ONTOLOGY DEFINITION IN SOFIA2

JULY 2014

VERSIÓN 4

indra

1 INDEX

2 INTRODUCTION

2.1 Requirements and document´s goals and scope

This document explains how to define ontologies on the Platform in a clear and simple manner.

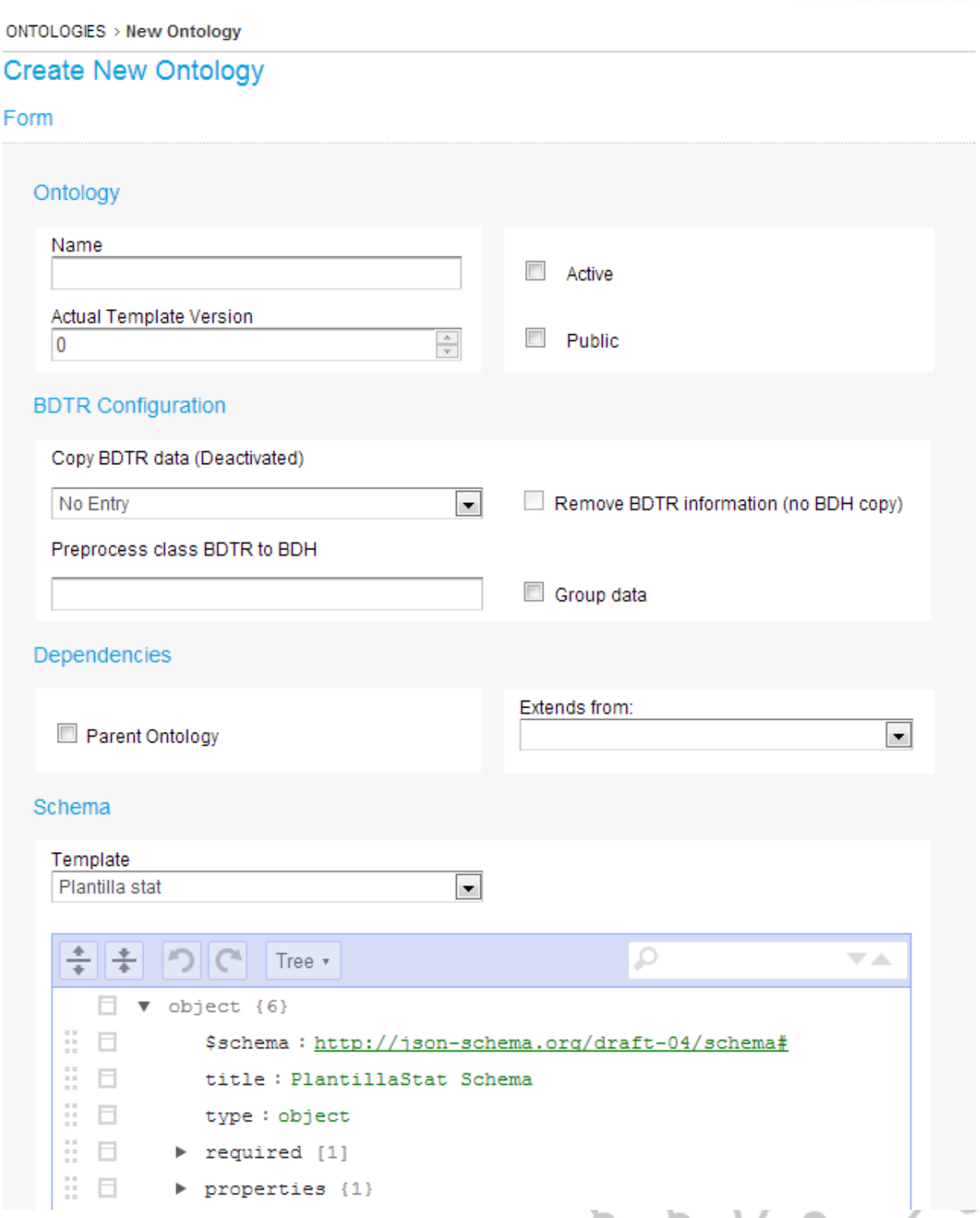
3 BASIC CONCEPTS

- **JSON**: is a lightweight format for data exchange (as XML, but less verbose)
- **JSON-Schema:** a JSON Schema is a JSON document that specifies the JSON document that it is referred to (if there are required attributes, if they are number type, if they can be null). In XML equivalence it would correspond to an XML schema or a DTD.
- **Ontology** formally defines a common set of terms used to describe and represent a domain.
- **Ontology SOFIA2:** In SOFIA2 an ontology is the definition of the set of classes and attributes that will share various applications that interoperate within the SmartSpace. In SOFIA2 ontologies are defined in JSON according to a JSON schema.
- **Instance of Ontology:** is a specific element of an ontology.

4 A FIRST GLANCE

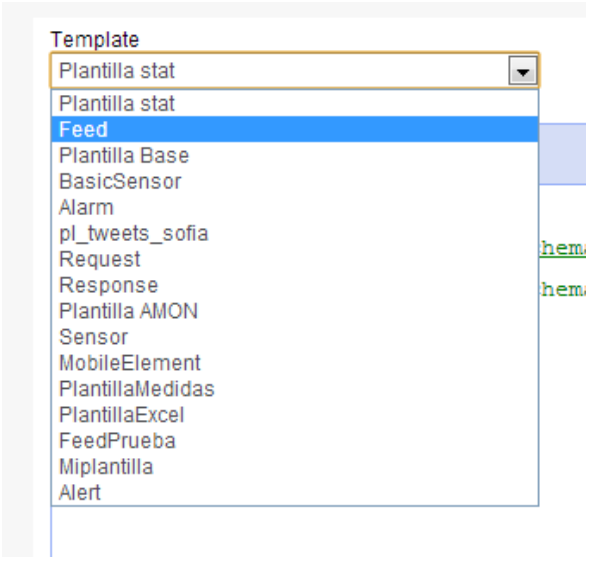
As we said in Sofia2 an ontology represents an entity in my system (SmartSpace), and is defined in **JSON**.

The Platform offers a **Configuration Web (+API REST)** in which users with permissions (collaborators and administrators) can create their ontologies.



An ontology is defined from a **JSON schema**.

To simplify the creation of Ontologies the Platform provides the concept of templates that are preloaded JSON schemas that can be used and expanded by the user to create their ontologies:



We will start defining a simple Ontology such as representing a temperture sensor that stores: identifier, timestamp, measurement, unit and GPS coordinates.

An instance of this ontology would be something like:

```
{
 "SensorTemperatura": {
  "identificador":"ST-TA3231-1",
  "timestamp":{"$date": "2014-01-27T11:14:00Z"},
  "medida":25.1,
  "unidad":"C",
   "geometry":{
     "type": "Point",
     "coordinates":[90,-10.1]
    }
 }
```

I can see how the ontologies JSON schema is defined in SOFIA2:

This ontology is public, which means that anyone can view the data.

If we click View  we will see the JSON Schema that describes the Ontology (in later sections we will go into detail about the syntax of this schema):


```
{"$schema":"http://json-schema.org/draft-04/schema#",
  "title":"SensorTemperatura Schema",
  "type":"object",
  "required":["SensorTemperatura"],
  "properties":{
    "_id":{
      "type":"object",
      "$ref":"#/identificador"
    },
    "SensorTemperatura":{
      "type":"string",
      "$ref":"#/datos"}
   },
   "additionalProperties":false,
   "identificador":{
      "title":"id",
      "description":"Id insertado del SensorTemperatura",
      "type":"object",
      "properties":{
        "$oid":{
          "type":"string"
        }
      },
      "additionalProperties":false
    },
  "datos":{
    "title":"datos",
    "description":"Info SensorTemperatura",
    "type":"object",
    "required":["identificador","timestamp","medida","unidad","coordenadaGps"],
    "properties":{
      "identificador":{
        "type":"string"
      },
      "timestamp":{
          "type":"object",
          "required":["$date"],
          "properties":{
            "$date":{
              "type":"string",
              "format":"date-time"
            }
          },
          "additionalProperties":false
```

```
            },
        "medida":{
            "type":"number"
        },
        "unidad":{
            "type":"string"},
        "geometry":{
            "$ref":"#/gps"
        }
    },
    "additionalProperties":false
    },
    "gps":{
        "title":"gps",
        "description":"Gps SensorTemperatura",
        "type":"object",
        "required":["coordinates","type"],
        "properties":{
            "coordinates":{
                "type":"array",
                "items":[
                {
                    "type":"number",
                "maximum":180,
                "mininum":-180
                },
                {
                    "type":"number",
                "maximum":180,
                "mininum":-180
                }
                ],
                "minItems":2,
                "maxItems":2
            },
            "type":{
                "type":"string",
                "enum":["Point"]
            }
        },
        "additionalProperties":false
    }
```

NOTA

In the schema we can see that the **timestamp** and **geometry** properties are defined in a special way. This is to allow searches by date and geospatial in MongoDB. Let´s look ay each individual case:

- **timestamp**: MongoDB allows working with dates in ***ISO 8601*** date (**YYYY-MM-DDThh:mm:ss.fffZ)** format**.** For MongoDB to interpret that a field is type date, it expects a JSON with a similar structure to {"$date": "2014-01-27T11:14:00Z"} represented. The schema that allows us to validate instances of this type is the following:

```
{   "timestamp":{
            "type":"object",
            "required":["$date"],
            "properties":{
                 "$date":{
                      "type":"string",
                      "format":"date-time"
                    }
              },
             "additionalProperties": false
         }
   }
```

This allows us to query like the following :

```
db.SensorTemperatura.find({"Sensor.created":{"$lt": new ISODate()}});
```

- **geometry:** MongoDB allows geospatial queries and this requieres that the fields involved are defined with the following structure:

```
{  "geometry": {
       "type": "object",
       "required":["coordinates","type"],
       "properties":{
             "coordinates":{
                   "type":"array",
                   "items":[
                         {
                           "type":"number",
                           "maximum": 90,
                           "minimum": -90
                         },
                         {
                           "type":"number",
                           "maximum": 180,
```

```
                                "minimum": -180
                            }
                        ],
                        "minItems":2,
                        "maxItems":2
                    },
                    "type":{
                        "type":"string",
                        "enum":["Point"]
                    }
                },
                "additionalProperties":false
            }
```

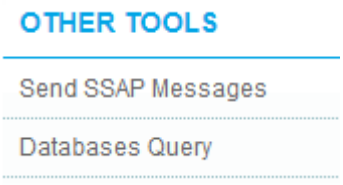
The *geometry* property consists of “Point” type and coordinates, representing a point, given by latitude and longitude (“coordenates”:[Latitud,Longitud]). The value range that MongoDB supports for these coordinates is between [90, -90] for the latitude and [180,-180] for the longitude. If you try to insert a value outside the range, MongoDB returns error.

An instance which complies with the following structure: {"geometry ": {"type":”Point”, “coordinates”:[1.9, -3.9]}}

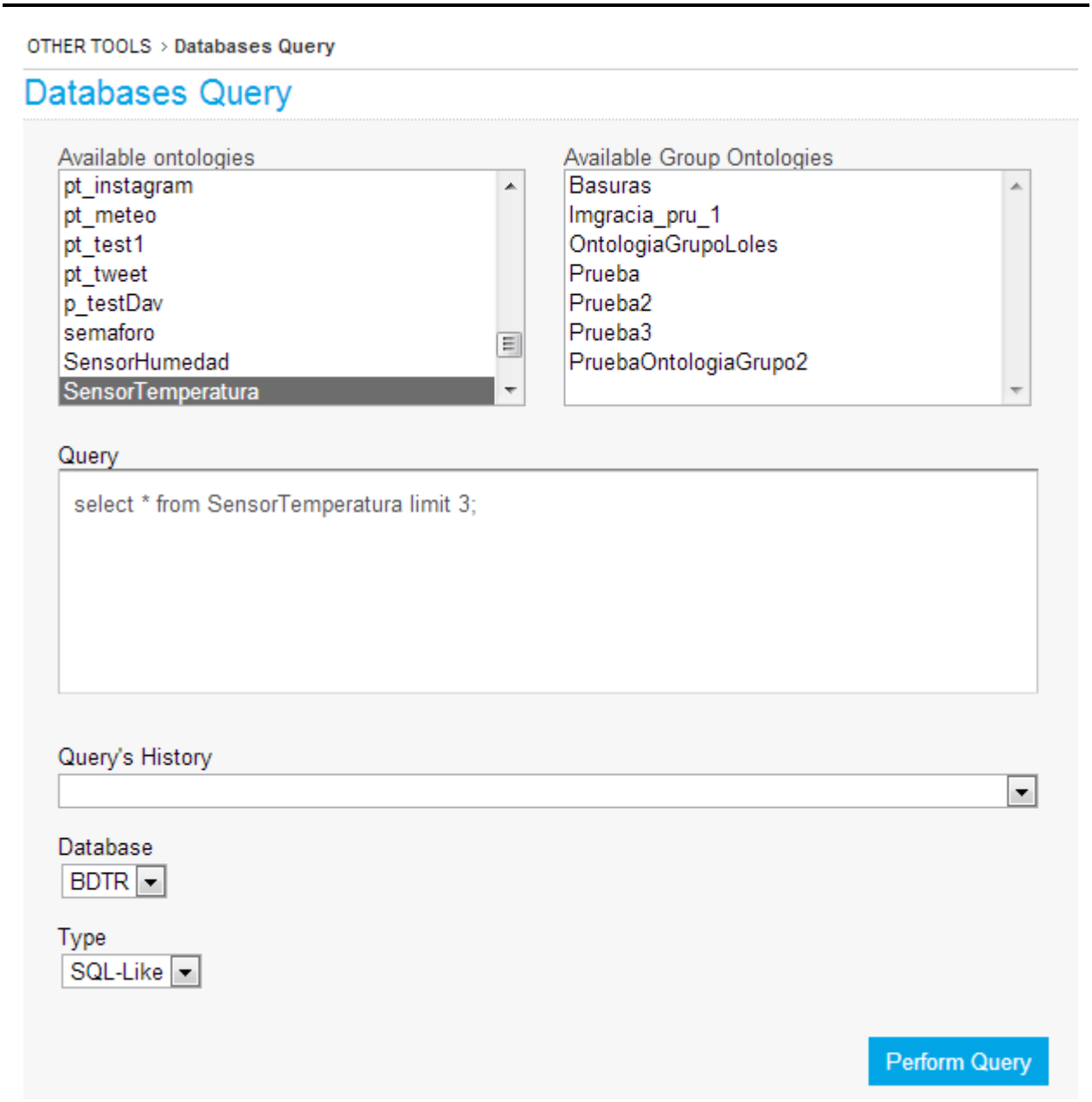
We can make geospatial searches in MongoDB like this:

```
db.SensorTemperatura.find({“Sensor.geometry.coordinates”:{$near:[12,12],$maxDistance:1}})
```

I can see the **instances of my ontologies** from the Configuration Web with the Query option to RTDB:



If I launch this query type:



I will see the information of the last instance inserted in RTDB of SOFIA2


```
{
  "_id": {
    "$oid": "51e3dbd465701fd8e0f69828"
  },
  "contextData": {
    "session_key": "08bf50c8-6ea6-41dc-99ac-5d12a6f517a3",
    "user_id": 1,
    "kp_id": 9,
    "kp_identificador": "gatewaysensores",
    "timestamp": {"$date": "2014-01-27T11:14:00Z"}
  },
  {
```

```
"SensorTemperatura": {
  "identificador":"ST-TA3231-1",
  "timestamp":{"$date": "2014-01-27T11:14:00Z"},
  "medida":25.1,
  "unidad":"C",
   "geometry":{
     "type": "Point",
     "coordinates":[90,-10.1]
    }
 }
}
```

The returned information includes:

- The instance **identifier**:

```
"_id": {
  "$oid": "52794144abc7d0b77e686c8b"
}
```

- **Context Information:** such as KP, instance, user, session and date in which in was inserted.

```
"contextData": {
 "session_key": "08bf50c8-6ea6-41dc-99ac-5d12a6f517a3",
 "user_id": 1,
 "kp_id": 9,
 "kp_identificador":  "gatewaysensores",
 "timestamp": {"$date": "2014-01-27T11:14:00Z"}
},
```

- **Instance of the Ontology**

```
"SensorTemperatura": {
 "identificador":"ST-TA3231-1",
 "timestamp":{"$date": "2014-01-27T11:14:00Z"},
 "medida":25.1,
 "unidad":"C",
  "geometry":{
    "type": "Point",
    "coordinates":[110.2,1233.1]
   }
}
```

5 IMPLIED TECHNOLOGIES

5.1 JSON

JSON stands for JavaScript Object Notation.

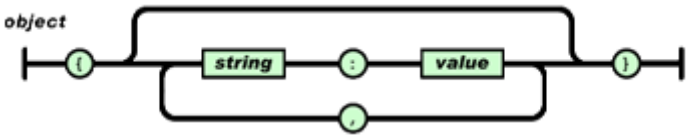
JSON is a light format originally designed for the data exchange on the Internet.

5.1.1 Data Types

- **string** : Text stringCadena de texto
- **number**: Numericl
- **object**: Object
- **char**: Valid Unicode characters
- **array**: Colection of values
- **null**: Null
- **boolean**: True or false values

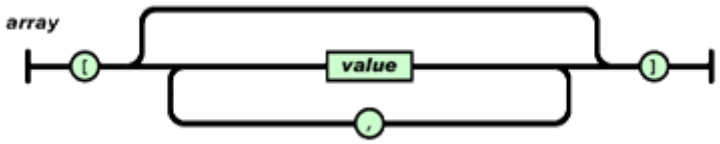
In JSON, they are presented in these ways:

An **object** is an unsorted set of key-value pairs. Starts with “{“ and ends with “}”. Each name is followed by “:”, the key-value pairs are separated by “,”.



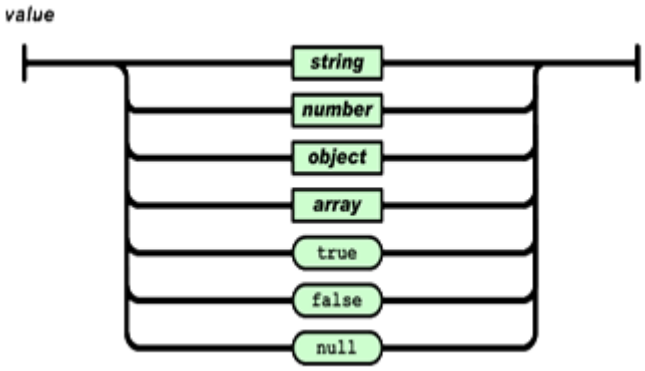

```
{
  “address” : {
    “line1” : “555 Main Street”,
    “city” : “Denver”,
    “stateOrProvince” : “CO”,
    “zipOrPostalCode” : “80202”,
    “country” : “USA”
  }
}
```

An **array** is a collection of values. Begins with “[“ and ends with “]”. The values are separated by “,”.

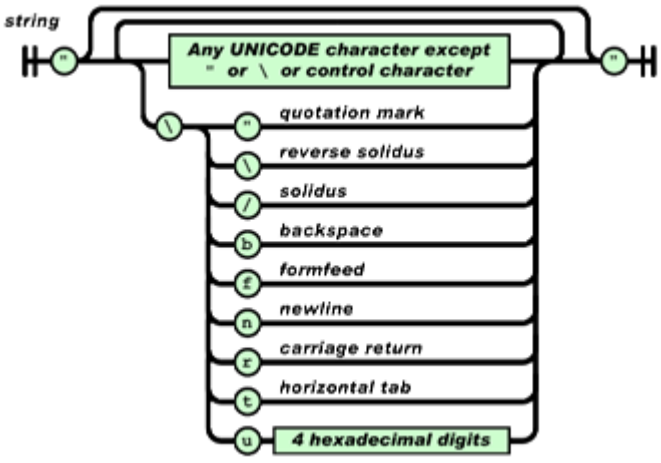



```
{
  "people" : [
    { "firstName": "John", "lastName": "Smith", "age": 35 },
    { "firstName": "Jane", "lastName": "Smith", "age": 32 }
  ]
}
```

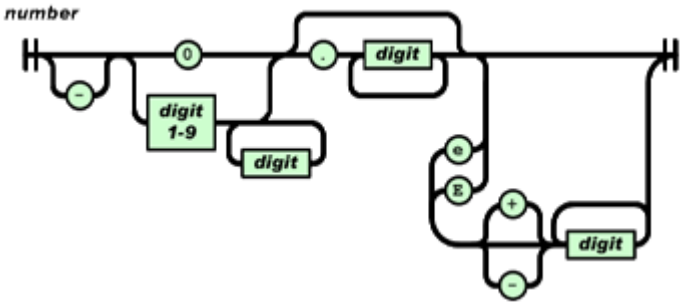
A **value** can be a character string with doublé quotation marks, number, true, false, null, an object or an array. This structure can be nested:



A **string** is a sequence of zero or more Unicode Characters, enclosed in double quotes (“)



A **number** is like a decimal number in Java.



5.1.2 References

http://cdn.dzone.com/sites/all/files/refcardz/rc173-010d-JSON_2.pdf

5.2 JSON-Schema

JSON-Schema (**http://json-schema.org**) is a JSON format to describe data in JSON. It is in JSON what XSD is to XML. It offers a contract to define the required data for a given application and how to interact with it.

5.2.1 Example

To get an idea we can see an example of a simple JSON schema:

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "Product",
    "description": "A product from Acme's catalog",
    "type": "object",
    "properties": {
        "id": {
            "description": "The unique identifier for a product",
            "type": "integer"
        },
        "name": {
            "description": "Name of the product",
            "type": "string"
        },
        "price": {
            "type": "number",
            "minimum": 0,
            "exclusiveMinimum": true
        }
    },
    "required": ["id", "name", "price"]
}
```

That validates as being valid JSONs like this:

```
{
```

```
    "id": 1,
    "name": "A green door",
    "price": 12.50,
    "tags": ["home", "green"]
}
```

And as invalid for not having the attribute price:

```
{
    "id": 1,
    "name": "A green door",
    "tags": ["home", "green"]
}
```

5.2.2 Attributes of a JSON schema

We can see the full reference of the JSON specification here: **http://json-schema.org/latest/json-schema-core.html**

```
    {
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "Product",
    "description": "A product from Acme's catalog",
    "type": "object",
    "properties": {
        …
        …
        …
        …
    },
    "required": ["id", "name", "price"]
}
```

The attributes most commonly used by JSON schema are:

- ***"$schema"***: Allows us to define the JSON Schema versión we want to use: 0.4 o 0.3, SOFIA2 is supported in version 0.4 (*http://json-schema.org/draft-04/schema#*).
- **"title"**: indicate a title to indentify the schema.
- **"description"**: This attribute can be used to include a description of what is represented in the JSON Schema.
- **"type"**: To indicate the type the schema is going to represent.
- **"properties"**: This attribute is an object with the definitions of properties that define the static values of an object. It is an unordered list of properties. The property names must

be fulfilled and the property values are defined from a schema, which must also be fulfilled.

- **"patternProperties"**: This attribute is an object with the definitions of properties that define the values of an object instance. It is an unordered list of properties. The property names are regular expression patterns, instances of properties must meet the defined standard and the proporty value with the schema that defines the property.
- **"additionalProperties"**: To indicate whether the JSON instance can contain properties that are not defined in the schema. It has two possible values (true or false), to indicate whether any property is acknowledged or not. If the property is not added, it may include any other property.
- **"required"**: Allows to specify all the properties that are required for a JSON instance and must be included at least. The properties will be included between brackets and separated by the character ",".
- **"$ref"**: Defines a URI of a schema that contains the complete representation for that property.

Considering this extract from a simple schema for the defined attributes

```
{
  "$schema":"http://json-schema.org/draft-04/schema#",
  "title":"SensorTemperatura Schema",
  "type":"object",

  "required":["SensorTemperatura"],

  "properties":{
    "_id":{
      "type":"object",
      "$ref":"#/identificador"
    },
    "SensorTemperatura":{
      "type":"string",
      "$ref":"#/datos"}
  },
  "additionalProperties":false,
```

In this example we can see there is a property that is mandatory "**SensorTemperatura**" and two properties **"_id**" y "**SensorTemperatura**", that include a reference to an element that contains in the complete representation of the property.

```
  "identificador":{
      "title":"id",
```

```
        "description":"Id insertado del SensorTemperatura",
        "type":"object",
        "properties":{
          "$oid":{
            "type":"string"
          }
        },
        "additionalProperties":false
      },

  "datos":{

      "title":"datos",
      "description":"Info SensorTemperatura",
      "type":"object",
      "required":["identificador","timestamp","medida","unidad","coordenadaGps"],
      "properties":{
        "identificador":{
          "type":"string"
        },
        "timestamp":{
              "type":"object",
              "required":["$date"],
              "properties":{
                "$date":{
                  "type":"string",
                  "format":"date-time"
                }
              },
              "additionalProperties":false
              },
        "medida":{
          "type":"number"
        },
        "unidad":{
          "type":"string"},
        "geometry":{
          "$ref":"#/gps"
        }
      },
    "additionalProperties":false
```

As we can see both “identificador” (id) and “datos” (data) are schemas that define their representation. We can also see that any property that is not defined is not supported (“additionalProperties” has been included).

- **Listed**: Those listed will be represented as a list in brackets and separated by the character ",". Those listed are always "string" type. For example is we want to difne a property called "tipo" that can only have two values "latitude" o "longitude", it would be as follows:

```
"tipo":{
        "type":"string",
        "enum":["latitude","longitude"]
}
```

  To instantiate, *"tipo": "latitude"*

- **"ítems":** Defines the allowed elements in an array, it must be a schema or a set of schemas.
- **"additonalItems":** To indicate whether ítems in the array, on addition to those defined in the schema.
- **"minItems":** Minimum number of elements that the array can have.
- **"maxItems":** Maximum number of elements the array can have.

In the following example we can see the schema for an array, "coordinates", must be numeric type and can only have two elements. We also see that the "type" property, is listed with a single possible value "Point".

```
"geometry":{
    "type": "object",
    "required":["coordinates","type"],
    "properties":{
         "coordinates":{
              "type":"array",
              "items":{
                   "type":"number"
              },
              "minItems":2,
              "maxItems":2
          },
         "type":{
              "type":"string",
              "enum":["Point"]
           }
    },
      "additionalProperties":false
}
```

An instance for this object would be like the following

```
"geometry":{
      "type": "Point",
      "coordinates":[110.2,1233.1]
  }
```

For more information and examples go to the following link: **http://json-schema.org/**