



HOW TO DEVELOP ON THE SOFIA2 PLATFORM

APRIL 2016

Version 4



1 INDEX

1	INDEX	2
2	INTRODUCTION	4
2.1	REQUIREMENTS	4
2.2	GOALS AND SCOPE OF THE CURRENT DOCUMENT	4
3	SOFIA2'S SDK INSTALLATION	5
3.1	SOFIA2'S SDK DOWNLOAD AND INSTALLATION	5
3.2	STARTING UP SOFIA2'S SDK	6
3.3	ELEMENTS MAKING UP SOFIA2'S SDK	7
3.3.1	IDE	7
3.3.2	Architecture Console	8
3.3.3	API	9
4	SOFIA2 APPLICATION DEVELOPMENT	11
4.1	REGISTERING IN THE SOFIA2 PLATFORM	11
4.1.1	Joining the platform	11
4.2	INFORMATION ONTOLOGIZATION	13
4.2.1	Identifying the information concepts	14
4.2.2	Identifying the concepts' attributes	14
4.2.3	Modelling using JSONSchema format	15
4.2.4	Adding a new ontology to the Platform	17
4.3	SOFIA2 CLIENT (KP) DEVELOPMENT	18
4.3.1	Creating permissions for the user who owns that KP in the Platform	19
4.3.2	Registering a KP to the Platform	20
4.3.3	Connection of KP with Platform	21
4.3.4	Catching/Exploiting Information	21
4.3.5	Translating the information to an ontological format	22
4.3.6	Sending to the Platform using SSAP	22

4.3.7	Query/Subscribe to information using SSAP	22
4.3.8	Receiving ontologically-formatted information.....	23
4.4	REAL-TIME COLLABORATION BETWEEN KP'S	23
5	A PRACTICAL EXAMPLE	25
5.1	ADDING THE USER TO THE PLATFORM	26
5.2	INFORMATION ONTOLOGIZATION	29
5.3	DEVELOPING THE INFORMATION PRODUCER KP	33
5.4	DEVELOPING THE INFORMATION CONSUMER KP	40
5.5	RUNNING	52



2 INTRODUCTION

2.1 Requirements

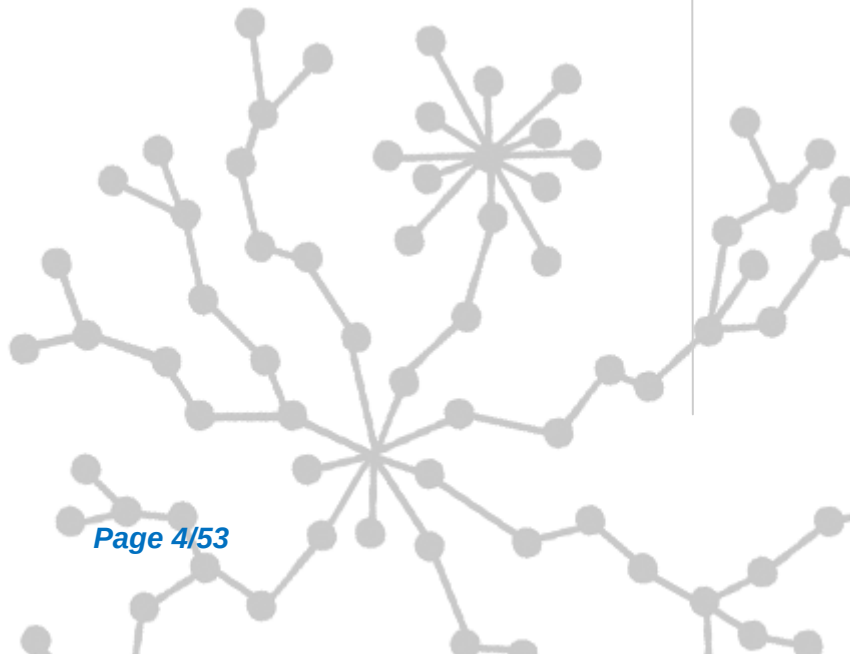
Before Reading this guide, we suggest you to read (EN) **SOFIA2-SOFIA2 Concepts.pdf**

To develop SOFIA2 applications, you will need to have installed SOFIA2's SDK as explained in the point **SOFIA2's SDK Installation**.

2.2 Goals and scope of the current document

The current document describes the steps to develop applications on the SOFIA2 platform, including:

- Installing the SOFIA2's SDK.
- Registering in the SOFIA2 Platform.
- Ontologization Process.
- Development of SOFIA2 Clients (KP's) covering from the different registrations and additions to be made on the platform for those applications, to the data production and consumption by those.



3 SOFIA2'S SDK INSTALLATION

SOFIA's **SDK** offers a completely operative version to develop on the SOFIA Platform.

To work with SOFIA's SDK, you must have installed SOFIA's Runtime.

3.1 SOFIA2's SDK Download and Installation

1. Download the Environment's last version from this URL:

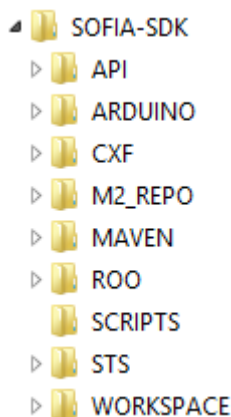
http://sofia2.com/desarrollador_en.html#descargas

Access the section Downloads.

2. Uncompress the ZIP file that you have downloaded. It must be named **SOFIA2_SDK.zip**.



3. You will find all the components making up the SDK inside the **SOFIA-SDK** directory.



- The **development environments** for the C environments in Arduino and Java are in the **ARDUINO** and **STS** directories, respectively.
- The **ROO** directory includes the Architecture console with the SOFIA commands to create Plugins, KP's and Gateways.
- The **CXF** directory includes the productivity tools to work with WSDL and WADL.
- The **MAVEN** and **M2_REPO** directories include the Maven Tool and the dependencies it uses to develop on the SOFIA Platform.
- The **API** directory includes Javascript and Java API's to develop KP's.

- The  **WORKSPACE** directory includes the IDE's' configuration directories.
- The  **SCRIPTS** directory provides the batch processes for the different components making up the SDK.

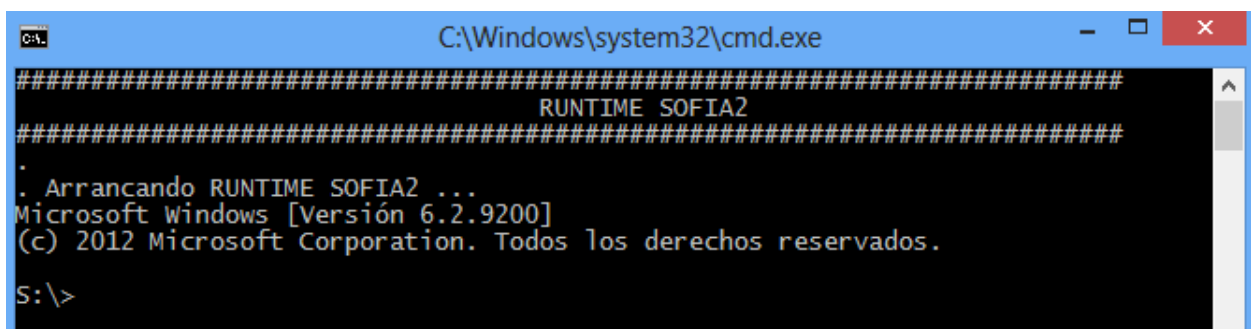
 ARDUINO
 ARQSPRING
 ARQSPRINGWSDL2JAVA
 IDE

3.2 Starting up SOFIA2's SDK

Once SOFIA2's SDK has been downloaded, we can find the following contents in the  **SOFIA** root directory:

 SOFIA2_API_JAVA
 SOFIA2-SDK
 SOFIA-RUNTIME
 SOFIA2_API_JAVA.zip
 Sofia2_IDE.bat
 Sofia2_SDK-START.bat
 Sofia2_SDK-STOP.bat
 Sofia2_VariablesEntorno.bat

By double-clicking the batch process  **Sofia2_SDK-START.bat** we will create the virtual drive **S:** and open the command console.



```
C:\Windows\system32\cmd.exe

#####
                        RUNTIME SOFIA2
#####

. Arrancando RUNTIME SOFIA2 ...
Microsoft Windows [Versión 6.2.9200]
(c) 2012 Microsoft Corporation. Todos los derechos reservados.

S:\>
```

This command console's session is configured to run with SOFIA's SDK and Runtime.

Installing SOFIA2's SDK modifies the  **Sofia2_SDK-START.bat** file, created by the Runtime. Due to this, installation directories must be observed.

IMPORTANT

If the command console is closed, we must re-run the batch process  Sofia2_SDK-START.bat to re-open the console with the predefined configuration.

3.3 Elements making up SOFIA2's SDK

The SOFIA Platform is made up of these elements:

- IDE
- Indra's iTR Technical Architecture Console
- Maven3 Local Repository with all the dependencies.
- SOFIA2's API's
-

3.3.1 IDE

It is the Platform's Development Environment.

SOFIA2 distributes 2 IDE's:

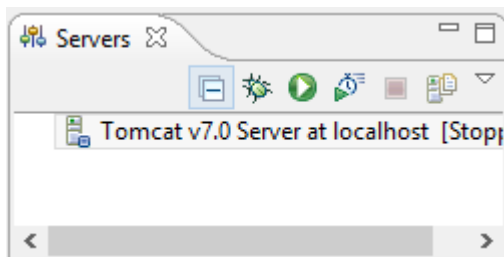
- IDE to develop on Java and JavaScript, based on Eclipse.
- IDE to develop on Arduino.

3.3.1.1 Java IDE

Run the command **S:\>Sofia2_IDE** in the console.

```
S:\>Sofia2_IDE
#####
                        SDK SOFIA2
#####
. Arrancando IDE Sofia2...
.
```

The iTR Architecture Eclipse IDE to develop SOFIA2 applications will open.

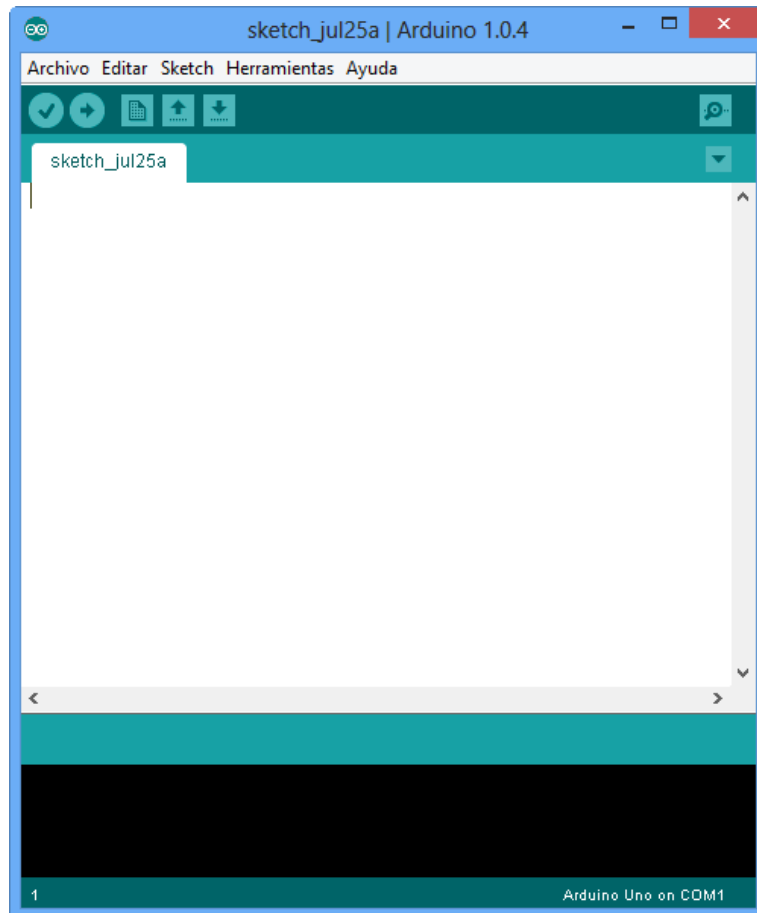


3.3.1.2 Arduino IDE

Run the command **S:\>arduino** in the console.

```
S:\>arduino
#####
                        SDK SOFIA2
#####
.
. Arrancando ARDUINO...
.
```

The IDE to develop Arduino clients will open.



3.3.2 Architecture Console

It offers productivity tools to easily create SOFIA2 Projects.

Run the command **S:\>arqspring** in the console.

```
S:\>arqspring
#####
                        SDK SOFIA2
#####
.
. Arrancando ArqSpring...
.
```

The iTR Architecture console will open.

It provides us with the command `ArgSpring>sofia create(Gateway, KP, Plugin)`

```
sofia2 configurarKp      sofia2 crearAppModelo    sofia2 crearKp
```

3.3.2.1 Command to generate Java KP's

You can read the detail on how to use the aforementioned commands in the guide **SOFIA2 – Herramientas Productividad SOFIA SDK**

For more detail on the development of Plugins, please refer to chapter 4 in the Guide

 [SOFIA-Guía Configuración Extensión y Personalización SIB](#)

The **/SOFIA-SDK/API** directory includes the APIs provided by the SOFIA2 platform to develop applications that will interact with it.

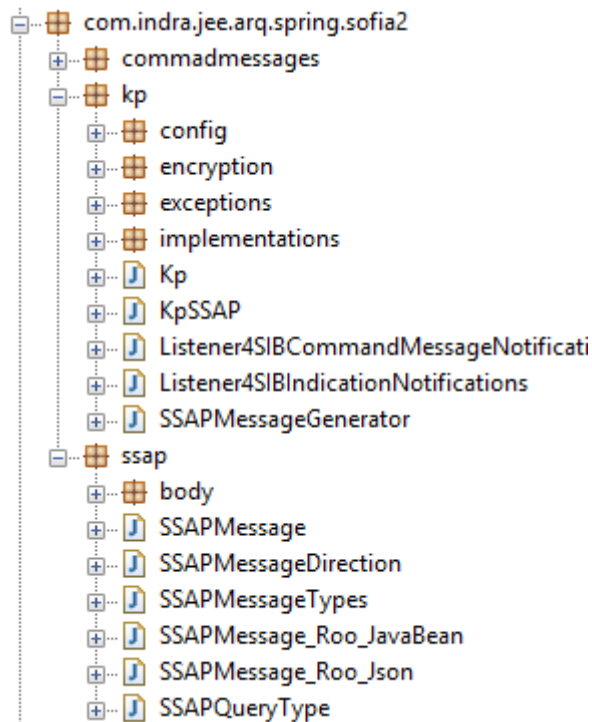
Read the Guides Platform's API's.

 (EN) SOFIA2-SOFIA2 APIs.pdf

for more information on the use of SOFIA

3.3.3.1 Java API

The Java API is in the path `SOFIA-SDK > API > API-KP-JAVA`, in a Jar called  `kp-core-1.0.0`, and it has the following package structure:



IMPORTANT

For more detail on the Java API, refer to **chapter 5** of  (EN) SOFIA2-SOFIA2 APIs.pdf

3.3.3.2 JavaScript API

The JavaScript API is available in the SIB itself.  `API-KP-JAVASCRIPT` is a direct access to the path `SOFIA-RUNTIME > SIB > webapps > sib-web > js` that contains the js file with the KP's API

 `kp-core`

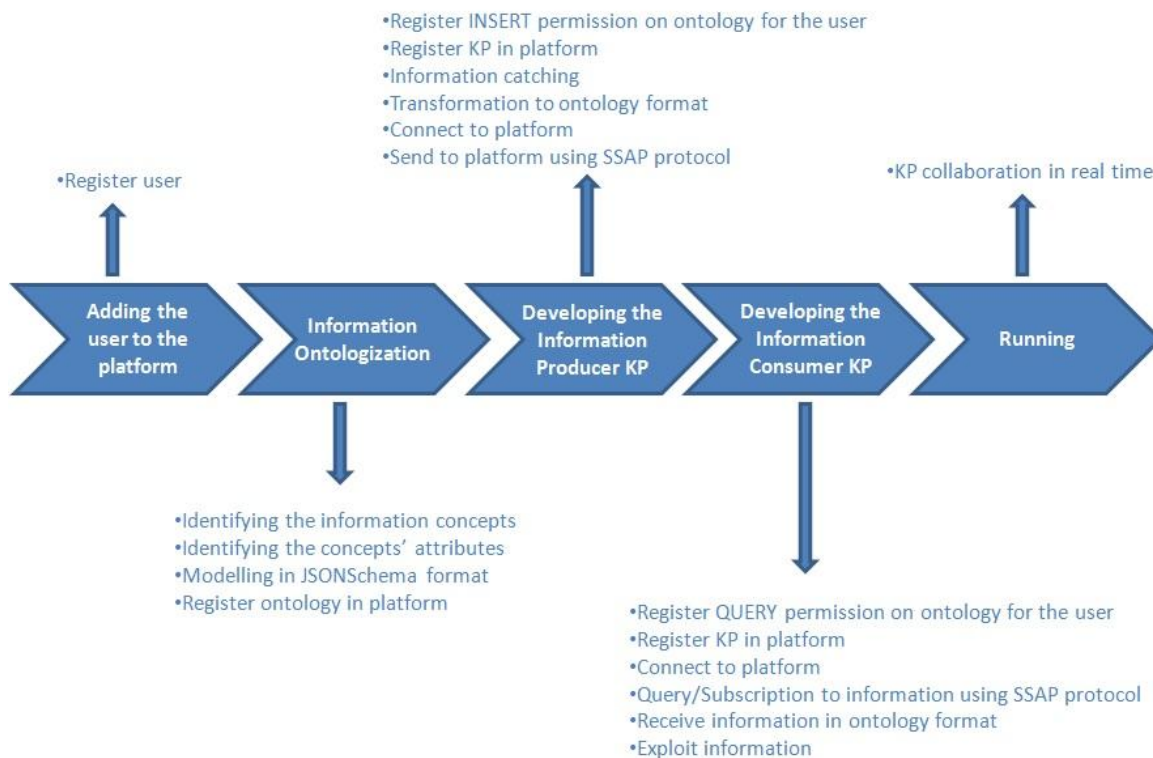
IMPORTANT

For more detail on the JavaScript API, refer to **chapter 8** of  (EN) SOFIA2-SOFIA2 APIs.pdf

4 SOFIA2 APPLICATION DEVELOPMENT

The SOFIA2 Platform allows for the interoperability of different applications (KP's) from concepts that are common in their domains, and that are modelled using the corresponding ontology. This allows for the standardization of the information which the applications exchange with each other.

Once the SOFIA2's SDK is installed, the steps to develop on the SOFIA Platform are the following ones:



4.1 Registering in the SOFIA2 Platform

The SOFIA Platform is a controlled interoperability environment.

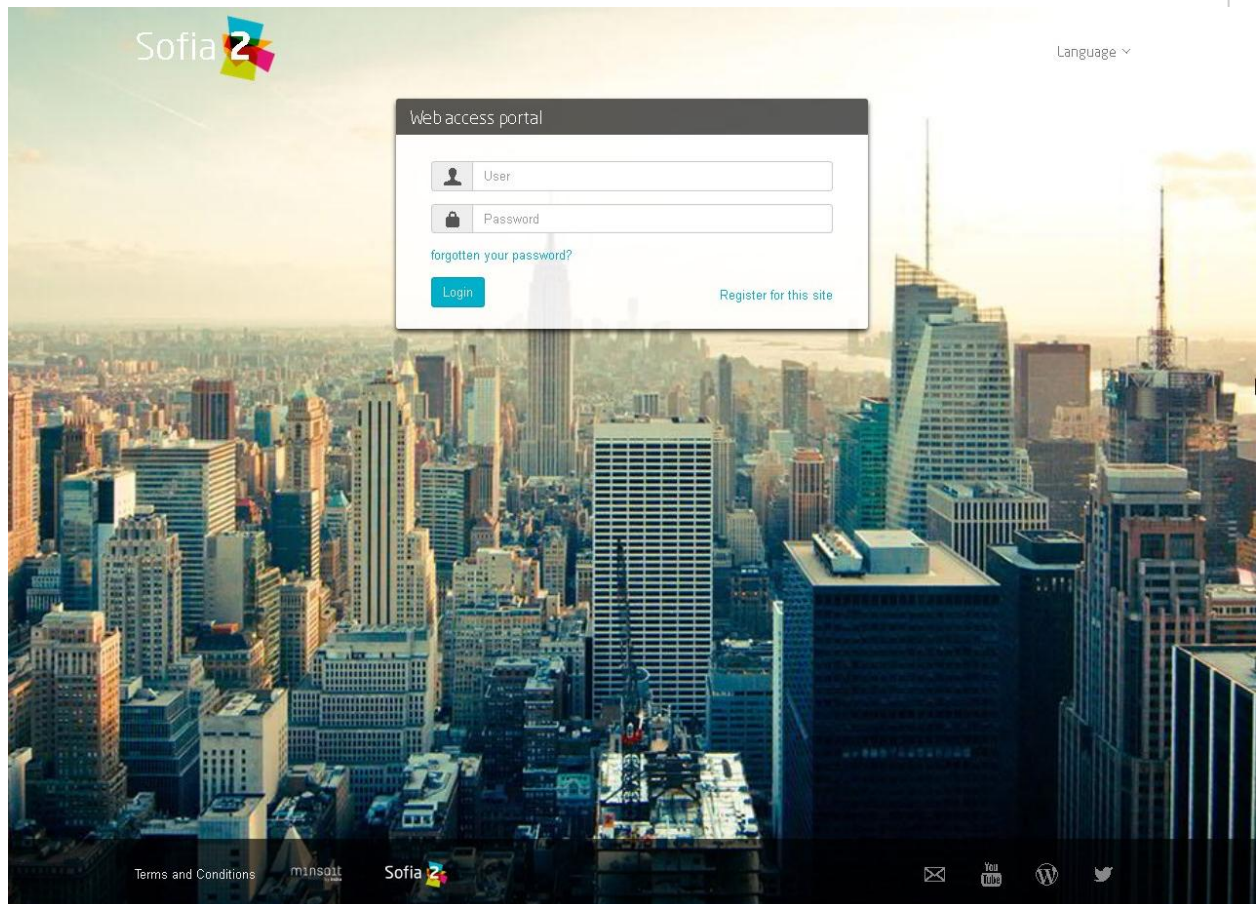
Different KP's exchange information to interoperate.

Each KP is owned by one user. One same user can be the owner of several KP's.

4.1.1 Joining the platform

Users can join the platform autonomously, or an administrator can add the users to the system.

- **Joining autonomously:** Access the URL <http://sofia2.com/console/login;lang=en>. The login screen offers the link under the caption Register for this site

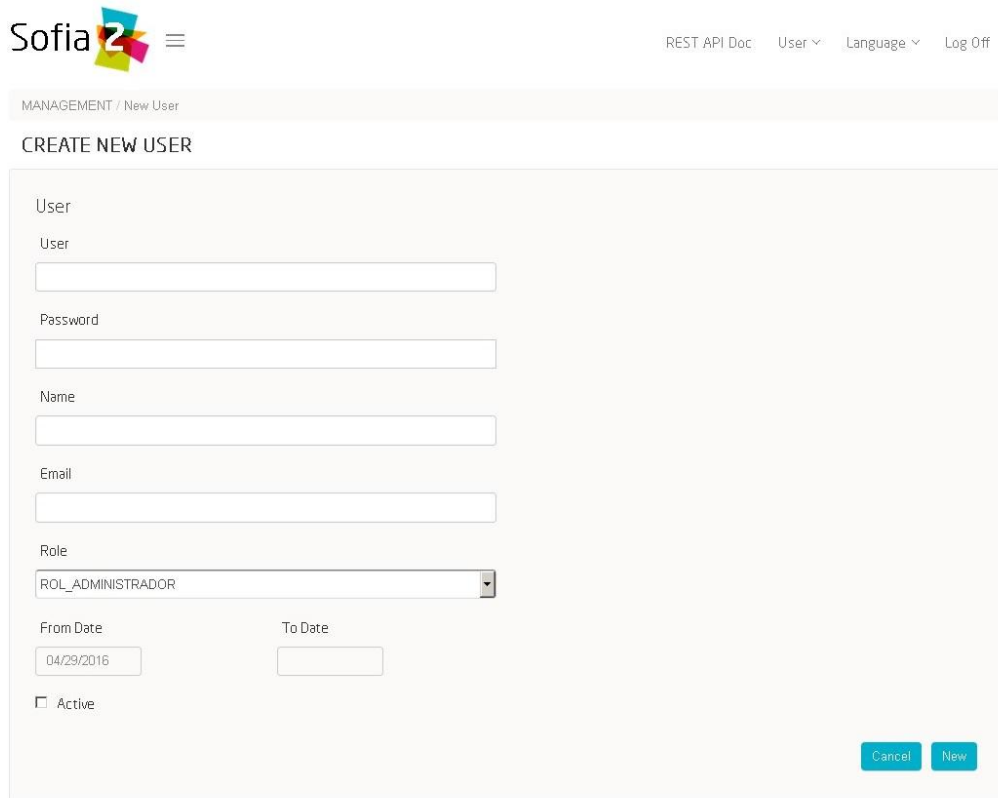


Which re-directs to a user registration form:

The image shows the 'User Registration Request' form. It has a dark header with the title. Below the header are four input fields: 'User' (with a person icon), 'Password' (with a lock icon), 'Name' (with a person and plus icon), and 'Email' (with an envelope icon). Below these fields is a checkbox with the text: 'By clicking "Register", you agree to the [Terms and Conditions](#) and confirm that you read our data policy, including the use of cookies.' At the bottom left is a blue 'Sign in' button, and at the bottom right is a 'Back to register' link.

By joining autonomously, you get the role **USER** (“USUARIO”) in the platform, lacking any special permissions. These users must ask an administrator to get special permissions.

- **Added by administrator:** The administrator can access a **User Management** (“Gestión de Usuarios”) area in the Platform. This area provides the administrator with a form to register new users in the platform, and to give them any needed role.



The screenshot shows the Sofia 2 web interface. At the top, there's a header with the Sofia 2 logo, a menu icon, and links for 'REST API Doc', 'User', 'Language', and 'Log Off'. Below the header, a breadcrumb trail reads 'MANAGEMENT / New User'. The main section is titled 'CREATE NEW USER' and contains a form with the following fields:

- User:** A text input field.
- Password:** A text input field.
- Name:** A text input field.
- Email:** A text input field.
- Role:** A dropdown menu currently showing 'ROL_ADMINISTRADOR'.
- From Date:** A date input field showing '04/29/2016'.
- To Date:** An empty date input field.
- Active:** A checkbox labeled 'Active'.

At the bottom right of the form, there are two buttons: 'Cancel' and 'New'.

Once the user is registered in the Platform, the user can access the Platform according to her role. We will later see that the user can become the owner of several KP's, and those KP's can establish a logical session to send and receive information.

4.2 Information Ontologization

The SOFIA Platform offers the following as the key of application interoperability: The ontological definition of the information that applications exchange with each other.

An ontology is a classification of information, standardizing the property of domain concepts with which the different KP's will interoperate. Thus, different KP's working with the same ontologies can exchange information through the platform in a completely uncoupled way, by exchanging instances of those ontologies.

SOFIA2 suggest the information exchange between KP's to use JSON format. As an ontology is, in SOFIA2, the unambiguous specification of a JSON-formatted information, those ontologies are defined following a JSON schema.

<http://json-schema.org/>.

4.2.1 Identifying the information concepts

Its object is identifying the entities which will group the data exchanged by the KP's through the Platform.

One concept will cover one or several related data, which is or are interesting for the KP's. Thus, one information producer KP will collect all these data, group them following the specification that was defined in the ontology describing that concept, and send them that way, normalized to the Platform. On the other hand, an information consumer KP will receive and exploit all these normalized data.

For instance, a SmartCity sensorization SmartSpace will have the following relevant concepts to be ontologized:

- TemperatureSensor
- HumiditySensor
- CO2Sensor
- MovementSensor
- TrafficLight
- Sprinkler
- .....

4.2.2 Identifying the concepts' attributes

Its goal is identifying the data grouped following the information concepts, and which are relevant for the KP's.

For instance, let us take the concept TemperatureSensor. We may consider the following attributes:

- **SensorIdentifier**: string
- **Timestamp**: integer

- **Measurement:** double
- **Unit:** string
- **GPS Location:** Object
  - **Altitude:** double
  - **Latitude:** double
  - **Longitude:** double

4.2.3 Modelling using JSONSchema format

After identifying the data which will be exchange, the next step is standardizing that data so that they have an unambiguous definition as applied to the KP's in the Platform. This is the goal of the information ontologization, where every relevant concept is defined following a JSON schema.

For instance, the TemperatureSensor concept, with the previously identified attributes, can be defined in JSONSchema format like this:

TemperatureSensor.json

```
{
  "$schema": "http://json-schema.org/draft-03/schema#",
  "title": "TemperatureSensor Schema",
  "type": "object",
  "properties": {
    "_id": {
      "type": "object",
      "$ref": "#/identifier"
    },
    "TemperatureSensor": {
      "type": "string",
      "$ref": "#/data"
    }
  },
  "identifier": {
    "title": "id",
    "description": "inserted Id for TemperatureSensor",
    "type": "object",
    "properties": {
      "$oid": {
        "type": "string",
```

```

        "required": false
      }
    }
  },
  "data": {
    "title": "data",
    "description": "Info TemperatureSensor",
    "type": "object",
    "properties": {
      "identifier": {
        "type": "string",
        "required": true
      },
      "timestamp": {
        "type": "integer",
        "minimum": 0,
        "required": true
      },
      "measurement": {
        "type": "number",
        "required": true
      },
      "unit": {
        "type": "string",
        "required": true
      },
      "GpsLocation": {
        "required": true,
        "$ref": "#/gps"
      }
    }
  },
  "gps": {
    "title": "gps",
    "description": "Gps TemperatureSensor",
    "type": "object",
    "properties": {
      "altitude": {
        "type": "number",
        "required": false
      },
      "latitude": {
        "type": "number",

```



```
    "required": true
  },
  "longitude": {
    "type": "number",
    "required": true
  }
},
"additionalItems": false
}
```

Thus, the information produced and consumed by KP's and coming from this kind of sensor will be standardized following this format:

TemperatureSensor-instance.json

```
{
  "TemperatureSensor": {
    "identifier": "ST-TA3231-1",
    "timestamp": 1357930309163,
    "measurement": 25.1,
    "unit": "C",
    "GpsLocation": {
      "altitude": 0.0,
      "latitude": 40.512967,
      "longitude": -3.67495
    }
  }
}
```

4.2.4 Adding a new ontology to the Platform

A new ontology must be registered in the Platform before it is operative and available for the KP's to produce or consume information described in that ontology.

To do so, the Platform has a section of **Ontology Management**, where ontologies can be administered, added and edited:

CREATE NEW ONTOLOGY

Ontology

Name

Actual Template Version

0

☐ Active
 ☐ Public

RTDB and HDB configuration

Remove RTDB information (no BDH copy) of the previous :

1 day

☒ Remove RTDB information (no BDH copy)

Dependencies

Extends from:

☐ Parent Ontology

Special Subscriptions

☐ Optimize masive Select all subscriptions

Optimize suscriptions for queries by field:

Add query by field

Where Field

Schema

☒ Template ☐ File xsd

Template

Log

Visual Modeler

Tree

- object (9)
  - \$schema : <http://json-schema.org/draft-04/schema#>
  - title : Logs
  - type : object
    - required [1]
    - properties (1)
      - datos (4)
        - point (4)
        - linestring (4)
        - polygon (4)

You can read the detail of the Ontology Management operative in the

 (EN) SOFIA2-Web Console Use Guide.pdf

4.3 SOFIA2 Client (KP) Development

A KP is any application that produces or consumes information to collaborate with others through the Platform, thus creating a **SmartSpace** with the KP's it works with.

To develop a KP you must program its business logic and also perform the following steps on the Platform:

4.3.1 Creating permissions for the user who owns that KP in the Platform

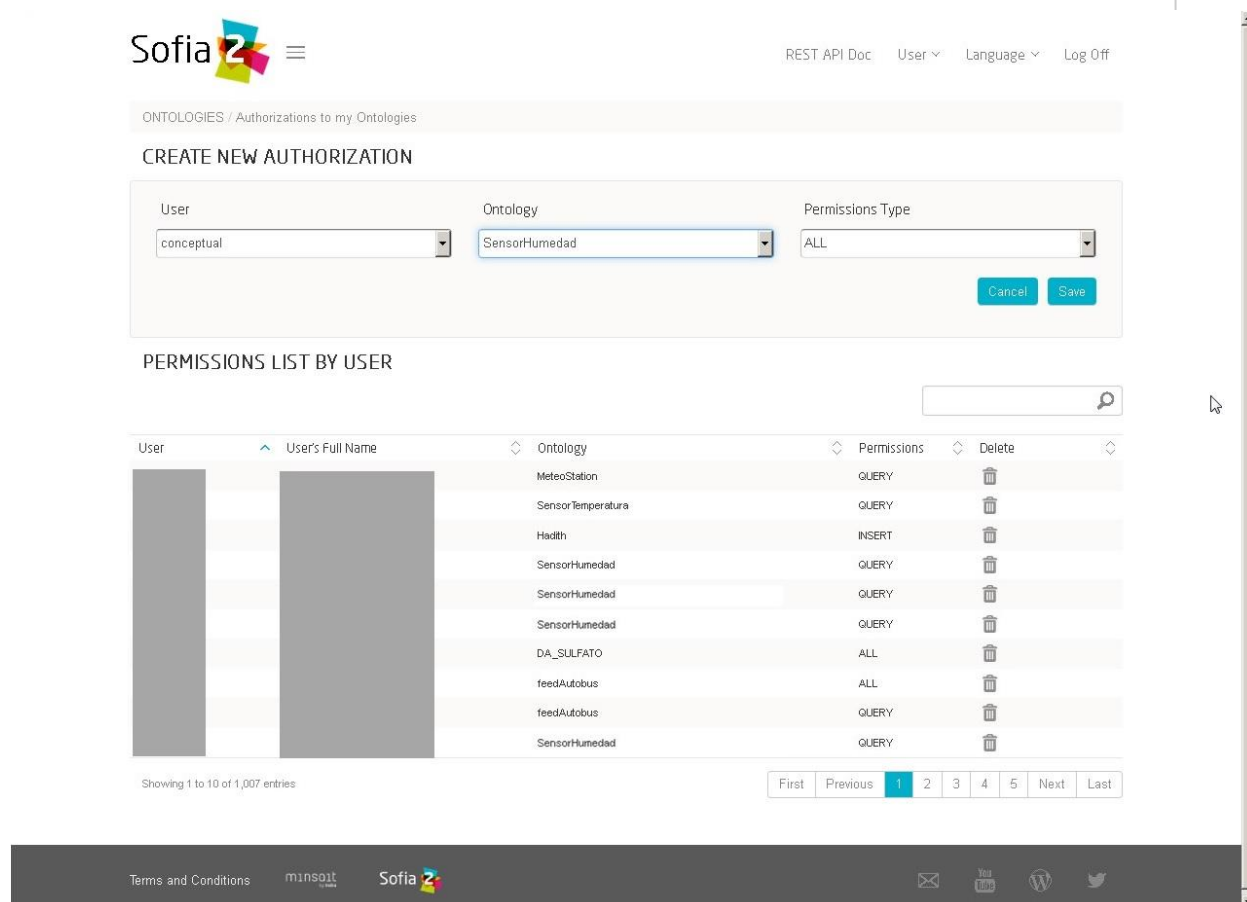
For a user's KP's to produce or consume data from a given ontology, that user must have the right permissions on that ontology.

An ontology that is registered in the Platform may not be visible for a certain user, or it may be visible but that user is limited in operating it due to lack of permissions.

The platform provides administrators, through the section **Ontologies > Authorizations to my Ontologies**, a screen to manage a user's authorizations on the different registered ontologies.

You can read the detail of the User Management operative in the

 (EN) SOFIA2-Web Console Use Guide.pdf



The screenshot shows the Sofia 2 web console interface. At the top, there's a navigation bar with the Sofia 2 logo, a hamburger menu, and links for REST API Doc, User, Language, and Log Off. Below the navigation bar, the breadcrumb trail reads 'ONTOLOGIES / Authorizations to my Ontologies'. The main section is titled 'CREATE NEW AUTHORIZATION' and contains three dropdown menus: 'User' (selected: conceptual), 'Ontology' (selected: SensorHumedad), and 'Permissions Type' (selected: ALL). There are 'Cancel' and 'Save' buttons. Below this is a section titled 'PERMISSIONS LIST BY USER' which contains a table with columns: User, User's Full Name, Ontology, Permissions, and Delete. The table lists several entries for the 'conceptual' user, including permissions for 'MeteoStation', 'SensorTemperatura', 'Hadith', 'SensorHumedad', 'DA_SULFATO', 'feedAutobus', and 'SensorHumedad'. At the bottom of the table, it says 'Showing 1 to 10 of 1,007 entries'. The footer of the page includes links for Terms and Conditions, minsoft, and Sofia 2, along with social media icons for Email, YouTube, WordPress, and Twitter.

Thus, depending on the KP's that the user is going to develop, the user must receive INSERT, **QUERY** or **ALL** permission on the ontology describing the data that the KP will use.

4.3.2 Registering a KP to the Platform

Users must register their KP's in the Platform. Otherwise, the Platform will reject the connection of that KP's.

To register a KP, the Platform provides an area of **KP Management**, where a user can create a new KP or manage those KP's that have been created previously:

The screenshot shows the 'CREATE NEW KP' form in the Sofia 2 web console. The form includes the following fields and options:

- KP Identification:** A text input field.
- Encryption Key:** A text input field containing the value '89dd770d-820d-4363-8b1c-a8ff584d36d'.
- Description:** A large text area.
- Ontology:** A dropdown menu with the following options: 'CompostSensor', 'meteoStat13', 'SensorHumedad', and 'SensorTemperatura'. The 'SensorTemperatura' option is currently selected.
- Group Ontologies:** A checkbox labeled 'Group Ontologies' which is currently unchecked.
- Meta-Inf:** A text area.
- Buttons:** 'Cancel' and 'New' buttons at the bottom right.

The top of the page shows the 'Sofia 2' logo, a hamburger menu icon, and links for 'REST API Doc', 'User', 'Language', and 'Log Off'. The breadcrumb trail indicates the user is in 'KP's/APPS SOFIA2 / New KP'.

As we can see, a KP can make use of one or several ontologies. This is the Platform's information that the KP will produce or consume.

Once the KP is registered in the Platform, the KP can establish connections with the Platform.

You can read the detail of the KP Management operative in the

 (EN) SOFIA2-Web Console Use Guide.pdf

4.3.3 Connection of KP with Platform

The connection of a KP with the platform must be understood in two layers.

- **Physical connection:** Established by the transport protocol used by a KP to connect (TCP/IP, MQTT, JMS, Ajax-Push...). The specifics to achieve this kind of connection greatly depend on the used KP's API (Java, JavaScript, Arduino, C++...).
- **Logical connection:** Established by the SSAP (Smart Space Access Protocol), the messaging protocol defined in SOFIA. It is common to all the KP's API's.

We will now focus in the logical connection that a KP must keep with the Platform.

For a KP to connect to the Platform and produce or consume data and interoperate with other KPs, the KP must first open a session with a SIB in the Platform.

The SSAP provides two operations in that sense:

- **JOIN:** Where a KP reports its owner's user name, password and instance data to the Platform. If everything is correct, the Platform authenticates the KP and opens a session with it.
- **LEAVE:** Where a KP wants to close the session and reports that to the Platform.

If there is a session between the KP and the Platform, the KP can use the other commands in the SSAP to produce and/or consume information.

You can get a greater detail on the API's distributed through the Platform by reading the document  (EN) SOFIA2-SOFIA2 APIs.pdf, where its use and installation is explained.

4.3.4 Catching/Exploiting Information

This is part of a KP's business logic and, as such, is independent from the Platform. The specifics to capture information from the different sources (if the KP is an information producer) and to exploit that information once it is received (if the KP is an information consumer) depend only on the KP's nature and purpose.

4.3.5 Translating the information to an ontological format

This document has already explained that the information that an information producer KP sends to the Platform must follow a format that is defined in the ontology representing that information. Thus, that information must be used to build a JSON message grouping the data while following the corresponding ontology's **JSONSchema**. By doing this, the data become an instance of the ontology.

4.3.6 Sending to the Platform using SSAP

Once you have built the JSON message with the data to be sent to the Platform, you must then build the corresponding INSERT message of SSAP, which will integrate the data.

The Platform will validate that the KP's owner user has the corresponding permission on the ontology representing such data, and it will also validate the data as following the ontology's JSONSchema. If there is any problem, the KP will be notified. Otherwise, the data will be added to the SIB's real-time database, and will be available for all the KP's.

The INSERT command, along with all the SSAP commands, is contemplated in every KP's API's. You can get a greater detail on SSAP messaging by reading the document

 (EN) SOFIA2-SOFIA2 APIs.pdf

4.3.7 Query/Subscribe to information using SSAP

The information that the KP's have sent can be consulted by other KP's, either explicitly using the QUERY command from the SSAP, or in subscription mode to future information entries using the SUBSCRIBE operation.

In both cases, the Platform must be notified of the query's criteria.

When using **QUERY**, the KP will receive the existing instances that are currently in the real-time database and that fulfil the query's criteria.

When using **SUBSCRIBE**, the Platform will send the KP any new instance that fulfil the command's criteria as soon as a KP inserts them, but only in the future.

SOFIA allows several kinds of **QUERY** commands:

- **Native Query:** When the query is solved by the underlying RTDB engine, which is *MongoDB* in the reference implementation.

```
db.TemperatureSensor.find().limit(3);
```

- **SQL-Like Query:** When the SOFIA translates the query to the underlying query language.

```
select * from TemperatureSensor limit 3;
```

You can get a greater detail on Query types by reading the document

 (EN) SOFIA2-SOFIA2 APIs.pdf

4.3.8 Receiving ontologically-formatted information

Similar to a KP sending information to the Platform according to an ontology, whenever a KP receives information from the Platform, the information is formatted in JSON following the corresponding ontology. Thus, once the KP has extracted the information from the corresponding SSAP message, the KP can deal with the information following the definition of the ontology, which is defined in the **JSONSchema**.

4.4 Real-Time Collaboration between KP's

This represents several KP's co-operating with each other to create a **SmartSpace**. This requires the following:

- The ontologies representing the information to be exchanged must have been added to the Platform using the **Web Console**.
- The user owners of the several KP's must have been added to the Platform using the **Web Console**. An active token must be associated to each of them.
- The users have the right permissions on the ontologies used by their KP's.
- The KP's (both information **producers** and **consumers**) must have been added to the Platform.
- The **information producers** are operative:
  - They have connected with the Platform using the SSAP message JOIN.
  - They are catching information following their own business logic.
  - They translate the information they catch to the corresponding ontology.
  - They insert their information in the Platform using the SSAP message INSERT.

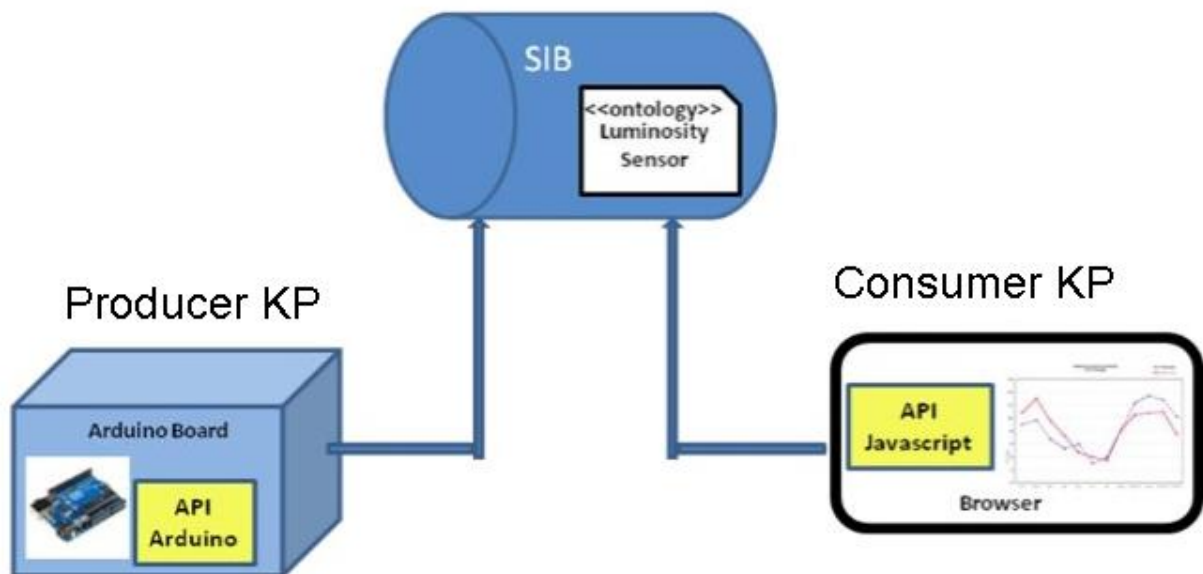
- The **information consumers** are operative:
  - They have connected with the Platform using the SSAP message JOIN.
  - They are catching information from the Platform:
    - Either explicitly using the SSAP's QUERY
    - And/or in subscription mode using the SSAP's SUBSCRIBE.
  - They exploit the information following their own business logic.

5 A PRACTICAL EXAMPLE

We will now follow the previously-described steps through a practical example.

In this example we will develop two KP's which will exchange information coming from a light-sensing device.

Graphically:



Where:

- **Ontology *LuminositySensor*:** It represents, in a JSON-formatted normalized way, the information from the light-sensing device.
- **Producer KP:** This KP runs from an Arduino board. It will use SOFIA's Arduino API. This KP will have a photovoltaic sensor connected to one of the analogical inputs in the board.

The KP will perform a SSAP JOIN operation to open a session with the Platform's SIB. Once connected, the KP will periodically ask for the value of the light sensor's analogical input; and then it will write a SSAP INSERT message to send the SIB an instance of the *LuminositySensor* ontology with that value.

- **Consumer KP:** An HTML page which will be run from a web browser: It will use SOFIA's JavaScript API.

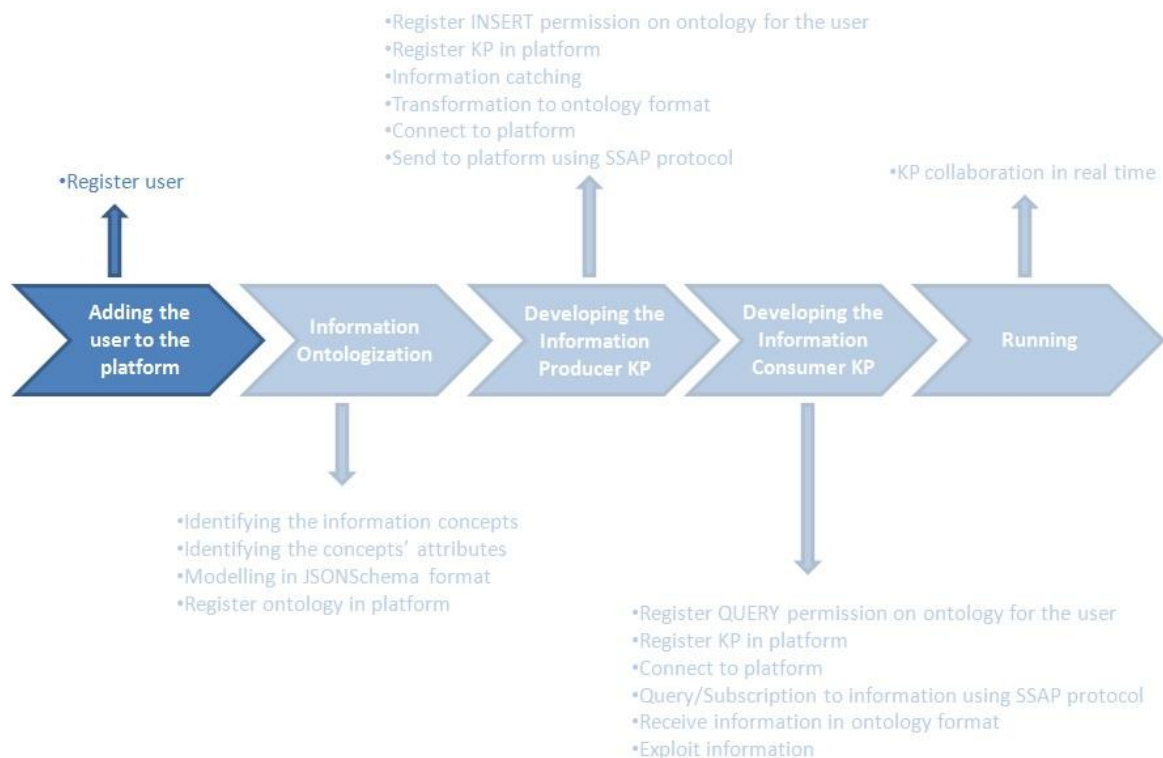
The KP will perform a SSAP JOIN operation to open a session with the Platform's SIB. Once connected, the KP will send the SIB an SSAP SUBSCRIBE operation. From that

point on, the Consumer KP will be notified whenever the Producer KP sends new instances of *LuminositySensor* ontology to the SIB.

Once subscribed, the KP will use the SIB's notification to represent the luminosity's evolution in a chart.

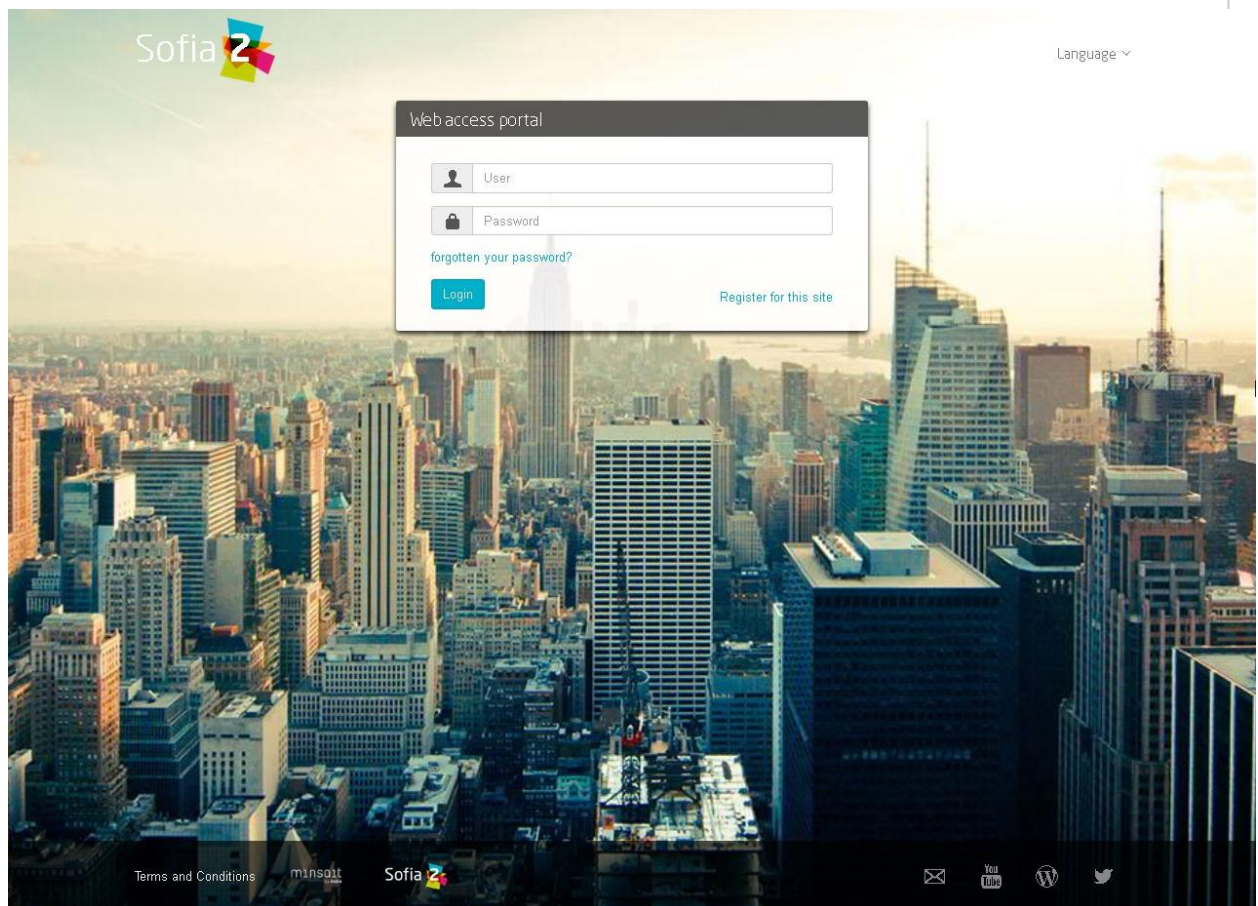
We will now build the described scenario step by step.

5.1 Adding the User to the Platform



We will add only one user to the Platform. This user will be the owner of both KP's. The user will be named *LuminositySensorization*.

We enter the platform and we select **Register for this site**:



We then insert the user's data in the form.

User Registration Request

User

Password

Name

Email

☐ By clicking 'Register', you agree to the [Terms and Conditions](#) and confirm that you read our data policy, including the use of cookies.

Sign in

Back to register

Once the user has been added like this, that user will have the role **USER** meaning that the user can create her own KP's on ontologies to which she is subscribed to, but she is not autonomous enough to run this practical example because it requires the creation of an information producer KP. To get the needed privileges, a different user with the role **ADMINISTRATOR** must boost her Platform-control level to **COLLABORATOR** or to **ADMINISTRATOR**.

For this example, we will give her the role **COLABORATOR** as explained in the

 (EN) SOFIA2-Web Console Use Guide.pdf :



REST API Doc User ▾ Language ▾ Log Off

MANAGEMENT / Modify User

MODIFY EXISTING USER

User

User

LuminositySensorization

Password

Name

Luminosidad S.A.

Email

contacto@luminosidadsa.com

Role

ROL_COLABORADOR

From Date

03/24/2015

To Date

☒ Active

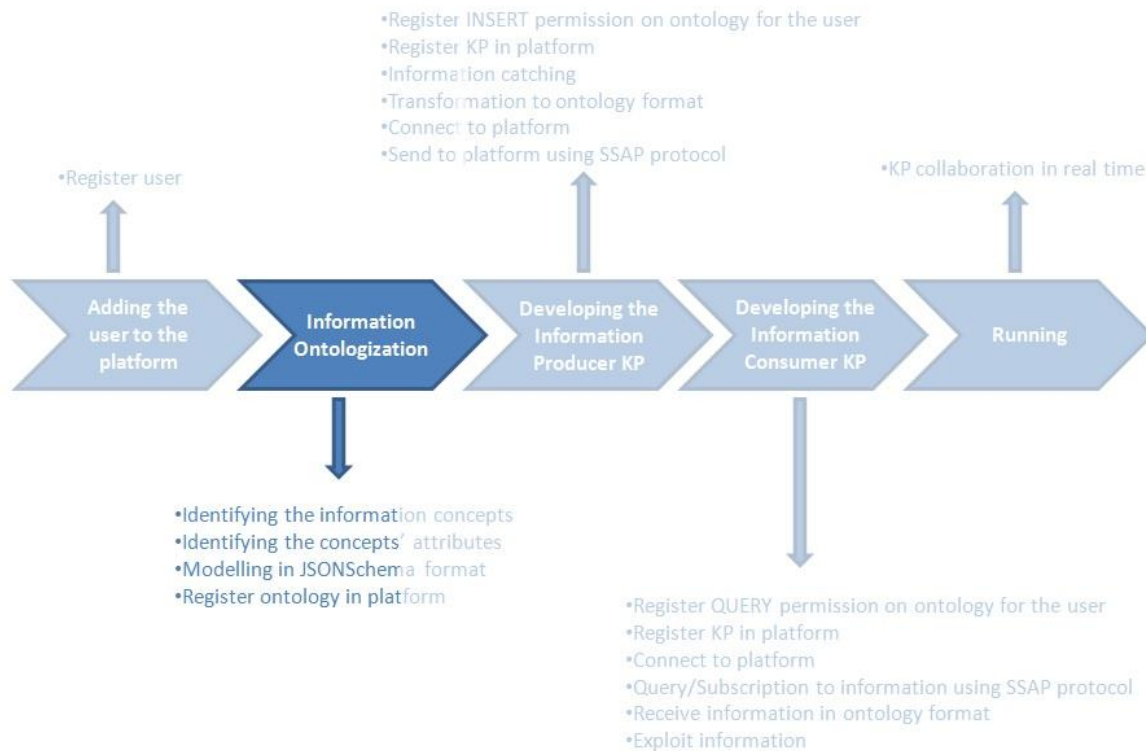
User Token

625dd912da3e4f53926ea29e7815fd63

Cancel Save

Thus, from now on, our user *LuminositySensorization* will be autonomous enough on the Platform to create and deploy her own KP's.

5.2 Information Ontologization



For our scenario, we will identify only one concept: **LuminositySensor**. This concept will have the following attributes:

- **Identifier**: Sensor's Identifier → String
- **Timestamp**: Moment of the measuring process → integer
- **Measurement**: Value of the measurement (in lumens) → integer
- **Unit**: Name of the unit used for the measurement (Lumen) → String

Once those properties are identified, we can build the *JSONSchema* of the ontology describing them:

LuminositySensor.json

```

{
  "$schema": "http://json-schema.org/draft-03/schema#",
  "title": "LuminositySensor Schema",
  "type": "object",
  "properties": {
    "_id": {
      "type": "object",
      "$ref": "#/identifier"
    },
  },
}
  
```

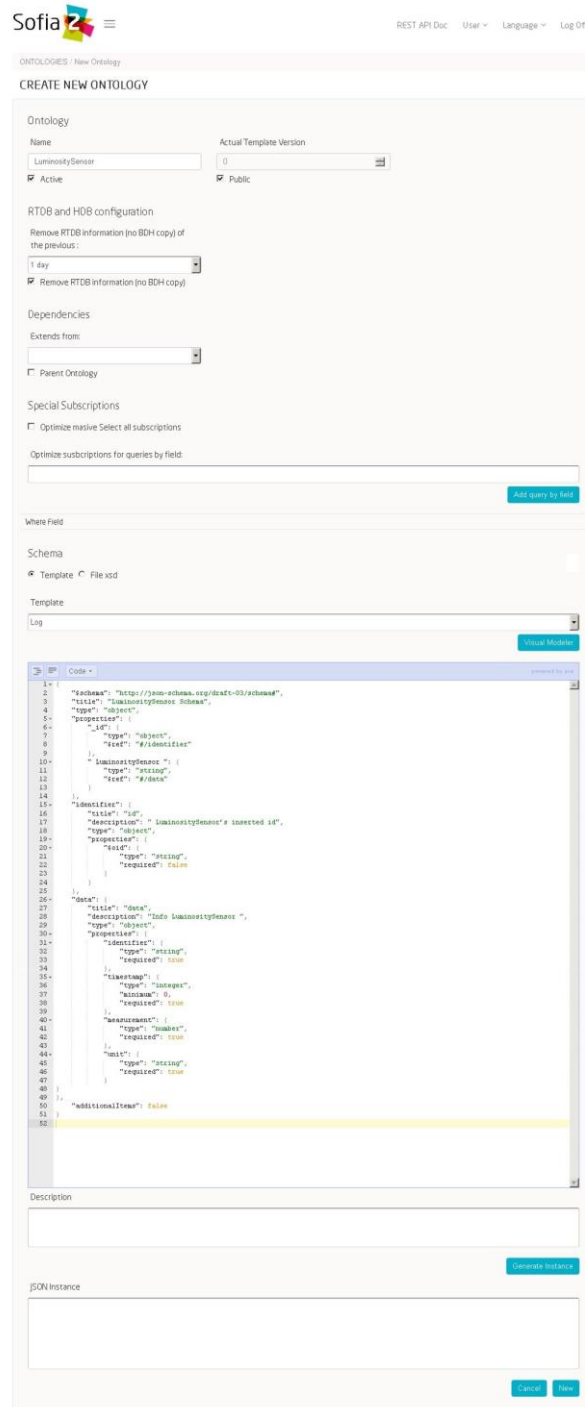
```

    "LuminositySensor ": {
      "type": "string",
      "$ref": "#/data"
    }
  },
  "identifier": {
    "title": "id",
    "description": " LuminositySensor's inserted id",
    "type": "object",
    "properties": {
      "$oid": {
        "type": "string",
        "required": false
      }
    }
  },
  "data": {
    "title": "data",
    "description": "Info LuminositySensor ",
    "type": "object",
    "properties": {
      "identifier": {
        "type": "string",
        "required": true
      },
      "timestamp": {
        "type": "integer",
        "minimum": 0,
        "required": true
      },
      "measurement": {
        "type": "number",
        "required": true
      },
      "unit": {
        "type": "string",
        "required": true
      }
    }
  },
  "additionalItems": false
}

```


We now add the ontology to the Platform using the **Ontology Management** area as described in the  (EN) [SOFIA2-Web Console Use Guide.pdf](#).

To do so, we give the ontology a name, in this case **LuminositySensor**. We then mark the ontology as **Active**, so that the SIB accepts its use. We mark it as Public or not, depending on whether we want it to be used by other users or not. We can also give it a textual description.



CREATE NEW ONTOLOGY

Ontology

Name: Actual Template Version:

☒ Active ☒ Public

RTDB and HDB configuration

Remove RTDB information (no BDI copy) of the previous:

☒ Remove RTDB information (no BDI copy)

Dependencies

Extends from:

Special Subscriptions

☐ Optimize massive Select all subscriptions

Optimize subscriptions for queries by field:

Where field:

Schema

☒ Template ☐ File xsd

Template:

Code

```

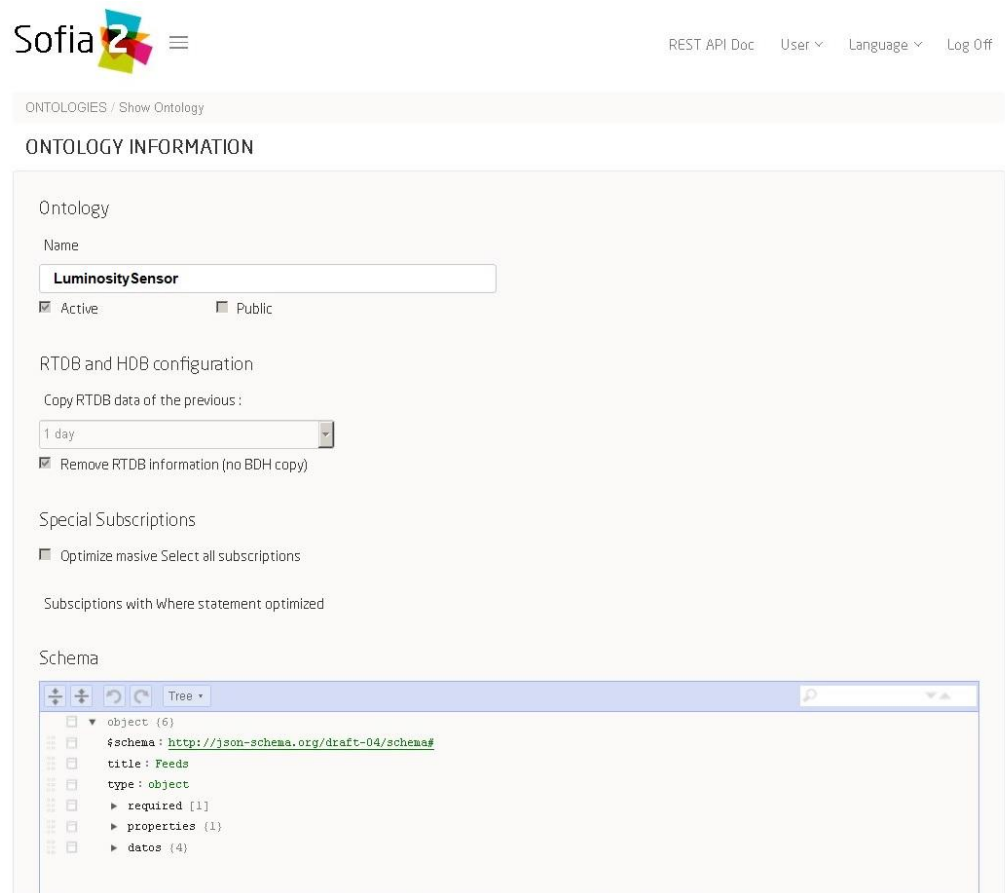
1 {
2   "name": "http://www.indra-robotics.org/indra-03/indradef",
3   "title": "LuminositySensor Schema",
4   "type": "object",
5   "properties": {
6     "id": {
7       "type": "object",
8       "id": "#/id/instance"
9     },
10    "luminositySensor": {
11      "type": "object",
12      "id": "#/data"
13    }
14  },
15  "description": "LuminositySensor's inserted id",
16  "type": "object",
17  "properties": {
18    "id": {
19      "type": "object",
20      "id": "#/id/instance"
21    },
22    "luminositySensor": {
23      "type": "object",
24      "id": "#/data"
25    }
26  },
27  "data": {
28    "title": "Data",
29    "description": "LuminositySensor's inserted id",
30    "type": "object",
31    "properties": {
32      "id": {
33        "type": "object",
34        "id": "#/id/instance"
35      },
36      "luminositySensor": {
37        "type": "object",
38        "id": "#/data"
39      }
40    },
41    "luminositySensor": {
42      "type": "object",
43      "id": "#/data"
44    },
45    "luminositySensor": {
46      "type": "object",
47      "id": "#/data"
48    }
49  },
50  "additionalItems": false
51 }
52

```

Description:

JSON Instance:

Once the ontology has been added, the following screen will be shown and it will be available for future administration in the area **Ontology Management** of the [Web Console](#):



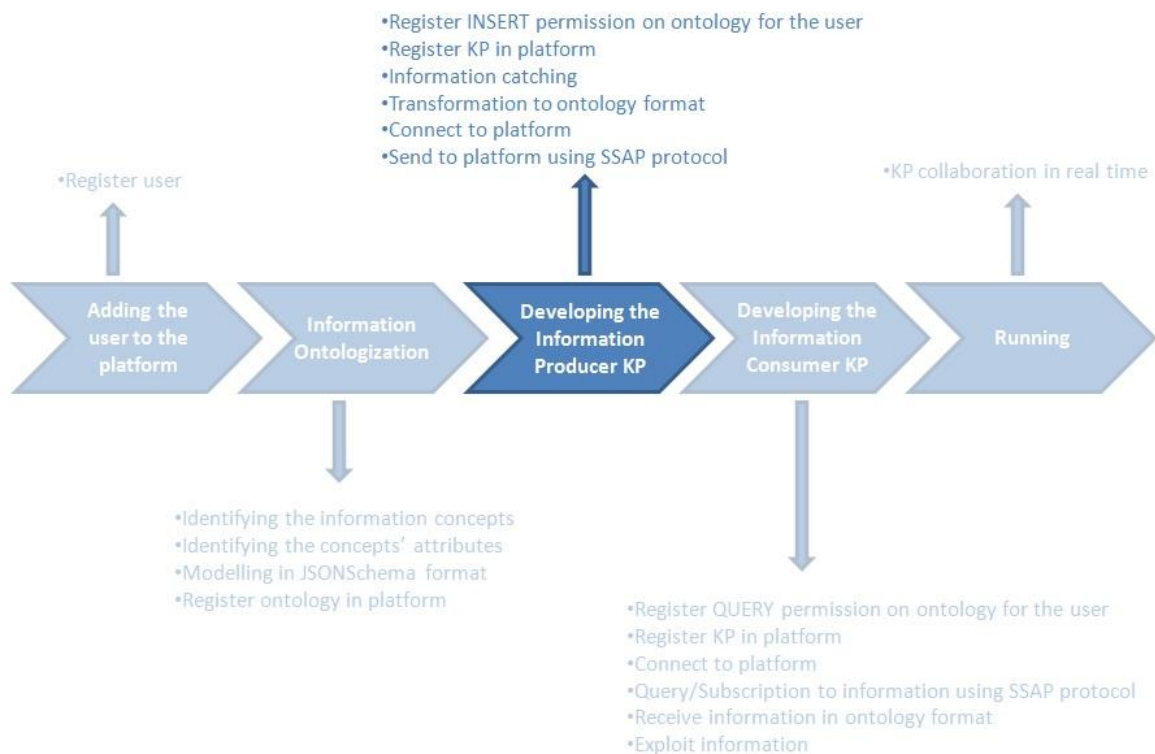
The screenshot shows the 'Sofia 2' web console interface. At the top, there's a navigation bar with the 'Sofia 2' logo, a hamburger menu, and links for 'REST API Doc', 'User', 'Language', and 'Log Off'. Below this is a breadcrumb trail 'ONTOLOGIES / Show Ontology'. The main section is titled 'ONTOLOGY INFORMATION' and contains several configuration options for the 'LuminositySensor' ontology:

- Ontology Name:** A text input field containing 'LuminositySensor'.
- Active:** A checked checkbox.
- Public:** An unchecked checkbox.
- RTDB and HDB configuration:**
  - Copy RTDB data of the previous:** A dropdown menu set to '1 day'.
  - Remove RTDB information (no BDH copy):** A checked checkbox.
- Special Subscriptions:**
  - Optimize masive Select all subscriptions:** An unchecked checkbox.
- Subscriptions with Where statement optimized:** A text input field.
- Schema:** A JSON schema editor showing a tree view of the schema:


```

{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "title": "Feeds",
  "type": "object",
  "required": [
    "required"
  ],
  "properties": {
    "properties": {}
  },
  "datos": {}
}
```

5.3 Developing the Information Producer KP



In the development of our KP, we should start by giving **INSERT** permission to our user for the ontology **LuminositySensor** as explained in the  (EN) SOFIA2-Web Console Use Guide.pdf . In our case, the user owner of this KP is also owner of the ontology that the KP will use. Due to this, she already has all the permissions, and she does not need to be given the INSERT permission.

The next step is adding the KP to the Platform. To do so, we go to the **KP Management** area, once there we create a new KP as explained in the  (EN) SOFIA2-Web Console Use Guide.pdf , with the following information:

KP's/APPS SOFIA2 / New KP

CREATE NEW KP

KP

Identification

LuminosityProducer

Encryption Key

0a9a1e81.8368.4562.88a5-c03fde53af54

Description

Ontology

LuminositySensor

meteostat13

SensorHumedad

SensorTemperatura

Group Ontologies

Meta-Inf

Cancel

New

[Terms and Conditions](#)

[minISO11](#)





Where:

Identification	KP Name
Encryption Key	XXTEA cipher key for coded communications with the SIB
Description	Text describing the KP's purpose
Ontology	Ontologies that the KP will use
Meta-Inf	Value key pairs with additional information about the KP

Once it has been added, it is available to be managed through the **KP Management** area.

KPs/APPS SOFIA2 / My KPs/APPS

MY KPS/APPS

Identification	Description
____	____
Ontologies	Group Ontologies
CompostSensor meteostat13 SensorHumedad SensorTemperatura	____
Search New KP	

EXISTING KPS LIST

Identification	Description	Ontologies	User	Options
LuminosityProducer		LuminositySensor	colaborador	
KPepito		SensorTemperatura	colaborador	
KpScada_jrodriguezv	Kp para la suscripcion y el envío de información desde SCADA	TagMeasures	colaborador	
meteoemplekpgjrv		meteostat13	colaborador	
MonitorTempHumed	To see ontologies SensorHumedad & SensorTemperatura	SensorTemperatura SensorHumedad	colaborador	

Showing 1 to 5 of 5 entries

First Previous 1 Next Last

Terms and Conditions

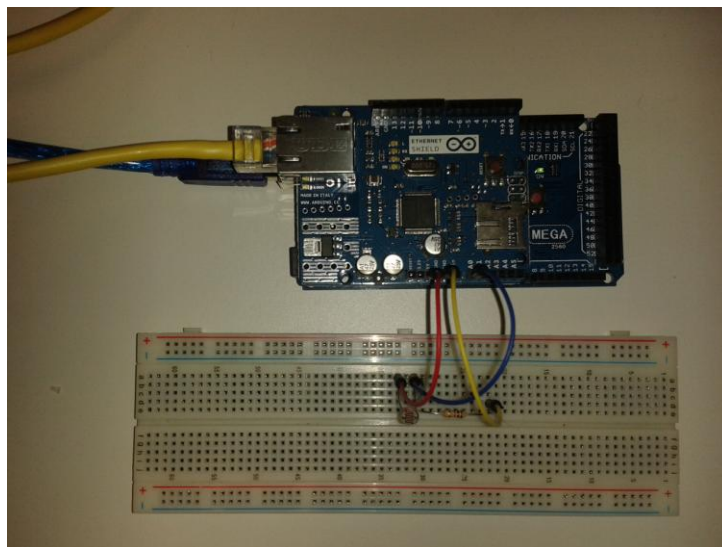
minisql

Sofia 2



The following step in the development of our information producer KP is programming it. To get more information on how to develop a KP, please refer to [\(EN\) SOFIA2-SOFIA2 APIs.pdf](#).

As our producer KP is an application which will run on an **Arduino** board, it will be developed in Arduino's language.



Arduino is a **microcontroller** and its programming logic revolves around the following two functions:

- **Setup():** It is invoked only once when the microcontroller is started. Its goal is performing configuration activities.
- **Loop():** Iteratively invoked (Whenever it ends, it is re-invoked). It covers the microcontroller's business logic.

Following this programming philosophy, we open a **session to the SIB** of the Platform using the SSAP message **JOIN**. To do this, we will use the Arduino API. This API provides us with the following utilities:

- **MQTT KP Connector** (KPMqtt.h) to create a physical connection with the Platform's SIB. This connection will be used to send the SSAP messages through it.
- **API to generate and to parse SSAP messages** without having to build the JSON messages directly (SSAPMessageGenerator.h and SSAPBodyReturnMessage.h).

With this utility, we can establish the physical connection through MQTT and later the logical connection through the SSAP message JOIN. This will be done following these steps:

```
if(!kp.isConnected()){
    kp.connect();
}

Serial.println("Send join");
joinMessage=ssapGenerator.generateJoinMessage("sensorizacionLuminosidad", "slum2012", "ProductorLuminosidad:prod01");

SSAPMessage joinResponse=kp.send(joinMessage);

if(joinResponse!=NULL){

    char* bodyJoin=joinMessage.getBody();
    delete[] bodyJoin;

    char* responseBody=joinResponse.getBody();

    SSAPBodyReturnMessage bodyMessage=SSAPBodyReturnMessage::fromJsonToSSAPMessage(responseBody);

    if(bodyMessage.isOk()){
        joined=true;
    }else{
        joined=false;
    }
}
```

Once we have connected to the Platform, the next step is catching data, then ontologizing them, then inserting them in the SIB using the corresponding INSERT message.

Our KP can catch data by reading the photovoltaic sensor's analogical input in every iteration of the Loop() function:

```
//meter Luminosity
int lightRead = analogRead( photoSensorPinIn );
```

Once the data is catch, we must translate them to the **LuminositySensor** ontology before sending them to the Platform.

We have two choices here:

- We can build a *LuminositySensor* class with *get* and *set* methods for its attributes, and a *toJson()* function that will return the instance's JSON.
- We can build the JSON directly as a String, concatenating the measurement's value.

As this is a very simple ontology, we resort to the second choice:

```
strcpy(sensorJson, "{\"LuminositySensor\":{\"identificador\":\"Luminosity001\", \"timestamp\" : 1 , \"medida\" :\"});
strcat(sensorJson, strLightRead);
strcat(sensorJson, \" , \"unidad\" : \"lumen\"}");
```

Once the ontology's instance has been built with the value, we have to send it to the Platform using the corresponding SSAP message **INSERT**. To do this, we use the Arduino API again:

```
//Generates new SSAP INSERT message
insertMessage = ssapGenerator.generateInsertMessage(kp.getSessionKey(), "LuminositySensor", sensorJson);
//Send the message to the SIB
SSAPMessage insertResponse=kp.send(&insertMessage);

char* responseBody=insertResponse.getBody();
```

The whole source code for the producer KP can be seen here:

KP_LuminositySensor.ino

```
#include <SSAPMessageGenerator.h>
#include <SSAPBodyReturnMessage.h>
#include <SPI.h>
#include <PubSubClient.h>
#include <Ethernet.h>
#include <KPMqtt.h>
#include <LightSensor.h>

byte mac[] = { 0x00, 0xAA, 0xBB, 0xCC, 0xDE, 0x01 };
IPAddress ip( 192, 168, 10, 129 ); // My Ip
byte serverIp[] = { 192, 168, 10, 128 }; // Server Ip

boolean joined=false;
ConnectionConfig config;
```



```
KPMqtt kp;
SSAPMessageGenerator ssapGenerator;

int photoSensorPinIn = 0;//A0 input3

void setup() {
  Serial.begin( 9600 );

  config.setLocalMac(mac);
  config.setLocalIp(&ip);
  config.setServerIp(serverIp);

  kp.setClientId("Arduino");
  kp.setConnectionConfig(&config);
}

SSAPMessage joinMessage;

void loop() {
  Serial.println("new Loop");
  if(!joined){
    join();
  }

  if(joined){
    meterValues();
  }

  Serial.println("End of Loop");
  delay(1000);
}

void join(){
  if(!kp.isConnected()){
    Serial.println("connect");
    kp.connect();
  }

  Serial.println("Send join");
  joinMessage=ssapGenerator.generateJoinMessage("sensorizacionLuminosidad", "slum2013", "ProductorLuminosidad:prod01");

  SSAPMessage joinResponse=kp.send(&joinMessage);
  if(&joinResponse!=NULL){
    char* bodyJoin=joinMessage.getBody();
  }
}
```

```

delete[] bodyJoin;

char* responseBody=joinResponse.getBody();

SSAPBodyReturnMessage bodyMessage=SSAPBodyReturnMessage::fromJsonToSSAPMessage(responseBody);

if(bodyMessage.isOk()){
    joined=true;
}else{
    joined=false;
}

delete[] bodyMessage.getData();
delete[] bodyMessage.getError();
delete[] responseBody;
delete[] joinResponse.getSessionKey();
delete[] joinResponse.getMessageId();
delete[] joinResponse.getOntology();
}
}

char sensorJson[100];
char strLightRead[4];

SSAPMessage insertMessage;
void meterValues(){
    Serial.println("MeterValues");

    //meter Luminosity
    int lightRead = analogRead( photoSensorPinIn );

    //Generates Luminosity sensor JSon
    itoa(lightRead, strLightRead, 10);

    strcpy(sensorJson, "{\"LuminositySensor\":{\"Identifier\":\"Luminosity001\", \"timestamp\" : 1 , \"Measurement\" :\"");
    strcat(sensorJson, strLightRead);
    strcat(sensorJson, " , \"unit\" : \"lumen\"}"}");

    Serial.print("Instancia LuminositySensor enviada: ");
    Serial.println(sensorJson);

    //Generates new SSAP INSERT message
    insertMessage = ssapGenerator.generateInsertMessage(kp.getSessionKey(), "LuminositySensor", sensorJson);
    //Send the message to the SIB
    SSAPMessage insertResponse=kp.send(&insertMessage);

```

```

char* responseBody=insertResponse.getBody();

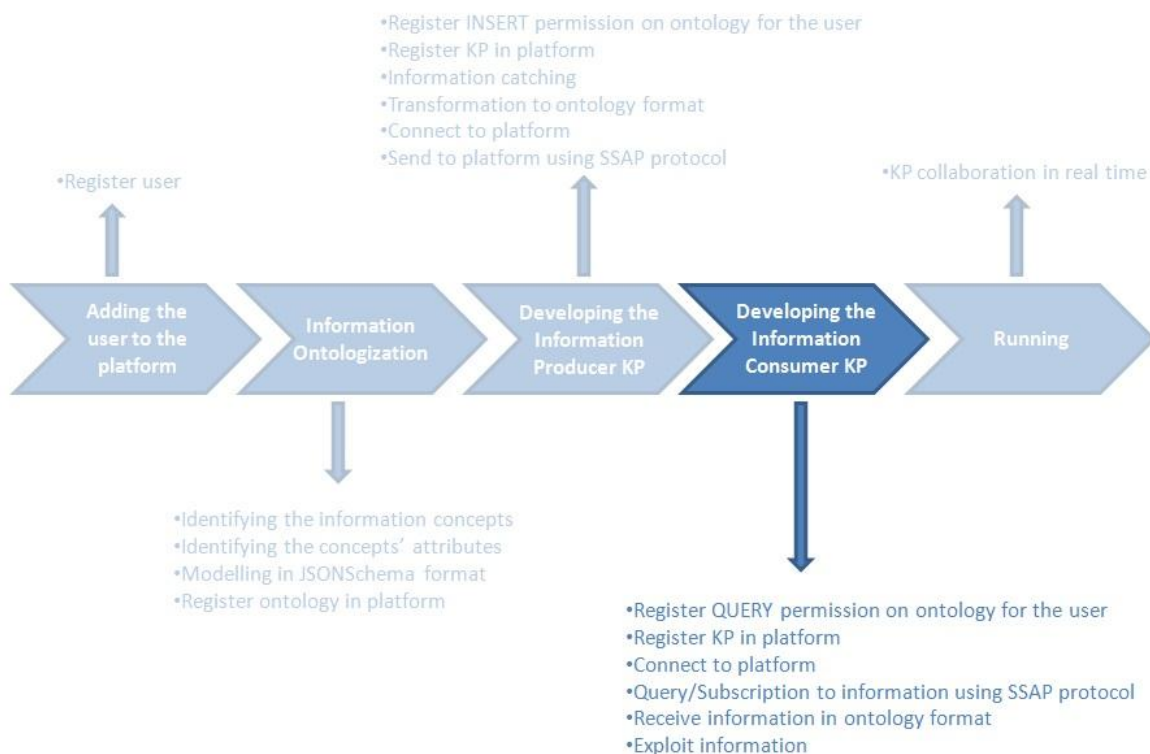
//Free responseBody char*
delete[] responseBody;

//Free json char* memory
//delete[] sensorJson;
//delete[] jsonPrefix;
// delete[] strLightRead;
//delete[] jsonSufix;
delete[] insertMessage.getSessionKey();
delete[] insertMessage.getBody();
delete[] insertMessage.getOntology();

delete[] insertResponse.getSessionKey();
delete[] insertResponse.getMessageId();
delete[] insertResponse.getOntology();
}

```

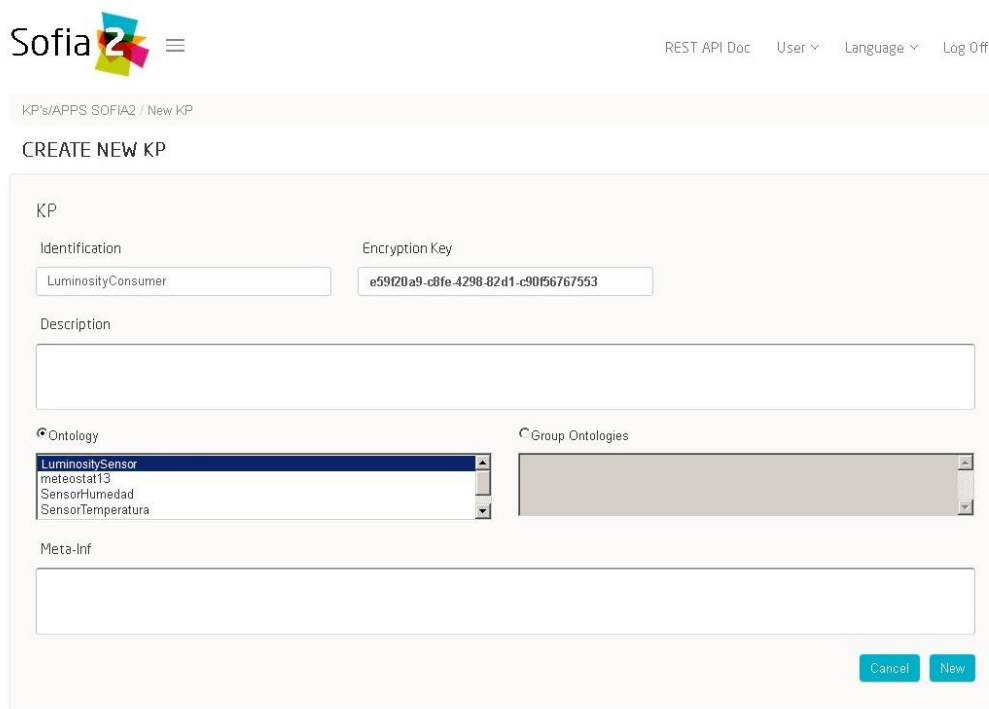
5.4 Developing the information consumer KP



To develop our information consumer KP we must start the same way as when developing our information producer KP: We will start by giving **QUERY** permissions to

our user on the **LuminositySensor** ontology. Again, the KP owner is also owner of the ontology that the KP will use, and thus she does not need to receive that permission because she already has all the needed permissions. If we were using another ontology that the user does not own, then an administrator would need to give permissions to that user as explained in section 3.3.1 [Joining the platform](#)

The next step is adding the KP to the Platform. To do so, we go to the section **KP Management** area and then we create a new KP with the following information:



Sofia 2 ≡ REST API Doc User ▾ Language ▾ Log Off

KP's/APPS SOFIA2 / New KP

CREATE NEW KP

KP

Identification: Encryption Key:

Description:

Ontology:

Group Ontologies:

Meta-Inf:

Where:

Identification	KP Name
Encryption Key	XXTEA cipher key for coded communications with the SIB
Description	Text describing the KP's purpose
Ontology	Ontologies that the KP will use
Meta-Inf	Value key pairs with additional information about the KP

Once the KP has been added, it is available for future management using the **KP Management** area. Now we have both the Consumer KP and the Producer KP, which we added in a previous step, in that area:

The screenshot shows the Sofia 2 web interface. At the top, there's a header with the Sofia 2 logo and navigation links: REST API Doc, User, Language, and Log Off. Below the header, the main content area is titled 'MY KPS/APPS'. It contains a form for adding new KPs with fields for Identification, Description, Ontologies, and Group Ontologies. Below the form is a table titled 'EXISTING KPS LIST' showing a list of KPs with columns for Identification, Description, Ontologies, User, and Options. The table lists five KPs: LuminosityProducer, LuminosityConsumer, KpScada_jrodriguez, meteoemplotkpirv, and MonitorTempHumed. At the bottom of the page, there's a footer with links to Terms and Conditions, minsoit, and Sofia 2, along with social media icons for email, YouTube, GitHub, and Twitter.

The next step when developing our information consumer KP is programming it. For more information on how to develop a KP, please refer to [\(EN\) SOFIA2-SOFIA2 APIs.pdf](#).

As this KP will be run from a web browser, it will be developed in **HTML** and **JavaScript**, and due to this we will use SOFIA's JavaScript API. This API provides us with the needed abstraction to interact with SOFIA's SIB through functions for any operation in the SSAP:

- **join(user, pass, instance, joinResponse):** Sends a SSAP JOIN message to the SIB, including the user, password and instance specified in the parameters. It receives its response in the callback function that is given in the joinResponse argument.
- **leave(leaveResponse):** Sends a SSAP LEAVE message to the SIB. It receives its response in the callback function given in the leaveResponse argument.
- **insert(data, ontology, insertResponse):** Sends a SSAP INSERT message to the SIB. The arguments include the data and the ontology, which will be sent. It

receives the response in the callback function given in the `insertResponse` argument.

- **query(query, ontology, queryResponse)**: Sends a SSAP QUERY message to the SIB. The QUERY is for the condition and ontology specified in the parameters. It receives the response in the callback function given in the `queryResponse` argument.
- **subscribe(suscription, ontology, refresh)**: It sends a SSAP SUBSCRIBE message to the SIB. The arguments specify the condition, ontology and refresh time for the subscription.
- **unsubscribe(querySubs, unsubscribeResponse, unsubscribeMessages)**: It sends a SSAP UNSUBSCRIBE message to the SIB. The parameter `querySubs` specifies the condition of the UNSUBSCRIBE. It receives its response in the callback function given in the `unsubscribeResponse` argument. If there is any error, it will be notified through the callback function given in the `unsubscribeMessages` argument.

Our HTML KP is made of:

- A form to perform JOIN/LEAVE to the SIB
  - Textfield user name.
  - Textfield user's password.
  - Textfield KP instance.
  - Button JOIN.
  - Button LEAVE.
- A form to perform SUBSCRIBE/UNSUBSCRIBE on the *LuminositySensor* ontology.
  - Textfield subscription's condition.
  - Textfield refresh time.
  - Button SUBSCRIBE.
  - Button UNSUBSCRIBE.
- A chart showing the luminosity data evolution, reflecting the information received as subscription notifications.

KP Visualización

Conectar / Desconectar al SIB

Token: 3ab58085dbf84a11843cefb143b327eb

Instance: KpvisualizacionLum:KpvisualizacionLum01

Conectar SIB

Desconectar SIB

Suscripción

Suscripción: L lm >

SensorLuminosidad (lm): 1

Refresco (ms): 1000

Suscribir

Desuscribir

Luminosidad en Tiempo Real

Luminosity / Real Time

Luminosity (lm)

— Luminosity

As we have suggested, the connection and disconnection to the SIB is performed using the buttons that the previous image shows. The HTML code is as follows:

```

<table>
  <tr style="border:none;">
    <td align="right" style="border:none;">
      <b>User:</b>
      <input type="text" id="user" size="10" style="margin:8px" value="luminosidadweb"></input>
    </td>
    <td align="center" style="border:none;">
      <b>Password:</b>
      <input type="text" id="pass" size="10" style="margin:8px" value="luminosidadweb"></input>
    </td>
    <td align="Left" style="border:none;">
      <b>Instance:</b>
      <input type="text" id="inst" size="10" style="margin:8px" value="luminosidadweb:luminosidadweb01"></input>
    </td>
  </tr>
  <tr style="border:none;">
    <td colspan="2" align="center" style="border:none;">
      <input type="button" size="40" style="margin:4px" value="Connect SIB" onclick='conectarSIB($("#user").val(), $("#pass").val(), $("#inst").val());' ></input>
      <input type="button" size="40" style="margin:4px" value="Disconnect SIB" onclick='desconectarSIB()'></input>
    </td>
  </tr>
</table>

```

We will now show the **conectarSIB** (“connect to SIB”) and **desconectarSIB** (“disconnect from SIB”) functions. Those are invoked by the buttons and make use of SOFIA’s JavaScript API.



```

function conectarSIB(user, pass, inst) {
  join(user, pass, inst, function(mensajeSSAP){
    if(mensajeSSAP != null && mensajeSSAP.body.data != null && mensajeSSAP.body.ok == true){
      $('#info').text("Conectado al sib con sessionkey: "+mensajeSSAP.sessionKey).show();
    }else{
      $('#info').text("Error conectando del sib").show();
    }
  });
}

function desconectarSIB() {
  leave(function(mensajeSSAP){
    if(mensajeSSAP != null && mensajeSSAP.body.data != null && mensajeSSAP.body.ok == true){
      $('#info').text("Desconectado del sib").show();
    }else{
      $('#info').text("Error desconectando del sib").show();
    }
  });
}

```

Now we will talk about the producer KP subscribing to information inserted in the SIB. This is done in the same way, using a subscription form. That form's HTML code is as follows:

```

<table>
<tr style="border:none;">
<td style="border:none;">
<b>Suscripción:</b>
<select id="suscripcion" style="margin:4px">
<option value="gt" selected="selected">L lm &gt;</option>
<option value="lt">L lm &lt;</option>
<option value="=">L lm &#61;</option>
</select>
</td>
<td style="border:none;">
<b>LuminositySensor (lm): </b>
<input type="text" id="valor" size="2" value="1" style="margin:8px" />
</td>
<td style="border:none;">
<b>Refreshment (ms): </b>
<input type="text" id="refresco" size="5" style="margin:8px" value="1000"/>
</td>
</tr>
<tr style="border:none;">
<td colspan="3" align="center" style="border:none;">
<input type="button" value="Start" style="margin:4px" onclick='suscribirSIB($("#suscripcion").val(),$("#valor").val(),$("#refresco").val());'/>
<input type="button" value="Stop" style="margin:4px" onclick='desuscribirSIB($("#suscripcion").val(),$("#valor").val());'/>
</td>
</tr>
<tr style="border:none;">
<td colspan="3" align="center" style="border:none;">
<b id="info" style="color:Crimson;margin-top:10px;display:none;" />
</td>
</tr>
</table>

```

This form has a combo box to select whether we want to subscribe to a luminosity value greater than, less than or equalling a second value which is given in another field. It also includes a new field to specify the subscription's refresh time, and the buttons to launch both the subscription and the unsubscription.

We will now show the **suscribirSIB()** ("Subscribe to SIB") and **desuscribirSIB** ("unsubscribe to SIB") functions that the buttons invoke:





```
function suscribirSIB(suscripcion, valor, refresco) {
  suscripcion = suscripcion.replace(/:/,valor).replace(/>/,"{$gt:"+valor+"}").replace(/</,"{$lt:"+valor+"}");
  var queryMongo = "{LuminositySensor.medida:"+suscripcion+"}";
  var subscriptionNotExists=subscribe(queryMongo, "LuminositySensor", refresco);

  if(!subscriptionNotExists){
    $("#info").text("Ya existe una suscripcion para esa query").show();
  }
}

function desuscribirSIB(suscripcion, valor) {
  suscripcion = suscripcion.replace(/:/,valor).replace(/>/,"{$gt:"+valor+"}").replace(/</,"{$lt:"+valor+"}");
  var queryMongo = "{LuminositySensor.medida:"+suscripcion+"}";

  unsubscribe(queryMongo,
    function(mensajeSSAP){
      if(mensajeSSAP != null && mensajeSSAP.body.data != null && mensajeSSAP.body.ok == true){
        $("#info").text("Desuscrito de "+queryMongo).show();
      }else{
        $("#info").text("Error desuscribiendo del sib").show();
      }
    },
    function(error){
      if(error=="ERROR_1" || error=="ERROR_2"){
        $("#info").text("No existe suscripcion para la query").show();
      }
    }
  );
}
}
```

Finally, once we are subscribed to the information, we need to implement the reception for the notifications sent from the SIB before we can exploit that information.

To do this, the JavaScript API forces us to implement the following function:

function indicationForSubscription(ssapMessageJson, sourceQuery)

In it we receive an SSAP message which is INDICATION. Its **body** attribute includes the instance of the ontology which we have subscribed to and which fulfils the query's criteria. Now the only things we have to do is extracting this information from the body field and process it to draw the chart:

```
// Recepción de notificaciones de la suscripción:
// 1) extraer datos luminosidad
function indicationForSubscription(ssapMessageJson, sourceQuery) {
  var mensajeSSAP = parsearMensajeSSAP(ssapMessageJson);
  if (mensajeSSAP != null){
    try{
      // 1) Cogemos solo el primer mensaje de la notificación
      var medida = mensajeSSAP.body.data[0].LuminositySensor.medida;

      pintarDatosGrafica(medida);
    }catch(err){
    }
  }
}

function pintarDatosGrafica(temp) {
  var x = new Date();
  datosTH.push([x,temp]);
  // scroll datos gráfica en escala de tiempos de 10s
  grafica.updateOptions({'file': datosTH, 'dateWindow': [x.getTime() - 10 * 10000, x.getTime()]});
}
```

The whole code of the Consumer KP can be seen here:





KP_LuminosityJavascript.html

```

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>KP Consumidor</title>

    <link rel="stylesheet" type="text/css" href="styles/standard.css" />
    <script src="jquery.min.js" type="text/javascript"></script>
    <script type="text/javascript"> var pathToDwrServlet =
'http://localhost:10000/sib-web/dwr';</script>
    <script type='text/javascript' src='http://localhost:10000/sib-
web/dwr/engine.js'></script>
    <script type='text/javascript' src='http://localhost:10000/sib-
web/dwr/util.js'></script>
    <script type='text/javascript' src='http://localhost:10000/sib-
web/dwr/interface/GatewayDWR.js'></script>
    <script type='text/javascript' src="kp-core.js"></script>
    <script type='text/javascript' src="dygraph-combined.js"></script>

    <script type="text/javascript">
var datosTH = [];
var grafica = null, map = null;

$(function(){
  dwr.engine.setActiveReverseAjax(true);
  dwr.engine.setErrorHandler(errorHandler);
  dwr.engine.setTimeout(0);
  initGrafica();
});

function errorHandler(message, ex){
}

function initGrafica() {
  grafica = new Dygraph(
    document.getElementById("grafica"),
    [[0,0]],
    {
      title: 'Luminosity / Real Time',
      legend: 'always',
      labels: ['Time', 'Luminosity'],
      ylabel: 'Luminosity (lm)',
      yAxisLabelWidth: 50,
      digitsAfterDecimal : 0,
      drawXGrid: false,
      drawYGrid: false,
    }
  );
}

function conectarSIB(user, pass, inst) {
  join(user, pass, inst,function(mensajeSSAP){
    if(mensajeSSAP != null && mensajeSSAP.body.data != null &&

```





```

mensajeSSAP.body.ok == true){
    $('#info').text("Conectado al sib con sessionkey:
"+mensajeSSAP.sessionKey).show();
}else{
    $('#info').text("Error conectando del sib").show();
}
});
}

function desconectarSIB() {
    leave(function(mensajeSSAP){
        if(mensajeSSAP != null && mensajeSSAP.body.data != null &&
mensajeSSAP.body.ok == true){
            $("#info").text("Desconectado del sib").show();

        }else{
            $("#info").text("Error desconectando del sib").show();
        }
    });
}

function suscribirSIB(suscripcion, valor, refresco) {
    suscripcion =
suscripcion.replace(/:/,valor).replace(/gt/, "{$gt:"+valor+"}").replace(/lt/, "{$lt:"+v
alor+"}");
    var queryMongo = "{LuminositySensor.measurement:"+suscripcion+"}";
    var subscriptionNotExists=subscribe(queryMongo, "LuminositySensor", refresco);

    if(!subscriptionNotExists){
        $("#info").text("Ya existe una suscripcion para esa query").show();
    }

}

function desuscribirSIB(suscripcion, valor) {
    suscripcion =
suscripcion.replace(/:/,valor).replace(/gt/, "{$gt:"+valor+"}").replace(/lt/, "{$lt:"+v
alor+"}");
    var queryMongo = "{LuminositySensor.measurement:"+suscripcion+"}";

    unsubscribe(queryMongo,
        function(mensajeSSAP){
            if(mensajeSSAP != null && mensajeSSAP.body.data != null &&
mensajeSSAP.body.ok == true){
                $("#info").text("Desuscrito de "+queryMongo).show();
            }else{
                $("#info").text("Error desuscribiendo del sib").show();
            }
        },
        function(error){
            if(error=="ERROR_1" || error=="ERROR_2"){
                $("#info").text("No existe suscripcion para la query").show();
            }
        }
    ));
}

```





```

}

//A implementar porque el API la necesita para notificar que la suscripcion se ha
hecho adecuadamente
function subscriptionWellLaunchedResponse(subscriptionId, subscriptionQuery){
    $("#info").text("Suscrito con id: "+subscriptionId+" a query:
"+subscriptionQuery).show();
}

// Recepci3n de notificaciones de la suscripci3n:
// 1) extraer datos temp
function indicationForSubscription(ssapMessageJson, sourceQuery) {
    var mensajeSSAP = parsearMensajeSSAP(ssapMessageJson);
    if (mensajeSSAP != null){

        try{
            // 1) Cogemos solo el primer mensaje de la notificaci3n
            var measurement = mensajeSSAP.body.data[0].LuminositySensor.measurement;
            pintarDatosGrafica(measurement);

        }catch(err){
        }

    }

}

function pintarDatosGrafica(temp) {
    var x = new Date();
    datosTH.push([x,temp]);
    // scroll datos gr3fica en escala de tiempos de 10s
    grafica.updateOptions({'file': datosTH, 'dateWindow': [x.getTime() - 10 * 10000,
x.getTime()]});
}

// Devuelve un mensaje SSAP JSON parseado a un objeto JavaScript
function parsearMensajeSSAP(mensaje) {
    try{
        return $.parseJSON(validarSSAP(mensaje));
    }catch(e){
        //alert ("Error parseo mensaje: " + e);
        return null;
    }
}

// Devuelve un string JSON SSAP v3lido
function validarSSAP(datos) {
    return datos.replace(/\\+\\/g, "\\")
        .replace(/(body|data)\\\"s*:\\s*\\\"({|\\[\\]/g, \"$1\\\":$2\")
        .replace(/(\\{|\\}|)\\\"s*:\\s*\\\"(direction|ontology|message|session|error|ok)/g, \"$1,\\\"$2\");
}

</script>
<style>
h4 {color:DarkCyan;}
.dygraph-legend {
    margin-right: 4px !important;
    top: 12px !important;
    text-align: right !important;
}

```



```

        font-size: 1em !important;
    }

    .dygraph-ylabel {
        margin: 30px !important;
        padding: 0 !important;
        text-align: left !important;
    }

    .dygraph-y2label {
        text-align: left !important;
    }

    </style>
    <head>
        <body>
            
            <h4>Connect / Disconnect to SIB </h4>
            <table>
                <tr style="border:none;">
                    <td align="right" style="border:none;">
                        <b>User:</b>
                        <input type="text" id="user" size="10" style="margin:8px"
value="sensorizacionLuminosidad"></input>
                    </td>
                    <td align="center" style="border:none;">
                        <b>Password:</b>
                        <input type="text" id="pass" size="10" style="margin:8px"
value="slum2013"></input>
                    </td>
                    <td align="left" style="border:none;">
                        <b>Instance:</b>
                        <input type="text" id="inst" size="10" style="margin:8px"
value="ConsumidorLuminosidad:Clm01"></input>
                    </td>
                </tr>
                <tr style="border:none;">
                    <td colspan="3" align="center" style="border:none;">
                        <input type="button" size="40" style="margin:4px" value="Connect
SIB" onclick='conectarSIB($("#user").val(), $("#pass").val(), ($("#inst").val());'
></input>
                        <input type="button" size="40" style="margin:4px"
value="Disconnect SIB" onclick='desconectarSIB()'></input>
                    </td>
                </tr>
            </table>
            <h4>Subscription</h4>
            <table>
                <tr style="border:none;">
                    <td style="border:none;">
                        <b>Subscription:</b>
                        <select id="suscripcion" style="margin:4px">
                            <option value="gt" selected="selected">L lm
                        &gt;</option>
                            <option value="lt">L lm &lt;</option>
                            <option value=".">L lm &#61;</option>
                        </select>

```

```

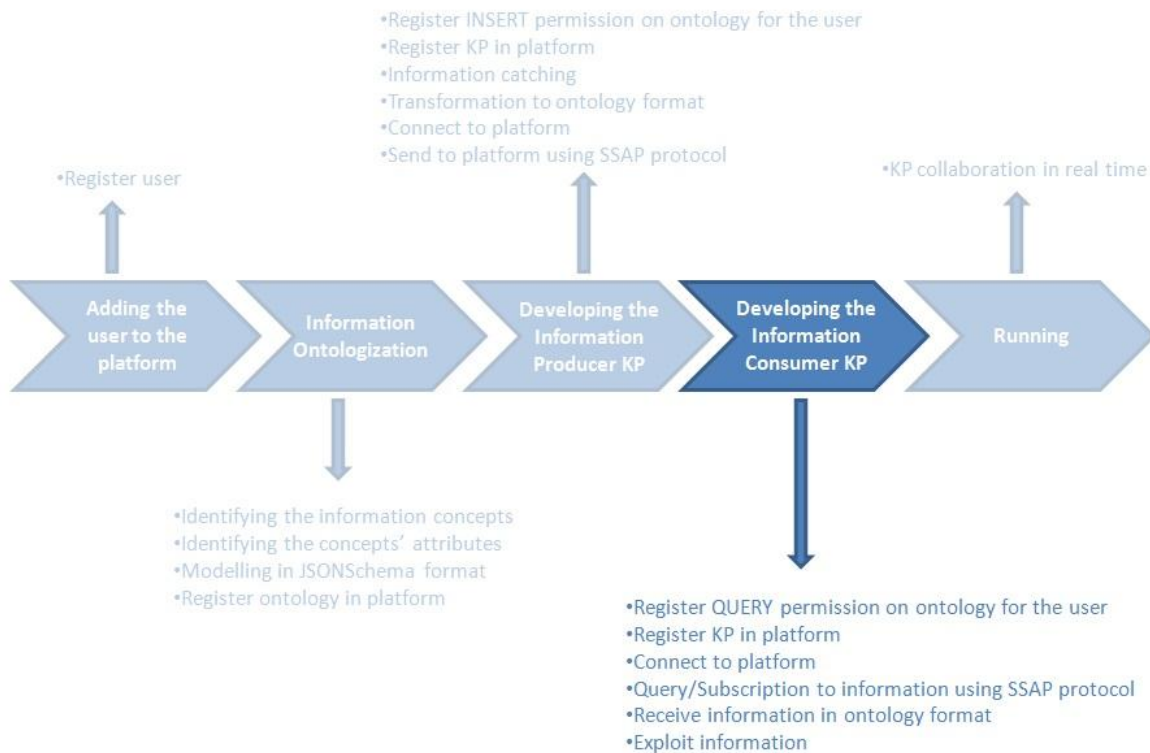
        </td>
        <td style="border:none;">
            <b>LuminositySensor (lm): </b>
            <input type="text" id="valor" size="2" value="1"
style="margin:8px" />
        </td>
        <td style="border:none;">
            <b>Refreshment (ms): </b>
            <input type="text" id="refresco" size="5" style="margin:8px"
value="1000"/>
        </td>
    </tr>
    <tr style="border:none;">
        <td colspan="3" align="center" style="border:none;">
            <input type="button" value="Start" style="margin:4px"
onclick='suscribirSIB($("#suscripcion").val(),$("#valor").val(),$("#refresco").val())
;'/>
            <input type="button" value="Stop" style="margin:4px"
onclick='desuscribirSIB($("#suscripcion").val(),$("#valor").val());'/>
        </td>
    </tr>
    <tr style="border:none;">
        <td colspan="3" align="center" style="border:none;">
            <b id="info" style="color:Crimson;margin-top:10px;display:none;" />
        </td>
    </tr>
</table>
<h4>Real time luminosity</h4>
<table>
    <tr>
        <td style="padding-right:2px;">
            <div id="grafica" style="width:98%;height:200px;margin:2px;"></div></td>
    </tr>
</table>

</body>
</html>

```

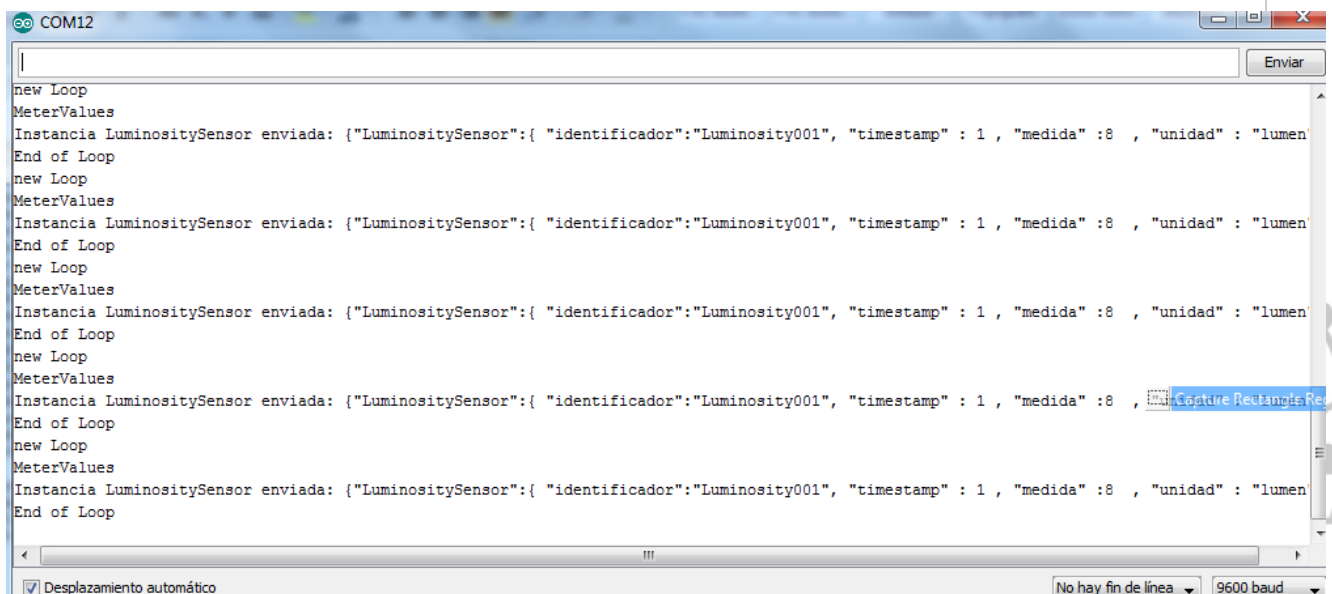


5.5 Running



Once all the previous steps have been performed, the next point in our agenda is checking that it all works correctly. To do so:

- Run the SOFIA Platform
- Load the Producer KP program in Arduino and run it.
  - This KP self-connects to the SIB when running and starts sending data. You do not need to do anything else.



```

COM12
new Loop
MeterValues
Instancia LuminositySensor enviada: {"LuminositySensor":{"identificador":"Luminosity001", "timestamp" : 1 , "medida" :8 , "unidad" : "lumen"
End of Loop
new Loop
MeterValues
Instancia LuminositySensor enviada: {"LuminositySensor":{"identificador":"Luminosity001", "timestamp" : 1 , "medida" :8 , "unidad" : "lumen"
End of Loop
new Loop
MeterValues
Instancia LuminositySensor enviada: {"LuminositySensor":{"identificador":"Luminosity001", "timestamp" : 1 , "medida" :8 , "unidad" : "lumen"
End of Loop
new Loop
MeterValues
Instancia LuminositySensor enviada: {"LuminositySensor":{"identificador":"Luminosity001", "timestamp" : 1 , "medida" :8 , "unidad" : "lumen"
End of Loop
  
```

Desplazamiento automático | No hay fin de línea | 9600 baud

- Open the Consumer KP with a web browser.
  - Connect to the SIB: To do this, simply click on the **ConnectSIB** button.

Conectar / Desconectar al SIB

Token: 3ab58085dbf84a11843cefb143b327eb	Instance: KpvisualizacionLum:KpvisualizacionLum01
Conectar SIB Desconectar SIB	

Suscripcion

Suscripción: L lm >	SensorLuminosidad (lm): 1	Refresco (ms): 1000
Suscribir Desuscribir		
Conectado al sib con sessionkey: ea47cb04-e56b-4680-9b46-3a9566c2a951		

- Subscribe: Simply click on the **Start** button

Suscripcion

Suscripción: L lm >	SensorLuminosidad (lm): 1	Refresco (ms): 1000
Suscribir Desuscribir		
Suscrito con id: 9642852c-dea3-4fb3-98cc-ded54d585039 a query: {LuminositySensor.va:{\$gt:1}}		

- Check that the chart shows the data that the Producer KP is sending. The effect should be automatic from the moment the subscription is launched, if the Producer KP is sending data to the SIB.

