, the Platform must know a priori all the information about a given asset that may allow to connect directly to it. **All the inventory assets must be registered in the Platform's Asset Inventory**. These assets can be sub-divided in:
    - **Managed:** These are the assets that are connected to the Platform, that is to say, that are managed by the KP's. Their main characteristic is that the KP's managing them need to know all the information to physically access them.
    - **Unmanaged:** These are known assets in the Platform's inventory, that generate and consume information from it; but the interconnectivity module is not responsible of their final management. That is to say, these assets are managed by other information systems that integrate in the Platform.
- **Virtual Assets:** Virtual assets are not known a priori. They generate information and this information reaches the platform using integrations, commonly against other cloud-based information services such as, for instance, social networks.

3.2 Ontologies

The connected assets generate and consume sensory information in the form of json ontologies.

When creating an ontology, you can use the following catalogues (templates):

- **Feed** (measurements): This is the information sent by the connected assets. This can be instantaneous measurements, aggregations by time periods, calculations, etc. A feed can gather several measurements simultaneously and **all the feeds must be geo-referenced** (points, lines or areas) albeit these can be mobile references.

- **Command:** These are **commands sent to the Assets** that have actuation capacities. These commands are asynchronous (There is no immediate answer); due to this there are two different types of command:
    - **Request Commands:** Instructions sent to an Asset.
    - **Response Commands:** The responses or ACK's to those commands sent by the Asset. The responses to the commands solely confirm that the instruction was received.
- **Alert:** The alerts are messages that notify situations (alerts, incidents, messages, notifications, etc.) and are unique because its state and criticity level changes with time, from the moment an initial messages generates the alert, until the alert is closed. The alerts are not known a priori, that is to say, they are generated in the moment they take place.
- **Schedule** (Events): These events reflect situations in time and space that are known, that is to say, they happen through a time span in a specific place and usually they are known a priori.
- **Audit** (Log): Audit messages or log are internal to the Platform. They are used to make explicit and manage specific concrete situations of an Asset. For instance, switching on a lamppost or registering a sensor's malfunctioning.
- **KPI** (Indicator): Indicators are a special type of measurement. Their main feature is that they are not generated by a single Assets. These are calculations made on multiple data, historical series and even special data layers that are stored as measurements in the Platform.

Special Rules to Consider:

- Every time a new set of Assets of the same type is connected to the Platform, at least the ontologies **feed** and **audit** must be registered for it; if needed, its **command** ontology must be registered too.
- Immediately after receiving a command and executing it correctly, the Asset (the KP managing it) must generate a new measurement to reflect its new state in the Platform.

4 NOMENCLATURE

We will next establish the basic nomenclature rules.

- The names will be in English language and follow the Java standard (aka “camel”).
- To define templates:
    - Short, self-explanatory names. First letter in capital. First letters identify the type of ontology:
        - Feed: **Feed**
        - Command: **Cmd**
        - Alert: **Alrt**
        - Schedule: **Schdl**
        - Audit: **Adt**
        - KPI: **Kpi**
- To define **ontologies:**
    - First letter in lower case. Starting always with the type of ontology. Example for a Spira: feedSpira, cmdSpira; adtSpira ...
- To define an ontology's **attributes:**
    - First letter in lower case. No spaces. No special characters.
- To define **constants** used as possible values for the ontologies’ attributes: All the letters in capitals: Example: MOBILE, FIXED, VIRTUAL.

5 TYPING AND FORMATS

To define ontologies we will use UTF-8 text strings following the json schema established by the corresponding template (currently following JSON SChema 0.4 **http://json-schema.org/draft-04/schema#**).

We will next establish the typing and format rules for the different supported types:

- **UUIDs:**
  - Text string. Standard Universally Unique Identifier.
- **Integer numbers:**
  - 64-bit Long Integers.
  - Example: {'counter' : 10}
- **Floating numbers:**
  - Simple notation. Decimal with point. 64 bits.
  - Example: {'value' : 10.5}
- **URLs and URIs:**
  - Text string. Codified following RFC-1738 standard.
  - Example: {'url' : 'http%3A%2F%2Fwww.coruna.es%2Fmedioambiente%2F'}
- **Timestamps:**
  - Date. Text string following format ISO-8601. RFC 3339.
  - Attribute “$date” in object.
  - Example: {"timestamp":{"$date":"2014-01-27T11:14:00Z"}}
- **Dates and ranges of dates:**
  - Text string following format ISO-8601. RFC 3339.
  - Attribute “$date” in object.
  - Date example: {"created":{"$date":"2014-01-27T11:14:00Z"}}
  - Range of date example: {“period”:{“$date” : “2010-07-02T11:44:09Z/2010-07-02T11:47:00Z”}}
- **Addresses:**
  - Simplified notation to ease integration tasks:

```
{"address": {
      "location": "text string",
      "number": "text string"
      }
}
```

- **Measurement units**
    - String text following the notation **JScience library** (**http://jscience.org/api/javax/measure/unit/SI.html)(http://jscience.org/api/javax/measure/unit/NonSI.html**)
    - Example: {'unit': 'A'} # Amperes
- **Geographical coordinates**
    - Follow the **OGC GeoJson** definition. Coordinate scheme **WGS84**. There is no Z coordinate. Order [longitude, latitude].
    - **Points:**
        - GeoJson Point
        - Example:

```
{"geometry": {
            "type": "Point",
            "coordinates": [-8.410161625142807, 43.360463863501934]
      }
}
```

- **Lines:**
    - GeoJson LineString
    - Example:

```
{"geometry": {
            "type": "LineString",
            "coordinates":[
                  [-8.410161625142807, 43.360463863501934],
                  [-8.410161625142807, 43.360463863501978]
                  ]
      }
}
```

- **Areas:**
    - GeoJson Polygon
    - Example:

```
{"geometry": {
                    "type": "Polygon",
                    "coordinates":[
                                [[-8.410161625142807, 43.360463863501934],
                                [-8.410161625142807, 43.360463863501978],
                                [-8.41016162514290, 43.360463863501978],
                                [-8.410161625142807, 43.360463863501934]]
                                ]
          }
}
```

[NOTE]: To close the polygon, the first and last value of each ring must be identical.

[NOTE]: A polygon can have two rings (the outer one and the inner one).

6 PRE-DEFINED TEMPLATES

6.1 Feeds (measurements)

To define the measurement template we use a simplification of the AMON data standard (**http://amee.github.io/AMON/**):

```
{"Feed":
      {
      "asset":
            {
             "assetId" : string, (required)
             "assetType": string, (required)
             "assetSource": string, (required)
             "assetName": string (optional)
            },
      "type": string, (required) [FIXED, MOBILE, VIRTUAL]
      "timestamp": (required)
            {
                  "$date": "RFC 3339 DATETIME"
            },
      "attribs": (optional)
            [
              { "name": "value" }
            ],
      "geometry": geojson [Point, LineString, Polygon], (optional)
      "measures": (required)
            {
              "timestamp" : (required)
                        {
                          "$date": "RFC 3339 DATETIME"
                        },
              "type" : string, (required) [INSTANT, CUMULATIVE, PULSE]
              "period" : number, (optional)
              "periodUnit": string, (optional) [m, s, h, d]
              "values" : (required)
                        [
                          {
                              "name": string, (optional)
                              "desc": string, (optional)
                              "unit": string, (required)
                              "measure": string, (required)
                              "method": string,(required)
                              "modifications": (optional)
                                    [ {
                                      "oldMeasure": string, (required)
                                      "changeTimestamp": (required)
                                          {
                                           "$date": "RFC 3339 DATETIME"
                                          }
                                      "changeDesc": string, (optional)
                                      }
                                    ]
                          }
                        ]
            }
      }
}
```

The **asset** object references the asset sending the measurement:

- **assetId:** Identifier of the asset in the reference system managing it (which is established in the field assetSource).
- **assetType:** Type of asset (Lamp post, humidity sensor, etc...)
- **assetSource:** Information system managing the asset.
- **assetName:** Optional attribute to associate a name to the asset, if deemed necessary.

The sensor type (**type**) references its nature. The valid types are:

- **FIXED:** Sensors that are known a priori (managed by a known inventory), geographically located in a known fixed position.
- **MOBILE:** Sensors that are known a priori, and that move. Its location is updated with every measurement.
- **VIRTUAL:** Sensors that are not known a priori (e.g. social networks).

The **timestamp** references the **feed capture's date and time.**

> **NOTE1:** Not to be confused with the timestamp that is automatically generated when the ontology is sent to the interconnection module.
>
> **NOTE2:** Not to be confused with the timestamp of the measurement that references the moment when the measurement is taken.

The **attributes** object (attribs) intends to gather an arbitrary list of attributes that are modeled as key:value. Its utility can go from getting secondary keys to storing any needed additional attribute.

The **geometry** object gets the position (point, line or polygon) that the feed references. **In any case, if the feed's position is known, it must appear in the feed** independently on whether the position is registered in the inventory or not. If the position is not known, the feed should not include the attribute.

The **measures** object references all the references features that are common to all the captured measurements, and lists all the measurements that have been made:

- **timestamp:** Reference date when the measurement was made.
- **type:** Type of measurement: Instantaneous, accumulated, pulse.
- **period:** If applies, time period used to calculate the measurements.
- **periodUnit:** Unit of time ('s', 'm', 'h', 'd') used to define the time period.

- **values:** List of performed measurments.
    - **name:** If applies, name that is representative of the measurement.
    - **desc:** If applies, a description of the performed measurement.
    - **unit:** Measurement unit
    - **measure:** Value of the measurement in its most updated version. That is to say, the measure attribute will always have the valid measurement. If the measurement has been modified, the historical values will be stored in the list of the modifications attribute.
    - **method:** Method used to get the measurement (mean, min, max, etc...).
    - **modifications:** List of modifications made on the originally-captured measurement.

6.2 Commands

To define commands, this is used:

```
{"Command":
      {
      "commandId": string, (required)
      "asset":
         {
            "assetId" : string, (required)
            "assetType": string, (required)
            "assetSource": string, (required)
            "assetName": string (optional)
         },
      "timestamp": (required)
            {
              "$date": "ISO 3339 DATETIME"
            },
      "desc": string, (optional),
      "type": string, (required) [REQUEST, RESPONSE],
      "command":
            {
            "type": string, (required) [SWITCH, DIM, SET, EXECUTE, SEND]
            "value1": string, (optional)
            "value2": string, (optional)
            "value3": string, (optional)
            "msg": string (optional)
            }
      "rule":
            {
            "type": string, (required) [ASAP, DATE]
            "date": (optional)
                  {
                    "$date": "ISO 3339 DATETIME"
                  }
            }
      }
}
```

6.3 Alerts

To define alerts, we use a simplification of the **CAP 1.2** standard (**http://docs.oasis-open.org/emergency/cap/v1.2/CAP-v1.2-os.html**)

```
{"Alert":
      {
      "id" : {
             "alertId": string, (required)
             "alertSource": string (required)
             },
      "timestamp":{
                  "$date": "ISO 3339", (required)
                  },
      "asset":
            {
             "assetId" : string, (required)
             "assetType": string, (required)
             "assetSource": string, (required)
             "assetName": string (optional)
            },
      "alert":
            {
            "sourceAlertId": identifier,
            "subject": string required,
            "description": string optional,
            "source": string required,
            "type": [ALARM, WARNING, MESSAGE, NOTIFICATION, INFO],
            "status": [OPEN, CLOSED, UNKNOWN],
            "affectedLocations": (optional)
                  [
                    {
                    "desc": string, (optional)
                    "geometry": geojson,(optional) [Point, Line, Polygon],
                    "locationUri": string, (optional)
                    }
                  ]
            }
      "info":
            {
            "action": [CREATE, CLOSE, UPDATE, ACK,
                  FOLLOW, SCALATION, REMINDER, CANCEL],
            "sender": string, (required)
            "contact": string, (optional)
            "description": string, (optional)
            "parameters": string, (optional)
            "urgency": [EXPECTED, FUTURE, INMEDIATE, PAST, UNKNOWN],
            "severity": [EXTREME, MINOR, MODERATE, SEVERE, UNKNOWN],
            "certainty": [LIKELY, OBSERVED, POSSIBLE, UNLIKELY, UNKNOWN],
            "resources": optional
                  [
                   {
                    "name": string, (required)
                    "description": string, (optional)
                    "uri": string, (required)
                    "mimeType": string (optional)
                   }
                  ]
            }
      }
}
```

6.4 Events

The definition of events follows this schema:

```
{"Event":
      {
      "id":
            {
             "eventId": string, (required)
             "eventSource": string (required)
            },
      "timestamp":{
                  "$date": "ISO 3339", (required)
                },
      "asset":
            {
             "assetId" : string, (required)
             "assetType": string, (required)
             "assetSource": string, (required)
             "assetName": string (optional)
            },
      "eventInfo":
            {
            "subject": string, (required)
            "description": string, (optional)
            "type": string,[INFO, PROGRAM, EVENT]
            "affectedLocations": (optional)
                  [
                        {
                        "desc": string, (required)
                        "geometry": geojson optional [Point, Line, Polygon],
                        "locationURI": string optional
                        }
                  ],
            "resources": (optional)
                  [
                        {
                        "name": string, (required)
                        "description": string, (optional)
                        "uri": string, (required)
                        "mimeType": string (optional)
                        }
                  ]
            }
      "eventRule":
            {
            "type": [SINGLE, PERIOD, RULE],
            "period": (required)
                  {
                   $date: "RFC 3339 INTERVAL"
                  },
            "repeatEach": entero, (opcional)
            "repeatUnit": string (opcional) [s, m, h, d, w, m]
}
}
```

6.5 Audit

The definition of audit messages follows this schema:

```
{"Adt":
      {
      "id" : {
            "auditId": string, (required)
            "auditSource": string (required)
            },
      "timestamp": (required)
                  {
                   "$date": "ISO 3339 DATETIME"
                  },
      "asset":
            {
            "assetId" : string, (required)
            "assetType": string, (required)
            "assetProvider": string, (required)
            "assetName": string (optional)
            },
      "message": {
            "source": string required,
            "sender": string optional,
            "subject": string required,
            "body": string optional,
            "level": [INFO, WARNING, ERROR, DEBUG]
      }
```