
Sentilo Documentation

Release 1.8.0

Sentilo

May 09, 2019

Contents

1	Setup	3
2	Quickstart	17
3	API Docs	19
4	Architecture	75
5	Integrations	83
6	Catalog and Maps	93
7	Multi Tenant	137
8	Clients	149
9	Technical FAQ	187
10	Platform Testing	189
11	Use a Virtual Machine	191

Contents:

This guide describes how to: **download, configure, compile and install the last version of Sentilo in your own runtime environment**. Moreover, it details which are the infrastructure elements necessary for running Sentilo and how should be their default configuration settings. It's assumed you have the skills to configure and install the necessary software base(Operating System, Maven,JDK, Mongo DB, Redis, etc).

The main topics are:

- **Prerequisites:** describes the software elements that have to be installed before download the code.
- **Download and build:** explains the steps to obtain the Sentilo code, to adapt it and how to build the platform artifacts.
- **Platform infrastructure:** describes the mandatory infrastructure components for running Sentilo and its default configuration settings.
- **Deploy the artifacts:** describes the necessary steps to deploy all the Sentilo modules

1.1 Prerequisites

Sentilo uses Maven as a mechanism for building and managing the dependencies of the platform. In order to build **Sentilo**, it is necessary to ensure the next set of prerequisites:

- JDK 1.8.x +
- Git (**optional**)
- Maven 3 +
- Ensure that the the Java SDK and Maven executables are accessible using your PATH environment variable.

1.2 Download and build code

The Sentilo code must be downloaded from Github. Once downloaded, you can build it using a script named *buildSentilo.sh* which constructs the Sentilo artifacts “out-of-the-box”.

1.2.1 Download the source code from Github

The source code of the project can be obtained from git, cloning the remote project in a local directory named *sentilo*:

```
git clone https://github.com/sentilo/sentilo.git sentilo
```

An alternative method is to download a ZIP file from github repository and decompress it in a folder named *sentilo*:

<https://github.com/sentilo/sentilo/archive/master.zip>

In both cases, we will finally have a new directory named *sentilo* with the source code.

1.3 Compiling and build artifacts

1.3.1 Without changing the default configuration

If you want to build Sentilo out-of-the-box (i.e. build all artifacts that define the Sentilo platform without changing any of the default settings that are defined), we distribute a script named *./scripts/buildSentilo.sh* which can be used to build Sentilo from the command line.

This script compiles the code and build the artifacts from scratch, but it doesn't deploy them in the execution environments. This process must be done manually by different reasons, for example:

- The deployment environment could be distributed in different servers. In example, Tomcat server and Pub/Subscribe server.
- it's not required to install all the components, like the relational database agent.

1.3.2 Changing default settings

If you want modify the code before to build it, you should import it into an Eclipse workspace with maven plug-in installed. Below we explain how to do it by using the M2E plugin.

- Open the Eclipse workspace to import the code:
 - Go to **File> Import> Existing Maven Projects**
 - Select **./sentilo** as the root directory
 - Select all projects and import

Warning: be sure that JDK 1.8, or later, is correctly configured in your Eclipse environment.

After modifying the code, to compile and build the artifacts, our recommendation is to use the abovementioned* *buildSentilo** script.

1.4 Platform infrastructure

Before describing how to install all the Sentilo components, we're going to explain how to configure each element of the infrastructure.

Sentilo uses the following infrastructure elements (they are grouped into two categories):

- Mandatory
 - Redis 4.0.11
 - MongoDB 4.0.1
 - Tomcat 8.5.32 +
- Optional
 - MySQL 5.5.x (Sentilo has been tested on MySQL 5.5.34 but you could use your favourite RDBMS) **It is only necessary if you want to install the relational agent**
 - Elasticsearch 5+ **It is only necessary if you want to install the activity-monitor agent.**
 - openTSDB 2.2.0 + **It is only necessary if you want to install the historian agent**

You must ensure that you have all these elements installed properly (you can find information on how to install them in each provider site).

Below we explain the default settings for each Sentilo module.

1.4.1 Default settings

Sentilo configuration uses the Spring and Maven profiles to allow its customization depending on the runtime environment. By default, the platform comes with a predefined profile named **dev**, which considers that each of these infrastructure elements are installed on the same machine and listening in the following ports:

- Redis: 6379
- MongoDB: 27017
- Tomcat: 8080
- MySQL: 3306
- Elasticsearch: 9200
- openTSDB: 4242

All these settings can be found in the subdirectory `/src/main/resources/properties` of each platform's module.

1.4.2 Redis settings

Sentilo default settings consider Redis will be listening on port 6379, host 127.0.0.1, and with the parameter `requirepass` enabled and with value **sentilo**.

If you change this behaviour, you need to modify the following properties:

```
jedis.pool.host=127.0.0.1
jedis.pool.port=6379
jedis.pool.password=sentilo
```

which are configured in the following files:

```
sentilo-platform/sentilo-platform-service/src/main/resources/properties/jedis-config.  
↪properties  
sentilo-agent-alert/src/main/resources/properties/jedis-config.properties  
sentilo-agent-relational/src/main/resources/properties/jedis-config.properties  
sentilo-agent-location-updater/src/main/resources/properties/jedis-config.properties
```

1.4.3 MongoDB settings

Sentilo default settings consider MongoDB will be listening on 127.0.0.1:27017, and requires an existing database named *sentilo*, created before starting the platform, with [authentication enabled](#) and with login credentials preconfigured as *sentilo/sentilo* (username~:*sentilo*, password~:*sentilo*).

If you change this behaviour, you need to modify following properties:

```
catalog.mongodb.host=127.0.0.1  
catalog.mongodb.port=27017  
catalog.mongodb.user=sentilo  
catalog.mongodb.password=sentilo
```

configured in the following files:

```
sentilo-agent-alert/src/main/resources/properties/catalog-config.properties  
sentilo-catalog-web/src/main/resources/properties/catalog-config.properties
```

Data load

Moreover, you need to load on *sentilo* database the basic set of data needed to run the platform. The data include, among other things:

- An user **admin**: user for log in into the catalog webapp as administrator.
- An user **sadmin**: user for log in into the catalog webapp with role super-admin.
- A default **sentilo** tenant: used to configure the default viewer parameters (center, zoom, ...) from the catalog web app.
- An entity **sentilo-catalog**: internal app used by the platform to synchronize information between its components.
- An user **platform_user**: internal user used by the platform to synchronize information between its components.

To do this, you must load the data defined in the file:

```
./scripts/mongodb/init_data.js
```

For example, in your MongoDB machine, you should execute the following command from the directory where the file is located:

```
mongo -u sentilo -p sentilo sentilo init_data.js
```

Remember:

Please keep in mind that data defined in the previous file contains default passwords and tokens (which are recommended for run Sentilo in a test environment). In order to avoid compromising your platform, **we recommend to change them before installing Sentilo in a production environment**.

After change their values in the *init_data.js* and load them on MongoDB, and before compiling and building Sentilo, you will have to modify the following properties:

```
rest.client.identity.  
↪key=c956c302086a042dd0426b4e62652273e05a6ce74d0b77f8b5602e0811025066  
catalog.rest.credentials=platform_user:sentilo
```

configured in the following files:

```
sentilo-agent-alert/src/main/resources/properties/platform-client-config.properties  
sentilo-catalog-web/src/main/resources/properties/catalog-config.properties  
sentilo-platform/sentilo-platform-service/src/main/resources/properties/integration.  
↪properties
```

Test data load

In order to validate the correct installation of the platform, we could load a set of test data. These data includes, among other things: sensor types, component types, apps and providers.

These data is defined in the file:

```
./scripts/mongodb/init_test_data.js
```

and, as pointed aout above, you should run the following command to load it:

```
mongo -u sentilo -p sentilo sentilo init_test_data.js
```

1.4.4 MySQL settings

Remember:

This software is mandatory only if you want to export the published events to a relational database using the specific agent. Otherwise, you can skip this step. Please, check [this](#) out for more info.

Sentilo default settings consider MySQL server will be listening on 127.0.0.1:3306, and requires an existing database named *sentilo*, created before starting the platform, with authentication enabled and accessible using credentials *sentilo_user/sentilo_pwd* (username~:*sentilo_user*, password~:*sentilo_pwd*).

If you change this behaviour, you need to modify the following properties:

```
sentiloDs.jdbc.driverClassName=com.mysql.jdbc.Driver  
sentiloDs.url=jdbc:mysql://127.0.0.1:3306/sentilo  
sentiloDs.username=sentilo_user  
sentiloDs.password=sentilo_pwd
```

configured in the file:

```
sentilo-agent-relational/src/main/resources/properties/relational-config.properties
```

Creating the tables

Once we have MySQL configured, and the database *sentilo* created, the next step is to create the database tables required to persist historical platform data.

At the following directory of your Sentilo installation:

```
sentilo-agent-relational/src/main/resources/bd
```

you'll find the script to create these tables.

1.4.5 Tomcat settings

Sentilo default settings consider Tomcat will be listening on 127.0.0.1:8080.

If you change this behaviour, you need to modify the following property:

```
catalog.rest.endpoint=http://127.0.0.1:8080/sentilo-catalog-web/
```

configured in the following files:

```
sentilo-platform/sentilo-platform-service/src/main/resources/properties/integration.  
→properties  
sentilo-agent-location-updater/src/main/resources/properties/integration.properties
```

Your Tomcat should also be started with the user timezone environment variable set as UTC. To set Timezone in Tomcat, the startup script (e.g. *catalina.sh* or *setup.sh*) must be modified to include the following code:

```
-Duser.timezone=UTC
```

1.4.6 Elasticsearch settings

Remember:

It is only necessary if you want to index into Elasticsearch all the published events using the specific agent. Otherwise, you can skip this step. Please, check [this](#) out for more info.

Sentilo default settings consider Elasticsearch server will be listening on localhost:9200. If you change this behaviour, you need to modify the following property:

```
elasticsearch.url=http://localhost:9200
```

configured in the following file:

```
sentilo-agent-activity-monitor/src/main/resources/properties/monitor-config.properties
```

1.4.7 openTSDB settings

Remember:

It is only necessary if you want to store into openTSDB all the published events using the specific agent. Otherwise, you can skip this step. Please, check [this](#) out for more info.

Sentilo default settings consider openTSDB server will be listening on 127.0.0.1:4242. If you change this behaviour, you need to modify the following property:

```
opentsdb.url=http://127.0.0.1:4242
```

configured in the following file:

```
sentilo-agent-historian/src/main/resources/properties/historian-config.properties
```

1.4.8 Subscription/publication platform settings

Sentilo default settings consider subscription/publication server (a.k.a. *PubSub* server) will be listening on 127.0.0.1:8081

If you change this behaviour, you need to modify the following properties:

```
port=8081
rest.client.host=http://127.0.0.1:8081
```

configured in the following files:

```
sentilo-platform/sentilo-platform-server/src/main/resources/properties/config.
↪properties
sentilo-catalog-web/src/main/resources/properties/catalog-config.properties
```

1.4.9 Configuring logs

Sentilo uses **slf4j** and **logback** as trace frameworks. The configuration can be found in **logback.xml** file, located in the subdirectory **src/main/resources** of sentilo-common module of the platform.

By default, all platform logs are stored in the directory **/var/log/sentilo**

1.5 Platform installation

Once you have downloaded the code and you have modify, compile and built it, the next step is to deploy Sentilo artifacts. The platform has five artifacts:

- Web Application Catalog (is **mandatory**)
- Server publication and subscription (is **mandatory**)
- Internal agents (are **optional**):
 - alarms agent
 - relational database agent
 - location updater agent

1.5.1 Installing the Web App Catalog

After build Sentilo, to install the Web App, you just need to deploy the WAR artifact in your Tomcat server, i.e., copy the WAR artifact into the *webapps* subdirectory of your Tomcat server.

You will find the WAR artifact at the following subdirectory:

```
./sentilo-catalog-web/target/sentilo-catalog-web.war
```

1.5.2 Installing subscription/publication server

After build Sentilo, to install the PubSub server, you need to follow the following steps:

- a. Into the directory `./sentilo-platform/sentilo-platform-server/target/appassembler` you'll find two subdirectories named **repo** and **bin**:
 - **repo** directory contains all libraries needed to run the process
 - **bin** directory contains the script (`sentilo-server`) needed to initialize the process (there are two scripts, one for Linux systems and one for Windows)
- b. Copy these two directories in the root directory where you want to install this component (for example: `/opt/sentilo-server`).
- c. Once copied, for starting the process you just need to run the script:

```
$sentilo-server/bin/sentilo-server
```

1.5.3 Installing agents

As have been mentioned previously, all agents are optional and you are free to choose which of them will be deployed, depending on your specific needs. Agents are internal modules oriented to expand the platform functionality without having to alter its core. You will find more information about them in the [Integrations](#) section of our documentation.

We have currently *seven core* agents:

- **Alarms agent** is responsible for processing each internal alert defined in the catalog and publish a notification (a.k.a. *alarm*) when any of the configured integrity rules are not met. You need this agent if you want to make use of the internal alerts functionality provided by Sentilo.
- **Relational agent** is responsible for store all information received from the PubSub server into a set of relational databases. You need this agent if you want to persist data published in Sentilo in a relational database too.
- **Location updater agent** is responsible for updating automatically the component location according to the location of the published observations.
- **Historian agent** is responsible for store all information received from the PubSub server into a time series database. You need this agent if you want to persist data published in Sentilo in openTSDB too.
- **Activity monitor agent** is responsible for index all information received from the PubSub server into a search engine server. You need this agent if you want to store data published in Sentilo into Elasticsearch too.
- **Kafka agent** Publishes events to Kafka.
- **Federation agent** Synchronizes two independent Sentilo instances, publishing selected observations from a set of providers to another Sentilo.

Remember: As mentioned before, Sentilo always store all published events into Redis.

All the agents are installed in a similar manner to the PubSub server, as described below.

Installing alarms agent

After build Sentilo, to install the alarms agent, you need to follow the following steps:

- a. Into the directory `./sentilo-agent-alert/target/appassembler` you'll find two subdirectories named **repo** and **bin**:
 - **repo** directory contains all libraries needed to run the process

- **bin** directory contains the script (*sentilo-agent-alert-server*) needed to initialize the process (there are two scripts, one for Linux systems and one for Windows)
- b. Copy these two directories in the root directory where you want to install this component (for example: `/opt/sentilo-agent-alert`).
- c. Once copied, for starting the process you just need to run the following script:

```
$sentilo-agent-alert/bin/sentilo-agent-alert-server
```

Installing relational agent

As mentioned before, this agent exports all the received data, orders and alarms to a database named *sentilo* and located in the MySQL server.

These configuration settings are defined in the files:

```
./sentilo-agent-relational/src/main/resources/properties/subscription.properties
./sentilo-agent-relational/src/main/resources/properties/relational-client-config.
↪properties
```

To modify this behavior, just follow the instructions given in the properties files.

Additionally, with the purpose of optimizing the persistence process, insert process is done in batch mode and uses a *retries* parameter aimed to minimize any error. By default, the *batch size* is fixed to 10 records and the *retries* parameter is defined to 1.

This behaviour can be changed editing the file:

```
./sentilo-agent-relational/src/main/resources/properties/relational-client-config.
↪properties
```

and updating the following lines:

```
# Properties to configure the batch update process
relational.batch.size=10
relational.batch.workers.size=3
relational.batch.max.retries=1
```

After building Sentilo, to install the relational agent, you only need to follow the following steps:

- a. Into the directory `./sentilo-agent-relational/target/appassembler` you'll find two subdirectories named **repo** and **bin**:
 - **repo** directory contains all libraries needed to run the process
 - **bin** directory contains the script (*sentilo-agent-relational-server*) needed to initialize the process (there are two scripts, one for Linux systems and one for Windows)
- b. Copy these two directories in the root directory where you want to install this component (for example: `/opt/sentilo-agent-relational`).
- c. Once copied, for starting the process you just need to run the script:

```
$sentilo-agent-relational/bin/sentilo-agent-relational-server
```

Installing location updater agent

After building Sentilo, to install the location updater agent, you need to follow the following steps:

- a. Into the directory `./sentilo-agent-location-updater/target/appassembler` you'll find two subdirectories named **repo** and **bin**:
 - **repo** directory contains all libraries needed to run the process
 - **bin** directory contains the script (`sentilo-agent-location-updater-server`) needed to initialize the process (there are two scripts, one for Linux systems and one for Windows)
- b. Copy these two directories in the root directory where you want to install this component (for example: `/opt/sentilo-agent-location-updater`).
- c. Once copied, for starting the process you just need to run the script:

```
$sentilo-agent-location-updater/bin/sentilo-agent-location-updater-server
```

Installing historian agent

As mentioned before, this agent exports all the received events to a openTSDB server.

This agent works in a similar way to the relational agent: insert process is done in batch mode and uses a *retries* parameter aimed to minimize any error. By default, the *batch size* is fixed to 10 records and the *retries* parameter is defined to 1.

This behaviour can be changed editing the file:

```
./sentilo-agent-historian/src/main/resources/properties/historian-config.properties
```

and updating the following lines:

```
# Properties to configure the batch update process
batch.size=10
batch.workers.size=3
batch.max.retries=1
```

After building Sentilo, to install the historian agent, you only need to follow the following steps:

- a. Into the directory `./sentilo-agent-historian/target/appassembler` you'll find two subdirectories named **repo** and **bin**:
 - **repo** directory contains all libraries needed to run the process
 - **bin** directory contains the script (`sentilo-agent-historian-server`) needed to initialize the process (there are two scripts, one for Linux systems and one for Windows)
- b. Copy these two directories in the root directory where you want to install this component (for example: `/opt/sentilo-agent-historian`).
- c. Once copied, for starting the process you just need to run the script:

```
$sentilo-agent-historian/bin/sentilo-agent-historian-server
```

Installing activity-monitor agent

As mentioned before, this agent exports all the received events to elasticsearch server.

This agent works in a similar way to the relational agent: insert process is done in batch mode and uses a *retries* parameter aimed to minimize any error. By default, the *batch size* is fixed to 10 records and the *retries* parameter is defined to 1.

This behaviour can be changed editing the file:

```
./sentilo-agent-historian/src/main/resources/properties/monitor-config.properties
```

and updating the following lines:

```
# Properties to configure the batch update process
batch.size=10
batch.workers.size=3
batch.max.retries=1
```

After building Sentilo, to install the activity-monitor agent, you only need to follow the following steps:

- a. Into the directory *./sentilo-agent-activity-monitor/target/appassembler* you'll find two subdirectories named **repo** and **bin**:
 - **repo** directory contains all libraries needed to run the process
 - **bin** directory contains the script (*sentilo-agent-activity-monitor-server*) needed to initialize the process (there are two scripts, one for Linux systems and one for Windows)
- b. Copy these two directories in the root directory where you want to install this component (for example: */opt/sentilo-agent-activity-monitor*).
- c. Once copied, for starting the process you just need to run the script:

```
$sentilo-agent-activity-monitor/bin/sentilo-agent-activity-monitor-server
```

1.6 Enable multi-tenant instance

In order to enable multi-tenant feature you need to ensure that your Sentilo version is at least 1.5.0. Otherwise you will have to [upgrade](#) your Sentilo instance.

Once the above requirement is fulfilled, you only need to do the following steps:

1.6.1 Modify your Tomcat startup script

You should modify your Tomcat startup script (e.g *%TOMCAT_HOME%/bin/catalina.sh* or *%TOMCAT_HOME%/bin/setenv.sh*) to add a new JVM property:

```
-Dsentilo.multitenant=true
```

Once you have added the JVM property, you must restart your Tomcat server.

1.6.2 Edit the Catalog web.xml file

The next step is to edit the Catalog file *web.xml* located at:

```
sentilo-catalog-web/src/main/webapp/WEB-INF/web.xml
```

You will find some lines that are commented into this file which are needed to enable the multi-tenant feature. Therefore you should uncomment them:

```
<!--
    <filter>
        <filter-name>UrlRewriteFilter</filter-name>
        <filter-class>org.tuckey.web.filters.urlrewrite.UrlRewriteFilter</filter-
↪class>
        <init-param>
            <param-name>logLevel</param-name>
            <param-value>slf4j</param-value>
        </init-param>
    </filter>

    <filter>
        <filter-name>tenantInterceptorFilter</filter-name>
        <filter-class>org.sentilo.web.catalog.web.TenantInterceptorFilter</filter-
↪class>
    </filter>
-->

<!--
    <filter-mapping>
        <filter-name>tenantInterceptorFilter</filter-name>
        <url-pattern>/*</url-pattern>
        <dispatcher>REQUEST</dispatcher>
    </filter-mapping>
    <filter-mapping>
        <filter-name>UrlRewriteFilter</filter-name>
        <url-pattern>/*</url-pattern>
        <dispatcher>REQUEST</dispatcher>
        <dispatcher>FORWARD</dispatcher>
    </filter-mapping>
-->
```

Once you have uncomment the above lines, you should recompile the Catalog webapp module and redeploy it into your Tomcat server.

You will find more information about this feature in the [Multi-Tenant](#) section of our documentation.

1.7 Enable anonymous access to REST API

By default, anonymous access to REST API is disabled which means that all requests to REST API must be identified with the `identity_key` header.

From version 1.5, we provide a new feature that allows anonymous access to REST API but only for read *authorized* data of your Sentilo instance (here *authorized* means that you should configure your Catalog to define which data could be accessed anonymously from REST requests).

In order to enable anonymous access you should modify the following properties:

```
# Properties to configure the anonymous access to Sentilo
enableAnonymousAccess=false
anonymousAppClientId=
```

configured in the following file:

```
sentilo-platform/sentilo-platform-server/src/main/resources/properties/config.  
↪properties
```

This configuration has not mystery: if anonymous access is enabled (*enableAnonymousAccess=true*) then all anonymous requests to REST API are internally considered as if they have been performed by the application client identified by the *anonymousAppClientId* property value (this application client should exist in your Sentilo Catalog), and therefore these requests will have the same data restrictions as the requests performed by this client application.

1.8 What next?

Check the [Quick Start Page](#) or [Platform Testing page](#).

2.1 Prerequisites

After checking the [Sentilo Setup](#). No need to setup any agents or other optional components such as Elasticsearch or OpenTSDB.

You should have 4 components up & running:

- Redis Server
- MongoDB
- Sentilo API, running at <http://127.0.0.1:8081>
- Sentilo Catalog, running at <http://127.0.0.1:8080/sentilo-catalog-web>

2.2 Create a Provider, Component and a Sensor

In order to create a publication of sensor data, we have to create first the Provider, Component and a Sensor.

We'll do that from the catalog application as superuser, using the admin/admin credentials.

A provider is an entity that manages devices (sensors). We'll have to create one from the menu "Providers" -> "New Provider"

A component is a device that contains one or more sensors (such as a Raspberry PI). We'll have to create one from the menu "Components" -> "New Component". Make sure you select the provider created above.

Finally, we'll have to create a sensor from the menu "Sensors" -> "New Sensor". Make sure you select the component created above. Please select a numeric type of sensor.

2.3 Publish an Observation

In order to publish an observation, we'll use Sentilo's HTTP REST API. For that you can use the curl program or some more graphical tool such as [Postman](#):

```
curl -X PUT -H "IDENTITY_KEY: <your provider's token>" http://<your sentilo url>/data/  
-><your provider>/<your sensor>/42.0
```

The server should respond with HTTP status 200.

2.4 Read your Observations

```
curl -X GET -H "IDENTITY_KEY: <YOUR_KEY>" http://<your sentilo url>/data/<your_  
->provider>/<your sensor>
```

The response would be similar to:

```
{  
  "observations": [  
    {  
      "value": "42.0",  
      "timestamp": "22/11/2016T11:52:28",  
      "location": ""  
    }  
  ]  
}
```

Also, on the “Latest Data” tab of the sensor’s page in the catalog will appear your value, in this case, a 42.0.

2.5 What next?

Check the [API documentation](#) here.

Contents:

3.1 General Model

3.1.1 Intro

Sentilo offers an open source API based on REST interfaces.

Representational State Transfer (REST) is a style of architecture that exploits existing technologies and protocols of the World Wide Web (WWW).

The communication from external elements with Sentilo will be through HTTP protocol (Hypertext Transfer Protocol).

Here, briefly describes the concepts of REST terminology that Sentilo will use:

- **Resources:** Elements of the information system.
- **Identifiers:** Unique name that identifies a resource within the system.
- **Representations:** Format of the exchanged data.
- **Operations:** Actions that can be performed on a resource.
- **Response codes:** Result of the operation.

3.1.2 Resources

Resources, or pieces of information of Sentilo Platform, are:

- **Sensor:** item of hardware or software with the ability to generate an observation(data).
- **Component:** corresponding to a element of hardware or software, with geospatial location (fixed or mobile) who could be composed by 1 or more Sensors.

- **Provider:** entity that represents a grup of components and allows them to communicate with Sentilo for sending data and receive commands.
- **Client application / Module:** entity that consumes the data processed by the platform.

The actions that can be carried out are:

- **Applications / Modules**
 - Register on the platform, but always from the administration console.
 - Send orders to providers/sensors (order service).
 - Receive data from provider/sensors (data service).
 - Subscribe to system events (subscribe service).
- **Providers / Sensors**
 - Register on the platform (catalog service).
 - Subscribe to system events (subscribe service).
 - Publish data (data service).

Sensors and components have always a an associated typology.

3.1.3 Identifier

Unique name that identifies a resource in the system.

In the case of Sentilo, it is an **URLs** (Uniform Resource Locator).

The base URL is composed as follows:

```
protocol://domain:port/service
```

and consists of the following parts:

- **communication protocols:** HTTP or HTTPS.
- **domain:** Platform server API domain (e.g. localhost).
- **port:** Port defined for communications with the server API (e.g. 8081).
- **service:** catalog, data, order, etc..

Every service has a custom URL format as especified for each services.

3.1.4 Representations

Data formats that will support the platform are: **JSON** (initial version) and **XML** (in a future releases).

The default format used by the platform is **JSON**.

To specify a different format, you must add the parameter **“format”** in the request, as shown in the following example:

```
http://<your_api_server.com>/service/<id_provider>?format=XML
```


JSON format

Example data in JSON format:

```
{ "observations": [
  { "value": "12.3", "timestamp": "17/09/2012T12:34:45" }
]}
```

XML format

Example data in XML format:

```
<observation>
  <value>12.3</value>
  <timestamp>17/09/2012T12:34:45</timestamp>
</observation>
```

3.1.5 Operators

The platform operators are the **HTTP protocol methods**.

In general, the operation associated with the operations used by Sentilo are:

- **GET**: Request information.
- **POST**: Send new data.
- **PUT**: Update existing data.
- **DELETE**: Erase data.

The platform discriminates the action you want perform from the method used and by the service, provider or sensor specified in the URL invoked.

3.1.6 Reply

The response to a request to the platform is managed through the response **HTTP status codes**.

Error Code	HTTP	Description
200	Success	Request accepted and processed correctly
4xx	Client Error	Error in request (Wrong format, forbidden mandatory parameters, ...)
401	Unauthorized	Unauthorized request: empty or invalid credential
403	Forbidden	Not authorized for the requested action
5xx	Server Error	Error processing the request

In case of error the response body will include a description of the problem detected, as shown in the following examples:

This payload is returned when no credential is sent:

```
{ "code": 401, "message": "Invalid credential null" }
```

This payload is returned when JSON payload could not be read as JSON:

```
{ "code": 400,
  "message": "SIE03-1398350628224 Bad request data: could not read JSON payload.↵
↵Please review the following error and try again",
  "errorDetails": ["org.sentilo.common.exception.MessageNotReadableException:↵
↵Unexpected character ('o' (code 111)): ....."]}]
```

3.2 Security

3.2.1 Securing API requests

The platform will validate any request received by the system following the terminology AAA (*Authentication, Authorization, *Accounting*):

- **Authentication:** Identifying who is doing the request.
- **Authorization:** Validating that the action requested on the resource associated can be done.
- **Traceability:** Auditing the action and who has performed it.

So, for each request received, the platform performs the following actions:

- Identify the petitioner through the header HTTP.
- Check that it can do the requested action on the resource indicated.
- Register the performed action.

When necessary, the platform will check the integrity and confidentiality of communications is ensured using **HTTPS** protocol.

Authentication

To identify the petitioner, the platform uses an authentication mechanism based on tokens (**Token Based Authentication**).

It's necessary to establish a distribution mechanism outside the platform for send the tokens among the different users of the platforms securely. Future versions of Sentilo will include this feature.

The token will be included in the request by adding a header with key **IDENTITY_KEY**.

An example of a service request (GET in this case) using the curl tool:

```
curl --request GET --header "IDENTITY_KEY: <YOUR_KEY>" http://<your_api_server.com>/
↵resource
```

In case of incorrect or invalid token , the platform will respond with an error code 401.

Authorization

To validate the requested action on the resource indicated in the request can be performed, the platform uses a permit system that checks authorized entity (provider or application) is allowed to admin, write or read in a resource.

These permissions are defined via the catalog console of the platform and, by default, every entity is administrable by its owner.

If an action on a resource is done without the appropriate permission, platform will return an error 403.

3.2.2 Securing Callbacks

If it's necessary to secure the push requests sent by the platform, Sentilo provides a [HMAC](#) mechanism for the callbacks.

This mechanisms guarantees:

- That the message was sent by the platform
- That the message was not altered after sent
- That the message is still active

As hash algorithm the system uses [SHA-512](#). It accepts keys of any size, and produces a hash sequence of length 512 bits.

The target system should activate the security for callbacks when creates the subscription specifying the secret key ([see more](#)). This subscription **should be done using HTTPs protocol** to avoid compromising the key.

After the subscription has been created, all the related requests will include two new headers, one with the hash (**Sentilo-Content-Hmac**) and another with the timestamp (**Sentilo-Date**), as the following sample shows:

```
Sentilo-Content-Hmac:
j1OQ+fU667GQoHYHWzLBpigRjLJmRvYn53KHZhApTbrcphYWB1RPSBHkntODuqsqx11Vj8rsc7DDziiutTq/
↪5g==
Sentilo-Date: 10/06/2014T15:27:22
```

The responsibility of validating the headers will be always in the target system who is receiving the messages.

The pseudo-code to generate the HMAC token is the following:

```
var md5Body = MD5(body)
var endpoint = endpoint_configured_in_subscription
var secretKey = secret_key_configured_in_subscription
var currentDate = value_http_header_Sentilo-Date
var contentToSign = concatenate('POST',md5Body, 'application/json',currentDate,↪
↪endpoint)
var signature = HmacSHA512(contentToSign)

return base64UrlEncode(signature)
```

3.3 Services

Index of differents services.

3.3.1 Alarm

Publish Alarm

Description

This action allows you to publish an alarm related with an alert. Once the system receives the alarm, persists it and sends the notification to all who are subscribed to alarms alert.

```
http://<your_api_server.com>/alarm/<alert_id>
```

Formats	json
Method	PUT
Permission	Writing
Return	No additional data is returned

Parameters

Each alarm will have its own associated information structure defined in the generic format (JSON).

The platform only persists and transfers the information to recipients without interpreting its contents.

Key	Description	Optional
message	Free field	Not

Response data

This actions does not return additional data beyond the [HTTP status code](#) associated with each request to the platform.

Examples

Post a new alarm associated with an alert

The following example shows how to send a request to the platform in order to publish a new alarm associated to an alert with identifier 43:

```
http://<your_api_server.com>/alarm/43
```

and like body message:

```
{ "message": "Threshold limit exceeded: 32" }
```

Please note the following:\

- (% style="font-size: 16px; background-color: rgb(245, 245, 245);" %)If the alert is in offline state, the server rejects the publication.

Retrieve alarms

Description

This action allows to retrieve the latest alarms related with an alert. In addition, the service can also specify search criterias to retrieve alarms: filter by a given time period and/or indicate the maximum number of alarms to be retrieved.

```
http://<your_api_server.com>/alarm/<alarm_id>?<parameter>=<value>
```

Format	json
Method	GET
Permission	Reading
Return	Alarms associated with the alert

Parameters

Key	Description	Optional
from	Indicates the starting of the time period for which you want to retrieve alarms.	Yes
to	Indicates the end of the time period for which you want to retrieve alarms..	Yes
limit	Specifies the maximum number of alarms to recover.	Yes

Please note the following:

- The maximum number of records returned is defined in the platform configuration. If the limit parameter has a higher value than the configured one it will be dismissed.
- If limit parameter is not specified, it returns only one alarm.
- All dates must follow the format: dd/MM/yyyyTHH:mm:ss

Response data

In addition to the appropriate [HTTP status code](#), if the operation runs properly, it will return the last alarms associated with the alert according to your search criteria.

Key	Description	Optional
alarms	Alarms list (<i>message</i>) of the alert	Not

Each alarm (message) will be composed by the following attributes:

Key	Description	Opcional
message	Message recorded when the alarm was fired	No
timestamp	The time in which system received the alarm (format dd/MM/yyyyTHH:mm:ss)	No
time	The time when the observation was made in milliseconds	No
sender	Identifier of the entity that issued the alarm	No

Examples

Retrieve the last alarm

To retrieve the latest alarm for the alert with ID 43 we do the following request to the platform:

```
http://<your_api_server.com>/alarm/43
```

In the response we will receive:

```
{ "alarms": [
  {
    "message": "threshold exceeded",
    "timestamp": "08/04/2013T09:44:01",
    "time": 1510561800008,
    "sender": "appDemo"
  }
]}
```

Recover N alarms

To retrieve the last 3 alarms for the alert with id 43 we do the following request to the platform:

```
http://<your_api_server.com>/alarm/43?limit=3
```

In the response we will receive:

```
{ "alarms": [
  {
    "message": "threshold exceeded: 34",
    "timestamp": "08/04/2013T09:44:01",
    "time": 1510561800000,
    "sender": "appDemo"
  },
  {
    "message": "threshold exceeded: 37",
    "timestamp": "08/04/2013T09:14:01",
    "time": 1510561800001,
    "sender": "appDemo"
  },
  {
    "message": "threshold exceeded: 38",
    "timestamp": "07/04/2013T23:23:10",
    "time": 1510561800002,
    "sender": "appDemo"
  }
] }
```

Retrieve N alarms in a given period

If we want to retrieve the alarms according to a given period of time we should do the following request:

```
http://<your_api_server.com>/alarm/43?limit=3&from=08/04/2013T00:00:00&to=08/04/
↪2013T23:59:59
```

In response the we will receive:

```
{ "alarms": [
  {
    "message": "threshold exceeded: 34",
    "timestamp": "08/04/2013T09:44:01",
    "time": 1510561800000,
    "sender": "appDemo"
  },
  {
    "message": "threshold exceeded: 37",
    "timestamp": "08/04/2013T09:14:01",
    "time": 1510561800000,
    "sender": "appDemo"
  }
] }
```

Description

The alarm service allows you to record and retrieve alarms associated with an alert stored in the system catalog.

All requests for this service will have the following format:

```
http://<your_api_server.com>/alarm/<id_alert>
```

where `id_alert` identifies the alert for which you want to perform the action. The alert always should be defined before throwing the alarm using the Catalog or the through the [Alert](#) service.

Actions

The available actions for this service are:

- [Publish a new alarm associated with an alert](#)
- [Retrieve the latest alarms associated with an alert](#)

3.3.2 Alert

Create Alerts

Description

This action allows to register one or more new alerts in the catalog.

```
http://<your_api_server.com>/catalog/alert/<entity_id>
```

Formats	json
Method	POST
Permission	Writing
Returns	No output data

The internal alerts should be defined through the catalog console or by the API, but only using the catalog token.

Parameters

Key	Description	Optional
alerts	Alerts list (<i>alert</i>) to register	Not

Every alert element has the following structure:

Key	Description	Optional
id	Alert ID to register	No
name	Alert name	Yes
description	Alert description	Yes
type	Alert type	No
trigger	Trigger type	Mandatory for internal, not applies for externals
expression	Expression to evaluate with the trigger	Mandatory for internal, not applies for externals
component	ID of the component to which the sensor belongs	Mandatory for internal, not applies for externals
sensor	ID of the sensor to which the alert applies	Mandatory for internal, not applies for externals
entity	Related entity identifier associated with the alert	Yes

Please, note the following observations:

- The ID must identify an univocal alert, e.g., 2 alerts may not have the same ID.
- The ID must have only alphanumeric (i.e. letters and numbers) and dashes characters, with no embedded spaces.
- The list of trigger's types and expressions are defined by the platform: [Trigger types](#).
- The possible values for the alert types are: INTERNAL or EXTERNAL.
- Entity parameter is not mandatory, if empty the alert will be associated with the entity specified in the URL

Response data

This action doesn't return additional data beyond the [HTTP status code](#).

Examples

Adding one external alert

If rec entity wants to register a new custom external alert with REC_ALERT_001 identifier, to monitorize that maximum daily values for sensor REC_001 ranged from 60 and 80, the request will be:

```
http://<your_api_server.com>/catalog/alert/rec
```

and in the body message:

```
{ "alerts": [
  { "id": "REC_ALERT_001",
    "name": "REC_ALERT_001",
    "description": "Custom alert to monitorize that maximum daily values for sensor_
↪ REC_001 ranged from 60 and 80",
    "type": "EXTERNAL"
  }
] }
```

This request will register a new external alert with ID REC_ALERT_001 and associated to rec entity (i.e. rec entity is who will publish alarms associated to this alert).

Remember, the external alerts are defined by third party entities(providers or applications), which will be the responsables of calculating their logic and throw the related alarms when applies.

Adding one internal alert

If we want to register a new internal alert with ID REC_GT_45_ALERT_001, to monitorize that values for sensor's rec REC_001 are greater than 45, the request to do is the following:

```
http://<your_api_server.com>/catalog/alert/rec
```

and in the body message:

```
{ "alerts": [
  { "id": "REC_GT_45_ALERT_001",
    "name": "REC_GT_45_ALERT_001",
    "description": "Internal alert to monitorize that values for sensor's rec REC_001_
↪are greater than 45",
    "type": "INTERNAL",
    "trigger": "GT",
    "expression": "45",
    "component": "REC_COMP_001",
    "sensor": "REC_001"
  }
] }
```

This request will register a new internal alert with REC_GT_45_ALERT_001 identifier and associated to REC_001 sensor which will publish an alarm when sensor value will be greater than 45.

This operation must be done using the catalog token.

Update Alerts

Description

This action allows to update one or more alerts in the catalog.

```
http://<your_api_server.com>/catalog/alert/<entity_id>
```

Formats	json
Method	PUT
Permission	Writing
Returns	No output data

The internal alerts should be updated through the catalog console or by the API, but only using the catalog token.

Parameters

Key	Description	Optional
alerts	Alerts list (<i>alert</i>) to update	Not

Every alert element has the following structure:

Key	Description	Optional
id	Alert identifier to update	No
name	New alert name	Yes
description	New alert description	Yes
type	Alert type	No
trigger	New trigger type	Mandatory for internal, not applies for externals
expression	New expression to evaluate with the trigger	Mandatory for internal, not applies for externals

Please, note the following observations:

- The list of trigger's types and expressions are defined in: [Trigger types](#).
- The possible values for the alert type are: INTERNAL or EXTERNAL.

Response data

This action doesn't return additional data beyond the [HTTP status code](#).

Examples

Update one external alert

If rec entity wants to update the external alert with REC_ALERT_001 identifier to modify its name, the request to do will be:

```
http://<your_api_server.com>/catalog/alert/rec
```

and in the body message:

```
{ "alerts": [
  { "id": "REC_ALERT_001",
    "name": "REC_EXTERNAL_ALERT_001",
    "type": "EXTERNAL"
  }
]}
```

This request will update the external alert with REC_ALERT_001 identifier updating its name to REC_EXTERNAL_ALERT_001.

Remember, the external alerts are defined by third party entities(providers or applications), which will be the responsables of calculating their logic and throw the related alarms when applies.

Update one internal alert

If we want to update the internal alert with REC_GT_45_ALERT_001 identifier to change its description, the request will be:

```
http://<your_api_server.com>/catalog/alert/rec
```

and in the body message:

```
{
  "alerts": [
    {
      "id": "REC_GT_45_ALERT_001",
      "type": "INTERNAL",
      "description": "New description"
    }
  ]
}
```

This request will update the description of the internal alert with REC_GT_45_ALERT_001 identifier changing its value to “New description”.

This operation must be done using the catalog token.

Retrieve Authorized Alerts

Description

This action returns the list of alerts for which the entity_id could do a subscription, i.e., alerts that belongs to entity_id or alerts for which entity_id has read permission over its owner. In addition, the service also allows you to specify search criteria to filter alerts to be retrieved: filter by alert type and / or filter by trigger type.

```
http://<your_api_server.com>/catalog/alert/<entity_id>?<parameter>=<value>
```

The entity_id is optional and can be an Application or a Provider.

Format	json
Method	GET
Permission	Reading
Return	List of authorized alerts

Parameters

Key	Description	Optional
type	Alert's type filter	Yes
trigger	Trigger's type filter	Yes

Please, note the following observations:

- The list of trigger's types available are defined by the platform: [Trigger types](#).
- The possible values for the alert type is also defined by the platform and are: INTERNAL, EXTERNAL.

Response data

As commented before, this action, in addition to the [HTTP status code](#), returns the list of alerts for which entity_id has at least read permission.

Key	Description	Optional
alerts	Alerts list (<i>alert</i>)	Not

Every alert element has the following structure:

Key	Description	Optional
id	Alert ID	No
name	Alert name	Yes
description	Alert description	Yes
entity	Related entity	No
type	Alert type	No
trigger	Trigger type	No, but only returned for internal alerts
expression	Expression to evaluate with the trigger	No, but only returned for internal alerts
component	Component identifier to which the sensor belongs	No, but only returned for internal alerts
sensor	Sensor identifier to which the alert applies	No, but only returned for internal alerts

Examples

Request to retrieve all the authorized alerts

The following request shows an example to retrieve all the authorized alerts for rec entity:

```
http://<your_api_server.com>/catalog/alert/rec
```

and the response will be:

```
{
  "alerts" : [
    {
      "id" : "REC_ALERT_001",
      "name" : "REC_ALERT_001",
      "description" : "Custom alert to monitorize that maximum daily values for_
↪sensor REC_001 ranged from 60 and 80",
      "entity" : "SAMCLA",
      "type" : "EXTERNAL"
    },
    {
      "id" : "REC_ALERT_002",
      "name" : "REC_ALERT_002",
      "description" : "Internal alert to check if S00020114-0 value is greater than 45
↪",
      "entity" : "SAMCLA",
      "type" : "INTERNAL",
      "trigger" : "GT",
      "expression" : "45",
      "component" : "S00020114",
      "sensor" : "S00020114-0"
    }
  ]
}
```

Request to retrieve all the authorized alerts filtered by type and trigger

The following request shows an example to retrieve all internal alerts for rec entity with trigger type equal to GT.

```
http://<your_api_server.com>/catalog/alert/rec?type=INTERNAL&trigger=GT
```

and the response will be:

```
{
  "alerts": [
    {
      "id" : "REC_ALERT_002",
      "name" : "REC_ALERT_002",
      "description" : "Internal alert to check if S00020114-0 value is greater than 45",
      "entity" : "SAMCLA",
      "type" : "INTERNAL",
      "trigger" : "GT",
      "expression" : "45",
      "component" : "S00020114",
      "sensor" : "S00020114-0"
    }
  ]
}
```

Remove alerts

Description

This action allows to delete alerts from the catalog. The internal alerts can only be deleted using the Catalog's token or through the Catalog console. The external alerts can only be removed using the entity's owner token.

```
http://<your_api_server.com>/catalog/alert/<entity_id>
```

Remember, the entity_id can be also an Application or a Provider too.

Format	json
Method	DELETE, PUT
Permission	Writing
Return	No output data

Note that this action can be invoked using two HTTP methods: PUT and DELETE.

- DELETE will be used if we want to delete all of our alerts. It cannot contain any body content.
- PUT will be used when we want to delete a group of alerts. We should add the parameter method with delete value to the request . In this case, the alerts to delete should be specified in the body message.

Parameters

The structure of input message if we want to delete a group is:

Key	Description	Optional
alertsIds	Array of the alerts identifiers to delete	Yes

Each element of the list corresponds to an identifier to an alert to delete.

Response data

This action does not return additional data beyond the [HTTP status code](#).

Examples

Request to delete all alerts

If the entity rec wants to delete all its alerts, the request will be:

```
DELETE http://<your_api_server.com>/catalog/alert/rec
```

This action will delete all the external alerts belonging to entity rec. Be careful, if this request is done using the catalog token, it will remove all the internal alerts!.

Request to delete a set of alerts

If the entity rec only wants to delete a set of alerts, the request will be:

```
PUT http://<your_api_server.com>/catalog/alert/rec?method=delete
```

and in the body message:

```
{ "alertsIds": ["REC-ALERT-01", "REC-ALERT-02"] }
```

Description

The alert service provides methods to record, edit or retrieve alerts definition.

All requests for this service will have the following format:

```
http://<your_api_server.com>/catalog/alert/<entity_id>
```

where entity_id is optional and should be included depending on the operation. entity_id can be an Application or a Provider.

There are two alert types: **internal** and **external**.

The internal alerts are related to specific sensors and its logic is defined using basic math rules or configuring an inactivity time. They should be defined through the catalog console or by the API, but only using the catalog token.

The related alarms are triggered always by the Sentilo platform when the alert logic occurs.

The external alerts are defined by third party entities, which will be the responsables of calculating their logic and throw the related alarms when applies.

For both cases, the Sentilo platform is responsible of publishing the alarm for all entities subscribed to the related alert.

Actions

The available actions for this service are:

- Adding alerts
- Update alerts
- Retrieve list of authorized alerts
- Remove alerts

Internal trigger types

The list of trigger types accepted by Sentilo (and their associated expressions) are:

Id	Description	Expression value
GT	Greater than <i><expression></i>	Any numerical value
GTE	Greater than or equal <i><expression></i>	Any numerical value
LT	Less than <i><expression></i>	Any numerical value
LTE	Less than or equal <i><expression></i>	Any numerical value
EQ	Equal <i><expression></i>	Any value
CHANGE	Any change	Not apply here
CHANGE_DELTA	Any variation greater to delta <i><expression></i>	Any numerical value between 0 and 100
FROZEN	No data received in <i><expression></i> minutes	Any numerical value

The trigger types only apply for the internal alerts.

3.3.3 Catalog

Adding sensors or components to the catalog

Description

This action allows the provider to register one or more sensors / components in the catalog.

```
http://<your_api_server.com>/catalog/<provider_id>
```

Formats	json
Method	POST
Permission	Writing
Returns	No output data

Parameters

Key	Description	Optional
sensor	Sensor ID to register	No
description	Sensor description	Yes
type	Sensor type	No
dataType	Sensor data types	Yes
unit	Unit of measure	Yes
component	Component identifier to which the sensor belongs	Yes
componentType	Component type	Yes
componentDesc	Component description	Yes
location	Location/s of the component to which the sensor is	Yes
timeZone	TimeZone used by sensor observations when it is different to UTC	Yes
publicAccess	Visualization check for the sensor in the public zone	Yes
componentPublicAccess	Visualization check for the component in the public zone	Yes
additionalInfo	Additional params related to the sensor	Yes
componentAdditionalInfo	Additional params related to the component	Yes
technicalDetails	Technical params related to the sensor	Yes
componentTechnicalDetails	Technical params related to the component	Yes

Please, note the following observations:

- The state and substate of a sensor cannot be changed via the API, only from the catalog. The default value for state is 'online', default value for substate is empty.
- The identifier must identify an univocal sensor provider, e.g., 2 sensors of the same provider may not have the same ID.
- The identifier must have only alphanumeric (i.e. letters and numbers), underscores and hyphens characters, with no embedded spaces.
- The list of sensor's types are configured in the platform through the catalog web app. If you need a new one, it must be added using the administration.
- The possible values for the data type of the sensor is also defined in the platform configuration. The possible values are: number, text or boolean. The default value is number.
- If the attribute component is not passed as a parameter, the platform itself will create a catalog component with the same name as the sensor (if it does not already exist).
- If the attribute componentType is not reported and the component does not already exist in the system, the component will be defined as a generic component type.
- If the location attribute is not reported, the component is defined as a mobile type (with no fixed location). Otherwise it will be defined as static and set its location with the coordinates provided. If the element has several locations they should be informed separated by comma character.
- If the attribute type and / or componentType values are not configured in the catalog, the system will return a 400 error indicating that the parameters received are invalid .
- publicAccess param refers to the sensor's visibility in the sensor's public page. Default value is false.
- componentPublicAccess param refers to the components's visibility in the public map. Default value is false.
- additionalInfo param is a <key,value> map that allows to store additional sensor information not mapped to any specific parameter. The information of this data map must not follow any internal rule.
- componentAdditionalInfo param is a <key,value> map that allows to store additional component not mapped to any specific parameter. The information of this data map must not follow any internal rule.

- `technicalDetails` parameter is a `<key,value>` map that allows to store additional sensor information. The available keys and their possible values are:

Description	Key	Values
Producer	producer	not restricted
Model	model	not restricted
Serial number	serial-Number	not restricted
Energy	energy	220VAC (electric network), 12_24_VDC (PoE), 185_230_V (lighting network), AUT_BAT (battery), SOLAR_BAT (solar battery)

Response data

This action doesn't return additional data beyond the [HTTP status code](#) associated with each request to the platform.

Examples

Adding one sensor

If you want to register a new humidity sensor with RE0025 identifier associated with the component whose identifier is METEO-1 of rec provider, the request to do will be the following:

```
http://<your_api_server.com>/catalog/rec
```

and in the body message:

```
{ "sensors": [
  { "sensor": "RE0025",
    "description": "sensor 25 of moisture",
    "type": "humidity",
    "dataType": "number",
    "unit": "%",
    "component": "METEO-1",
    "componentType": "meteo",
    "componentDesc": "Test componente",
    "location": "41.39479 2.148768",
    "timeZone": "CET"
  }
] }
```

This request will register a new sensor with name METEO_HUM-1 in the system of humidity type . Additionally, this sensor will be associated with the component METEO-1. If the component does not exist in the system yet , will be registered with the properties defined in the request (`componentType`, `componentDesc` and `location`).

Adding several sensors

In case it is necessary to add a serie of sensors, the request will be very similar to the previous one, modifying the message body:

```
http://<your_api_server.com>/catalog/rec
```

in the body message

```
{ "sensors": [
  { "sensor": "tt01_REC013",
    "description": "sensor12",
    "type": "humidity",
    "dataType": "number",
    "unit": "grams",
    "component": "METEO-1",
    "componentType": "meteo",
    "location": "41.39479 2.148768"
  },
  { "sensor": "tt01_REC014",
    "description": "sensor12",
    "type": "humidity",
    "dataType": "number",
    "unit": "grams",
    "component": "METEO-1",
    "componentType": "estaciometeo",
    "location": "41.39479 2.148768"
  }
]}
```

In this case, instead of registering a single sensor, there will be added two new sensors associated with the component named METEO-1. If the component does not yet exist in the system, will be registered with the properties especificed in the request (type and localtzació).

Adding one sensor with additional info

If you want to register a new humidity sensor, as in the first example, but also need additional information for the sensor and its component, the request to do is the following:

```
http://<your_api_server.com>/catalog/rec
```

and in the body message:

```
{ "sensors": [
  { "sensor": "RE0025",
    "description": "sensor 25 of moisture",
    "type": "humidity",
    "dataType": "number",
    "unit": "%",
    "component": "METEO-1",
    "componentType": "meteo",
    "componentDesc": "Test componente",
    "publicAccess": "true",
    "componentPublicAccess": "true",
    "location": "41.39479 2.148768",
    "additionalInfo": { "accuracy": "4.5%", "voltage": "2.1-3.6" },
    "componentAdditionalInfo": { "altitude": "525 m." }
  }
]}
```

This request will register a new sensor with name METEO_HUM-1 in the system of humidity type, as in the first example, and stores with the sensor two new attributes: accuracy and voltage.

Update data of a component / sensor

Description

This action permits to update the catalog information related to components and/or sensors of a provider.

```
http://<your_api_server.com>/catalog/<provider_id>
```

Format	json
Method	PUT
Permission	Writing
Return	No output data

Parameters

The structure of the input parameters depends on what we want to modify, sensor or component data.

The following describes the structure of the input parameters in each case:

Update components

Key	Description	Optional
components	Components list (<i>component</i>) to update	Yes

Each element **component** has the following structure:

Key	Description	Optional
component	Component ID to update	No
componentType	Component type	Yes
componentDesc	Component description	Yes
location	Component location/s	Yes
componentPublicAccess	Visualization check for the public area	Yes
componentAdditionalInfo	Additional params	Yes
componentTechnicalDetails	Technical params	Yes

The constraints and validation for the parameters are the same as described in [Adding sensors or components](#).

Update sensors

Key	Description	Optional
sensors	Sensors list (<i>sensor</i>) to update	Yes

Each **sensor** element has the following structure:

Key	Description	Optional
sensor	Sensor ID to update	No
description	Sensor description	Yes
type	Sensor type	Yes
dataType	Data type of the sensor	Yes
unit	Measurement unit	Yes
publicAccess	Visualization check for the public area	Yes
additionalInfo	Additional params	Yes
technicalDetails	Technical params	Yes

The constraints and validation for the parameters are the same as described in [Adding sensors or components](#).

Response data

This action doesn't return additional data beyond the [HTTP status code](#).

Examples

Request to update the sensor data

If you want to modify the sensor's description for the identifiers RE0012 and RE0013, from rec provider, the request will be:

```
http://<your_api_server.com>/catalog/rec
```

in the body message:

```
{ "sensors": [
  { "sensor": "REC012", "description": "sensor 12" },
  { "sensor": "REC013", "description": "sensor 13" }
] }
```

This request will update the description of the sensors RE0012 and RE0013.

Note: If you need to move a sensor to another component, it should be done by deleting the sensor and creating it again in the other component.

Request to update the component data

If we want to update component data of a provider, like update its location and additional info, the request will be:

```
http://<your_api_server.com>/catalog/rec
```

in the message body:

```
{ "components": [
  { "component": "COMP-2", "location": "41.4051143 2.1320120", "componentAdditionalInfo": {
    ↪ "altitude": "530 m." } }
] }
```

Retrieve providers / sensors list

Description

This resource returns a list of providers and sensors for which you have at least read permission. Sensors that are in the offline state won't be listed. In addition, the service provides optional filtering by sensor type, component type and component name.

```
http://<your_api_server.com>/catalog
```

Format	json
Method	GET
Permission	Reading
Return	List of providers, with their sensors, on which we has at least read permission

Parameters

Key	Description	Optional
type	Sensor's type filter	Yes
component	Component name filter	Yes
componentType	Component's type filter	Yes

Response data

As mentioned, this action, in addition to the [HTTP status code](#), returns the list of providers for wich we have at least read permission.

Key	Description	Optional
providers	Providers list (<i>provider</i>) with at least read permission	Not

Each provider will have the following structure:

Key	Description	Optional
provider	Provider ID	No
permission	Indicates whether it readable (R) or write (W) on the provider	No
sensors	Provider list of sensors (sensor)	No

Each list element (**sensor**) will have the following structure.

Key	Description	Optional
sensor	Sensor identifier	No
description	sensor description	Yes
dataType	Data sensor type (NUMBER, BOOLEAN or TEXT)	No
location	Location where de sensor is	Yes
type	Sensor type	No
unit	Unities in the sensor data coming	Yes
timeZone	Sensor's timezone	Yes
publicAccess	Visualization check for the public area	Yes
component	Component is associated the sensor	No
componentType	Component type	No
componentDesc	Component description	Yes
componentPublicAccess	Visualization check for the public area	Yes
additionalInfo	Additional params related to the sensor	Yes
technicalDetails	Technical params related to the sensor	Yes
componentTechnicalDetails	Technical params related to the component	Yes

Examples

Request to retrieve all Providers / Sensors

```
http://<your_api_server.com>/catalog
```

in the response we will receive

```
{
  "providers": [{
    "provider": "A",
    "permission": "WRITE",
    "sensors": [{
      "sensor": "MAR_01_00_SN001_1010",
      "description": "Sound Sensor MODI 001",
      "dataType": "NUMBER",
      "type": "noise",
      "unit": "dBa",
      "component": "MAR_01_00_SN001_1010",
      "componentType": "generic",
      "timeZone": "CET"
    }]
  }, {
    "provider": "C",
    "permission": "READ",
    "sensors": [{
      "sensor": "MAR_02_20_PM001_1010",
      "description": "PM10 Sensor IMI 001",
      "dataType": "NUMBER",
      "type": "air_quality_pm10",
      "unit": "ug/m3",
      "component": "air_quality",
      "componentType": "generic"
    }], {
      "sensor": "MAR_02_20_PM001_1012",
      "description": "PM10 Sensor IMI 002",
```

(continues on next page)

(continued from previous page)

```

        "dataType": "NUMBER",
        "type": "air_quality_pm10",
        "unit": "ug/m3",
        "component": "air_quality",
        "componentType": "generic",
        "additionalInfo": {
            "supportMail": "support@imi.com"
        },
        "technicalDetails": {
            "producer": "xxxx",
            "model": "x-1",
            "serialNumber": "9999",
            "energy": "220VAC"
        },
        "componentTechnicalDetails": {
            "producer": "XXXX",
            "model": "X-1",
            "serialNumber": "9999",
            "macAddress": "00:17:4F:08:5F:61",
            "energy": "12_24_VDC",
            "connectivity": "WIFI"
        }
    }
}

```

Request to recover all the sensors in the catalog filtered by type

The request in this case is very similar to the previous one adding the type parameter:

```
http://<your_api_server.com>/catalog?type=air_quality_pm10
```

In this case as a response we will receive:

```

{
  "providers": [
    {
      "provider": "C", "permission": "READ",
      "sensors": [
        {
          "sensor": "MAR_02_20_PM001_1010",
          "description": "PM10 Sensor IMI 001",
          "dataType": "NUMBER",
          "type": "air_quality_pm10",
          "unit": "ug/m3",
          "component": "air_quality",
          "componentType": "generic"
        }, {
          "sensor": "MAR_02_20_PM001_1012",
          "description": "PM10 Sensor IMI 002",
          "dataType": "NUMBER",
          "type": "air_quality_pm10",
          "unit": "ug/m3",
          "component": "air_quality",
          "componentType": "generic",
          "additionalInfo": {
            "field1": "value1", "field2": "value2"
          }
        }
      ]
    }
  ]
}

```

(continues on next page)

(continued from previous page)

```
}  
  ]  
}  
[] }
```

Other examples

```
http://<your_api_server.com>/catalog?component=comp_demo&type=air_quality_pm10
```

```
http://<your_api_server.com>/catalog?componentType=air_quality&type=air_quality_pm10
```

Delete components / sensors

Description

This action allows the provider to delete catalog components and/or sensors.

WARNING: this operation performs a cascade delete and the execution of this action cannot be undone.

```
http://<your_api_server.com>/catalog/<provider_id>?<parameter>=<value>
```

Format	json
Method	DELETE, PUT
Permission	Writing
Return	No output data

Note that this action can be invoked using two HTTP methods: PUT and DELETE.

- DELETE will be used to delete all the sensors and components of a provider. It cannot contain body content.
- PUT will be used to delete a group of sensors or components. We should add a the parameter method with delete value to the request. In this case, the sensors or components to delete should be specified in the body message.

Parameters

The structure of the input parameters depends on whether you want to delete components or sensors.

The following describes the structure of the input in each case:

Delete components

Key	Description	Optional
components	Array of component identifiers to delete	Yes

Each element of the list corresponds to an identifier of a component to delete.

Delete sensors

Key	Description	Optional
sensors	Array of sensor identifiers to delete	Yes

Each element of the list corresponds to an identifier of a sensor to delete.

Response data

This action does not return additional data beyond the [HTTP status code](#).

Examples

Request to delete all components and sensors of a provider

To delete all components and sensors belonging to the provider named rec the request to do is the following:

```
DELETE http://<your_api_server.com>/catalog/rec
```

This request will delete in the catalog all the components and sensors of the rec provider

Request to delete a set of components of the catalog

To delete a set of components belonging to the provider rec the request to do is the following:

```
PUT http://<your_api_server.com>/catalog/rec?method=delete
```

in the body message:

```
{ "components": [ "COMP-3", "COMP-4" ] }
```

Request to delete a set of sensors of the catalog

To delete a set of sensors belonging to the provider rec the request to do is the following:

```
PUT http://<your_api_server.com>/catalog/rec?method=delete
```

in the body message:

```
{ "sensors": [ "RE001", "RE002", "RE003" ] }
```

Description

The catalog service allows to register or modify your own sensors/components or query the characteristics of a sensor or provider.

All requests for this service will have the following format:

```
http://<your_api_server.com>/catalog/<provider_id>
```

where provider_id is optional and should be included depending on the operation.

Actions

The available actions for this service are:

- Adding components / sensors
- Update data components / sensors
- Retrieve list of providers / sensors
- Remove components / sensors

Component types

The list of component types should be configured for each Sentilo instance, the following list could be used as a reference for any city:

Id	Name	Description
temperature	Temperature	Temperature measurement
noise	Soundmeter	Sound measurement
wind	Anemometer	Wind speed
humidity	Humidity	Humidity measurement
air_quality	Air Quality	Air Quality control
water_quality	Water Quality	Water Quality control
meteo	Meteorology	Weather Station
parking	Occupation parking	Parking control
luminosity	Luminosity	Luminosity measurement
glass_container	Occupancy container level	Glass occupancy container measurement
paper_container	Occupancy container level	Paper occupancy container measurement
plastic_container	Occupancy container level	Plastic occupancy container measurement
organic_container	Occupancy container level	Organic occupancy container measurement
refuse_container	Occupancy container level	Refuse occupancy container measurement
container_volum	Occupancy container level	Generic occupancy container measurement
soil_moisture	Soil moisture	Soil moisture measurement
park_meter	Parking meter	Parking meter control
traffic	Traffic	Traffic measurement
people_flow	People flow	Pedestrian flow measurement
flowmeter	Water flow	Water flow measurement
solenoid_valve	Electrovalve	Solenoid control
salinity	Salinity	Soil salinity measurement
internal_ambient_conditions	Internal Environmental Conditions	Temperature, humidity and luminosity measurement
external_ambient_conditions	External environmental conditions	Temperature, humidity and luminosity measurement
network_analyzer	Network analyzer	Electric network analyzer
gas_meter	Gas meter	Gas consumption meter
electricity_meter	Electricity meter	Electricity consumption meter
water_meter	Water meter	Water consumption meter
soil_sensor	Soil sensor	Soil mesurement of salinity, mosture, etc
generic	Generic component type	Default component type if not specified

Continued on next page

Table 1 – continued from previous page

Id	Name	Description
plugsense	Plug & Sense	Plug & Sense Libelium component

Sensor types

The list of sensor types should be configured for each Sentilo instance, the following list could be used as a reference for any city:

Id	Name	Description
temperature	Temperature	Temperature measurement
noise	Soundmeter Class II	Sound level measuring class II.
noise_class_i	Soundmeter Class I	Sound level measuring class I
anemometer	Anemometer	Wind Speed measuring
humidity	Humidity	Humidity measuring
parking	Occupation parking	Occupation parking control
luminosity	Luminosity	Luminosity measuring
container_volum	Occupancy container level	Occupancy container measurement
container_overturn	Container overturned	Container overturned indicator
container_open	Container open	Container opening indicator
status	Sensor status	Status control
battery	Battery level	Battery level measurement
soil_moisture_15	Soil moisture 15 cm.	Soil moisture measurement
soil_moisture_35	Soil moisture 35 cm.	Soil moisture measurement
park_meter	Parking meter	Parking meter control
vehicle_volume	Number of vehicles	Measurement of number of vehicles
vehicle_occupation_average	Average occupancy	Measurement of average occupancy in vehicles
vehicle_speed	Speed Vehicle	Vehicle speed measurement
air_quality_no2	NO2	Nitrogen dioxide measurement
air_quality_pm10	PM10	Measurement of suspension particles PM10
air_quality_pm25	PM25	Measurement of suspension particles PM25
air_quality_o3	O3	Ozone measurement
air_quality_so2	SO2	Sulfur dioxide measurement
air_quality_co	CO	Carbon Monoxide measurement
air_quality_co2	CO2	Carbon dioxide measurement
people_flow	People flow	Measurement of pedestrian flow
flowmeter	Water flow	Water flow measurement
solenoid_valve	Electrovalve	Solenoid actuator
eto	Evotranspiration	Evotranspiration measurement
salinity	Salinity	Soil salinity measurement
pluviometer	Pluviometer	Rain measurement
rain	Rain gauge	Rain indicator (it's raining/it's not raining)
wind	Wind gauge	Wind indicator (>X m/s)
wind_direction_6_m	Wind direction	Wind direction at 6 meters
wind_direction_10_m	Wind direction	Wind direction at 10 meters
voltage	Voltmetre	Electrical voltage measurement. Units: volts (V)
current	Ammeter	Electrical intensity measurement. Units amps (A)
frequency	Frequencymeter	Electrical frequency measurement. Units: herzs (Hz)
active_power	Active power	Active power measurement. Units: kilowatts (kW)
reactive_power	Reactive power	Reactive power measurement. Units: reactive kilovoltiamperis (kva)

Table 2 – continued from previous page

Id	Name	Description
cosphi	Power factor	Sensor that relates the active and reactive power. No units
active_energy	Active electrical energy meter	Measurement of accumulated active power. Units: kWh.
reactive_energy	Reactive electrical energy meter	Measurement of accumulated reactive power. Units: kvarh.
gas_volume	Gas meter	Measurement of accumulated gas consumption. Units: m ³ o Nm ³
water_meter	Water meter	Measurement of accumulated water consumption. Units: m ³ o l
global_solar_irradiance	Global solar irradiance	Mesurement of solar irradiance
leaf_moisture	Leaf moisture	Leaf wetness
oxygen	Oxygen	O ₂
vertical_level	Vertical level	Vertical liquid level (water)
bend	Bend	Bend
lpg	Lpg	Liquified petroleum gases (H ₂ , CH ₄ , ethanol & isobutane)
crack_detection	Crack detection	Crack detection gauge
solar_radiation	Solar radiation	Solar radiation
voc	Voc	Volatile Organic Compounds
chloride_ion	Chloride ion	Ion Cl ⁻
temperature	Temperature	Soil/Water temperature
magnesium_ion	Magnesium ion	Ion Mg ²⁺
distance	Distance	Distance (by metallic pressure or by pressure)
conductivity	Conductivity	Conductivity
liquid_leakage_line	Liquid leakage line	Water Leakage / Liquid Detection (Line)
crack_propagation	Crack propagation	Crack propagation gauge
nitrate_ion	Nitrate ion	Ion NO ₃
hall_effect	Hall effect	Hall effect
vibration	Vibration	Vibration (lamina)
copper_ion	Copper ion	Ion Cu ²⁺
calcium_ion	Calcium ion	Ion Ca ⁺
dendrometer	Dendrometer	Trunk, stem or fruit diameter
iodide_ion	Iodide ion	Ion I ⁻
bromide_ion	Bromide ion	Ion Br ⁻
sodium_ion	Sodium ion	Ion Na ⁺
linear_displacement	Linear displacement	Linear displacement
atmospheric_pressure	Atmospheric pressure	Atmospheric pressure
methane	Methane	CH ₄
pressure	Pressure	Pressure/ Weight
ammonia	Ammonia	NH ₃
redox_potential	Redox potential	Oxidation Reduction Potential
proximity_indoor	Proximity indoor	Ultrasound (indoor)
air_pollutant	Air pollutant	Air pollutants-I (NH ₃ , SH ₂ , ethanol and toluene) and air pollutants
liquid_leakage_point	Liquid leakage point	Water Leakage / Liquid Detection (Point)
proximity_outdoor	Proximity outdoor	Ultrasound (outdoor IP67)
potassium_ion	Potassium ion	Ion K ⁺
o_saturation	O saturation	Dissolved Oxygen
presence	Presence	Presence (PIR)
stretch	Stretch	Stretch
liquid_level	Liquid level	Horizontal liquid level (combustibles or water)
load	Load	Load
fluoride_ion	Fluoride ion	Ion F ⁻
ph	Ph	pH
solvent_vapors	Solvent vapors	Solvent vapors (H ₂ , CH ₄ , CO, ethanol and isobutane)

Table 2 – continued from previous page

Id	Name	Description
accelerometer	Accelerometer	Accelerometer

3.3.4 Data

Publish observations from a sensor

Description

This action allows a provider to publish the observations made by one of its sensors.

```
http://<your_api_server.com>/data/<provider_id>/<sensor_id>
```

Formats	json
Method	PUT
Permission	Writing
Retorna	No output data

Parameters

Key	Description	Optional
observations	Observations list to publish.	No
location	Geolocation coordinates in which the sensor got the observations(latitude longitude format).	Yes

Each observation will have the following structure:

Key	Description	Optional
value	Observation value to register	No
timestamp	Date and time when the observation was made (format dd/MM/yyyyTHH:mm:ssZ)	Yes
location	Geolocation coordinates, in decimal degrees, in which the sensor got the observations(latitude longitude format)	Yes

Please note the following:

- If you send an observation of a sensor without specifying timestamp, the platform will use the current timestamp as measurement time.
- The location of the observations is optional. But in case you want to set it, you can do it for all observations and/or individually for each one. The location informed for each observation takes precedence over the global localization.
- The TimeZone (Z) in the timestamps is optional. Its default value is UTC.
- In previous releases (up to 1.5.x) the system permitted publication of sensors that weren't registered in the catalog. Since 1.6, the sensor has to be correctly registered.

- If the sensor is in offline state, the server rejects the publication.

Response data

This action does not return additional data beyond the [HTTP status code](#).

Examples

Request to publish the last observation of a sensor

To publish an observation of a sensor service provides two ways to do it.

Abbreviated request

If we want to publish the last observation for the sensor with RE0012 identifier belonging to the provider with rec identifier we just have to add the value to the URL. In this case we can not send timestamp nor location:

```
http://<your_api_server.com>/data/rec/RE0012/12.3
```

where 12.3 is the observation value.

As indicated previously, this request will register a new observation (value 12.3) for the specified sensor. The timestamp of the observation will be the instant of arrival of the request to the platform.

Normal Request

If you wish to send the timestamp and the location of the observation too, we should use the format described, and send information in the body of the request.

For example, if in we want to include the timestamp of the observation, the request to do will be the following:

```
http://<your_api_server.com>/data/rec/RE0012
```

in the body message

```
{ "observations": [{  
  "value": "12.3",  
  "timestamp": "17/09/2012T12:34:45"}  
]}
```

This request will register a new observation(value 12.3) with the received timestamp (UTC time zone in this case) of the measurement.

Another example: it shows how to publish the temperature measured on Barcelona at a given time, sending the time in the Barceloca local time zone (CET):

```
{ "observations": [{  
  "value": "9.6",  
  "timestamp": "17/02/2016T11:43:45CET",  
  "location": "41.3888 2.15899"}  
]}
```

Request to publish several observations of the same sensor

If you want to send more of an observation of a sensor, the request is very similar to the previous one, only changing the message body.

```
http://<your_api_server.com>/data/rec/RE0012
```

in the body message

```
{ "observations": [{
  "value": "10.1"
}, {
  "value": "11.2",
  "timestamp": "17/09/2012T12:34:45"
}, {
  "value": "12.3",
  "timestamp": "17/09/2012T10:34:45"
}
]}
```

In this case are three observations with the corresponding timestamps.

Publishing observations from different sensors

Description

This action allows a provider to publish details of the observations made by more than one sensor in a single message.

```
http://<your_api_server.com>/data/<provider_id>
```

Format	json
Method	PUT
Permission	Writing
Return	No output data

Parameters

Key	Description	Optional
sensors	List of sensors (sensor) for which we publish at least one observation	No

Each sensor will have the following structure:

Key	Description	Optional
sensor	Sensor identifier	No
observations	Observations list (<i>observation</i>) to publish	No
location	Geolocation coordinates in which the sensor observations are obtained (latitude longitude format)	Yes

Each observation will have the structure described on page [Publish observations of a sensor](#):

Key	Description	Optional
value	Observation value	No
timestamp	Date and time at which the observation was made (dd/MM/yyyyTHH:mm:ssZ format)	Yes
location	Geolocation coordinates in which the sensor has achieved this observation (latitude longitude format).	Yes

Response Data

This action does not return additional data beyond the [HTTP status code](#).

Examples

Request to send multiple observations of several sensors setting a LTC TimeZone

If we want to send the observations of a set of sensors for the provider named rec, setting timeZone to CET, the request to do is:

```
http://<your_api_server.com>/data/rec
```

and in the body message:

```
{ "sensors": [
  {
    "sensor": "RE0012",
    "observations": [
      { "value": "1.1" },
      { "value": "1.2",
        "timestamp": "17/09/2012T12:34:45CET" },
      { "value": "1.3",
        "timestamp": "17/09/2012T10:34:45CET" }
    ]
  }, {
    "sensor": "RE0013",
    "location": "41.12345 2.12354",
    "observations": [
      { "value": "2.1" },
      { "value": "2.2",
        "timestamp": "16/09/2012T15:43:21CET" },
      { "value": "2.3",
        "timestamp": "16/09/2012T10:43:21CET" }
    ]
  }
]
```

Delete Observations

Description

This action allows to delete observations made by one or several sensors of a provider.


```
http://<your_api_server.com>/data/<provider_id>/<sensor_id>
```

Formats	json
Method	DELETE
Permission	Writing
Returns	No output data

Parameters

No additional data is sent.

Response data

This action does not return additional data beyond the [HTTP status code](#).

Examples

Request to delete the last observation of a sensor

If we want delete the last observation received by the platform of the sensor with id REC1102 of the provider named rec, the request to do is:

```
http://<your_api_server.com>/data/rec/RE0012
```

Request to delete the last observations of a provider's sensors

If we want to delete the last observation of each sensor of the provider named rec, the request to do is:

```
http://<your_api_server.com>/data/rec
```

Retrieve sensor observations

Description

This action allows you to retrieve the latest observations of a sensor. In addition, the service can also permit to specify search criteria to retrieve observations: filter by a given time period and / or to indicate the maximum number of observations to be retrieved.

```
http://<your_api_server.com>/data/<provider_id>/<sensor_id>?<parameter>=<value>
```

Format	json
Method	GET
Permission	Reading
Returns	Observations list

Parameters

Key	Description	Optional
from	Indicates the beginning of the time period for which you want to retrieve observations	Yes
to	Indicates the end of the time period for which you want to retrieve observations	Yes
limit	Indicates the number of observations to retrieve	Yes

Please, note the following:

- The maximum number of records returned will be fixed by the platform settings. If the parameter passed is higher, the number of records returned will be equals to the maximum value configured in the platform.
- If the limit parameter is not set, only one observation will be returned.
- All dates must have the following format: dd/MM/yyyyTHH:mm:ssZ with Z as optional (and with default value UTC)

Response data

As mentioned, in addition to [HTTP status code](#), the observation data is returned in the body contents as a list of observations:

Key	Description	Optional
observations	List the observations (observation)	No

Each observation has the following structure:

Key	Description	Optional
value	Observation value	No
timestamp	The time when the observation was made based on UTC (dd/MM/yyyyTHH:mm:ss format)	No
time	The time when the observation was made in milliseconds	No
location	Geolocation coordinates in which the sensor was recorded observation	Yes

Examples

Request to retrieve the latest observations of a sensor based on a date

The following request shows an example in which a call is made to retrieve the last 20 observations of the sensor with RE0012 identifier of the provider named rec which have been registered from 10/01/2013.

```
http://<your_api_server.com>/data/rec/RE0012?limit=20&from=10/01/2013T10:00:00
```

As response we receive:

```
{ "observations": [  
  {  
    "value": "28.61132406103821",  
    "timestamp": "13/11/2017T09:00:00",  
    "time": 1510563600000
```

(continues on next page)

(continued from previous page)

```

    }, {
      "value": "20.795568440010314",
      "timestamp": "13/11/2017T08:30:00",
      "time": 1510561800000
    }, {
      "value": "91.01094902496055",
      "timestamp": "13/11/2017T08:30:00",
      "time": 1510561800000
    }, {
      "value": "62.22915604583776",
      "timestamp": "11/01/2013T08:16:38",
      "time": 1510561800000
    }, {
      "value": "99.96065618303348",
      "timestamp": "11/01/2013T07:16:38",
      "time": 1510561800000
    }, {
      "value": "94.95685904585568",
      "timestamp": "11/01/2013T06:16:38",
      "time": 1510561800000
    }, {
      "value": "51.26506022800391",
      "timestamp": "11/01/2013T05:16:38",
      "time": 1510561800000
    }, {
      "value": "21.43303677241535",
      "timestamp": "11/01/2013T04:16:38",
      "time": 1510561800000
    }, {
      "value": "55.6601921120059",
      "timestamp": "11/01/2013T03:16:38",
      "time": 1510561800000
    }, {
      "value": "56.692086830598996",
      "timestamp": "11/01/2013T02:16:38",
      "time": 1510561800000
    }
  ]
}
```

Request to retrieve the last observation of a sensor

If you only want to retrieve the last observation of the RE0012 sensor, the request to do is:

```
http://<your_api_server.com>/data/rec/RE0012
```

As response we will receive:

```

{"observations": [{
  "value": "11.5",
  "timestamp": "18/09/2012T17:20:00",
  "time": 1510561800000}
]}
```

Read observations from provider's sensors

Description

This action allows to retrieve the latest observations of the sensors of a provider. In addition, the service can also specify search criterias to retrieve observations: filter by a given time period and / or to indicate the maximum number of observations to be recovered.

```
http://<your_api_server.com>/data/<provider_id>?<parameter>=<value>
```

Format	json
Method	GET
Permission	Reading
Return	List with the observations from provider's sensors

Because the number of sensors from a supplier can be very high, and therefore the amount of information returned can be very large, be careful using this operations due to performance reasons.

Parameteres

Key	Description	Optional
from	Indicates the beginning of the time period for which you want to retrieve the observations.	Yes
to	Indicates the end of the time period for which you want to retrieve the observations.	Yes
limit	Specifies the maximum number of observations for each sensor to recover.	Yes

Please note the following:

- The maximum number of records returned will be fixed by the platform settings. If the parameter passed is higher, the number of records returned will be equalsa to the maximum value configured in the platform.
- If the limit parameter is not set, only one record will be returned.
- All dates must have the following format: dd/MM/yyyyTHH:mm:ss

Response data

In addition to the [HTTP status code](#), the observation data is returned in the body contents as a list of observations:

Key	Description	Optional
sensor	List of sensors (sensor) for the observations that have been retrieved	No

Each sensor has the following structure:

Key	Description	Optional
sensor	Sensor identifier	No
observations	List of the latest sensor observations	No

Finally, each observation (observation) has the following structure:

Key	Description	Optional
value	Observation value	No
timestamp	The time at which the observation was made (dd/MM/yyyyTHH:mm:ss format)	No
time	The time when the observation was made in milliseconds	No
location	Geolocation coordinates in which the sensor was recorded observation	Yes

Examples

Request to retrieve the latest observations from a provider after a given date

If we want to retrieve the latest observations of the sensors associated with the provider named rec from a given date we should make the following request:

```
http://<your_api_server.com>/data/rec?from=10/09/2012T10:00:00
```

As response we will receive:

```
{ "sensors": [
  {
    "sensor": "RE0012",
    "observations": [
      {
        "value": "1",
        "timestamp": "10/09/2012T10:05:00",
        "time": 1510561800000
      }, {
        "value": "1.2",
        "timestamp": "10/09/2012T07:05:00",
        "time": 1510561800000
      }
    ]
  }, {
    "sensor": "RE0013",
    "observations": [
      {
        "value": "24",
        "timestamp": "10/09/2012T10:06:10",
        "time": 1510561800000
      }
    ]
  }
] }
```

Request to retrieve the latest observations from rec provider

If you only want to retrieve the last observation of the RE0012 sensor, the request to do is:

```
http://<your_api_server.com>/data/rec
```

As response we will receive:

```
{ "sensors": [
  {
    "sensor": "RE0012",
```

(continues on next page)

(continued from previous page)

```
    "observations":
    [{
      "value": "1",
      "timestamp": "10/09/2012T10:05:00",
      "time": 1510561800000
    }]
  }, {
    "sensor": "RE0013",
    "observations":
    [{
      "value": "24",
      "timestamp": "10/09/2012T10:06:10",
      "time": 1510561800000
    }]
  }
}
```

Description

The data service allows to read, write or delete the observations of the registered sensors.

All requests for this service will have the following format:

```
http://<your_api_server.com>/data/<provider_id>/<sensor_id>
```

where **<provider_id>** and **<sensor_id>** correspond to the sensor and provider identifiers on which we want to perform the requested action.

Actions

The available actions for this service are:

- Publish observations of a sensor
- Publish observations from sensors of a provider
- Delete observations
- Read observations from a sensor
- Read observations from sensors of a provider

3.3.5 Order

Publish an order

Description

This operation allows to send an order to single sensor or to all sensors of a provider. Once the system receives the order, it sends a notification to all its subscribers.

```
http://<your_api_server.com>/order/<provider_id>/<sensor_id>
```

Format	json
Method	PUT
Permission	Writing
Return	No output data

Parameters

Each order will have its specific structure with its associated information in the defined format (JSON).

The platform will only transfer the information to the subscribers, without checking its contents nor reading into it.

Key	Description	Optional
order	Orden content	Not

Response data

This action does not return additional data beyond the [HTTP status code](#).

Examples

Publish an order to a sensor/actuator

The following example shows how to send a request to the platform to publish a new order destined to the sensor with RE0012 identifier belonging to the provider with ID rec:

```
http://<your_api_server.com>/order/rec/RE0012
```

in the body message:

```
{"order": "Stop"}
```

Publish an order to all the provider's sensors/actuators

The following example shows how to send a request to the platform to publish a new order to all the sensors belonging to the provider with rec identifier:

```
http://<your_api_server.com>/order/rec
```

in the body message

```
{"order": "Start RE0012, RE0013"}
```

Retrieve orders

Description

This action allows you to retrieve the last orders associated with a sensor or provider. In addition, we can also specify search criteria to retrieve the orders: filter by a given time period and/or indicate the maximum number of orders that you want to retrieve.

```
http://<your_api_server.com>/order/<provider_id>/<sensor_id>?<parameter>=<value>
```

Format	json
Method	GET
Permission	Read
Returns	List of orders destined to sensor or provider listed

Parameters

Key	Description	Optional
from	Indicates the beginning of the time period for which you want to retrieve orders.	Yes
to	Indicates the ending of the time period for which you want to retrieve orders.	Yes
limit	Specifies the maximum number of orders to retrieve.	Yes

Please, note the following:

- The maximum number of records returned will be fixed by the platform settings. If the parameter passed is higher, the number of records returned will be the configured in the platform.
- If the limit parameter is not set, only one record will be returned.
- All dates must have the following format: dd/MM/yyyyTHH:mm:ss

Response data

As mentioned, in addition to [HTTP status code](#), the requested data is returned in the body contents as a list of orders.

The response structure depends on what we are retrieving, orders from a sensor or a provider.

Last orders for a sensor

Key	Description	Optional
orders	List with the last sensor's order	No

Each order will have the following structure:

Key	Description	Optional
order	Order message recorded at the time the order was published	No
timestamp	The time when the order was made (dd/MM/yyyyTHH:mm:ss format)	No
sender	Entity identifier that issued the order.	No
time	The time when the observation was made in milliseconds	No

Last orders for provider

Key	Description	Optional
sensors	List with sensors (sensor)	No

Each (**sensor**) will have the following structure:

Key	Description	Optional
sensor	Sensor identifier	No
orders	List with the last orders for the sensor	No

Finally, each command (**order**) will have the structure that we have defined previously.

Key	Description	Optional
order	Order message recorded at the time the order was published	No
timestamp	The time when the order was made (dd/MM/yyyyTHH:mm:ss format)	No
sender	Entity identifier that issued the order.	No

Examples

Retrieve the last order for a sensor

To retrieve the last order for the sensor with RE0012 identifier belonging to the provider named rec, we do the following request:

```
http://<your_api_server.com>/order/rec/RE0012
```

As response we will get:

```
{ "orders": [{
  "order": "Shutdown",
  "timestamp": "21/03/2013T14:25:39",
  "sender": "app_demo_provider"
}] }
```

Retrieve the last N orders for a sensor

If we want to retrieve more than one order, we can specify the number of records to retrieve with the following request:

```
http://<your_api_server.com>/order/rec/RE0012?limit=3
```

As response we will get:

```
{ "orders":
  [{
    "order": "Shutdown",
    "timestamp": "21/03/2013T14:25:39",
    "sender": "app_demo_provider",
    "time": 1510570798597
  }, {
```

(continues on next page)

(continued from previous page)

```
    "order": "Start",
    "timestamp": "20/03/2013T23:59:59",
    "sender": "app_demo_provider",
    "time": 1510570798597
  }, {
    "order": "Shutdown",
    "timestamp": "20/03/2013T14:25:39",
    "sender": "app_demo_provider",
    "time": 1510570798597
  }
]
```

Retrieve the last N orders for a sensor between dates

If we want to retrieve orders for a sensor between two dates, we should do the following request:

```
http://<your_api_server.com>/order/rec/RE0012?limit=3&from=19/03/2013T00:00:00&to=20/
↪03/2013T23:59:59
```

As response we will get:

```
{ "orders":
  [ {
    "order": "Start",
    "timestamp": "20/03/2013T23:59:59",
    "sender": "app_demo_provider",
    "time": 1510570798597
  }, {
    "order": "Shutdown",
    "timestamp": "20/03/2013T14:25:39",
    "sender": "app_demo_provider",
    "time": 1510570798597
  }
] }
```

Retrieve the last orders for a provider

All the previous examples are focused on recovering the last command of a sensor, but the service also allows you to search the latest orders destined for all the sensors of provider.

In this case, we only specify the provider, and the request will be:

```
http://<your_api_server.com>/order/rec2
```

As response we get a list of sensor elements, and each one will contain its last orders.

```
{ "sensors":
  [ {
    "sensor": "RE0012",
    "orders":
      [ {
        "order": "Shutdown",
        "timestamp": "21/03/2013T14:25:39",
```

(continues on next page)

(continued from previous page)

```

        "sender": "app_demo_provider",
        "time": 1510570798597
    }]
}, {
    "sensor": "RE0013",
    "orders":
    [{
        "order": "Shutdown",
        "timestamp": "21/03/2013T14:25:39",
        "sender": "app_demo_provider",
        "time": 1510570798597
    }]
}, {
    "sensor": "RE0014",
    "orders":
    [{
        "order": "Shutdown",
        "timestamp": "21/03/2013T14:25:39",
        "sender": "app_demo_provider",
        "time": 1510570798597
    }]
}]
}

```

Description

The order service allows to send or retrieve orders to sensors/actuators.

All requests for this service will have the following format:

```
http://<your_api_server.com>/order/<provider_id>/<sensor_id>
```

The sensor identifier, **<sensorId>**, is optional and should be informed depending on the action we want to execute.

Actions

The available actions for this service are:

- Publish orders
- Retrieve orders

3.3.6 Subscription

Subscription to sensor data

Description

This action allows to subscribe to observation data associated to sensors.

It's important to note that we only can subscribe to the sensor data over we own read permission.

```
http://<your_api_server.com>/subscribe/data/<provider_id>/<sensor_id>
```

Format	json
Method	PUT
Permission	Read
Returns	No additional data returned

Parameters

Key	Description	Optional
endpoint	URL where the platform will send a HTTP request with the observation data	No
secret-Call-backKey	Secret key for callbacks	Yes
retries	Maximum number of retries	Yes
re-retries_delay	Delay parameter in minutes. Delays are spaced exponentially according to following equation: delay (N) = delay * 2 ^(N-1) Where N is the current entry turn. More detailed explanation follows.	Yes

Retries

In case the remote endpoint is down or does not respond with an success HTTP 2xx code, Sentilo can try to resend the data later. In order to overcome major number of remote outages, Sentilo sends the data in delay times that are exponential according to equation:

```
delay (N) = delay * 2(N-1)
```

For example, if we have a subscription configured with 5 retries and 10 minutes, first retry would occur at 10 minutes, the second 20 minutes after the first, the third 40 minutes after the second, etc up to the fifth retry.

The total time used for the 5 retries would occur in 10+20+40+80+160=310 minutes after the first failed intent.

Response data

This action does not return additional data beyond the [HTTP status code](#).

Examples

Request to subscribe to a sensor's data

If we want to subscribe to the sensor's data with RE0012 identifier of the provider named rec, the request will be:

```
http://<your_api_server.com>/subscribe/data/rec/RE0012
```

and the body message:

```
{"endpoint": "http://<your_endpoint_notification_server.com>/resource"}
```

Request to subscribe to provider's data

If we want to subscribe to all the sensors belonging to a provider, the request will be:

```
http://<your_api_server.com>/subscribe/data/rec
```

and the body message:

```
{ "endpoint": "http://<your_endpoint_notification_server.com>" }
```

Subscription to orders

Description

This action allows to subscribe to orders associated to sensors.

It's important to note that we only can subscribe to the sensor data over we own read permission.

```
http://<your_api_server.com>/subscribe/order/<provider_id>/<sensor_id>
```

Formats	json
Method	PUT
Permission	Read
Returns	No additional data returned

Parameters

Key	Description	Optional
endpoint	URL where the platform will send a HTTP request with the order data	No
secret-Call-backKey	Secret key for callbacks	Yes
retries	Maximum number of retries	Yes
retries_delay	Delay parameter in minutes. Delays are spaced exponentially according to following equation: delay (N) = delay * 2^(N-1) Where N is the current entry turn. More detailed explanation follows.	Yes

Retries

In case the remote endpoint is down or does not respond with an success HTTP 2xx code, Sentilo can try to resend the data later. In order to overcome major number of remote outages, Sentilo sends the data in delay times that are exponential according to equation:

```
delay (N) = delay * 2^(N-1)
```

For example, if we have a subscription configured with 5 retries and 10 minutes, first retry would occur at 10 minutes, the second 20 minutes after the first, the third 40 minutes after the second, etc up to the fifth retry.

The total time used for the 5 retries would occur in 10+20+40+80+160=310 minutes after the first failed intent.

Response data

This action does not return additional data beyond the [HTTP status code](#).

Examples

Request to subscribe to orders for a sensor

If we want to subscribe to the orders for the sensor with RE0012 identifier of the provider named rec, the request will be:

```
http://<your_api_server.com>/subscribe/order/rec/RE0012
```

and the body message:

```
{"endpoint": "http://<your_endpoint_notification_server.com>"}
```

Request to subscribe to orders for a provider

If we want to subscribe to all the sensor's orders belonging to the rec provider, the request will be:

```
http://<your_api_server.com>/subscribe/order/rec
```

and like body message:

```
{"endpoint": "http://<your_endpoint_notification_server.com>"}
```

Subscription to alerts

Description

This action allows to subscribe to alarms associated to sensors.

It's important to note that we only can subscribe to the sensor data over we own read permission.

```
http://<your_api_server.com>/subscribe/alarm/<alert_id>
```

Format	json
Method	PUT
Permission	Read
Returns	No additional data returned

Parameters

Key	Description	Optional
endpoint	URL where the platform will send a HTTP request with the alarm message	No
secret-Call-backKey	Secret key for callbacks	Yes
retries	Maximum number of retries	Yes
re-retries_delay	Delay parameter in minutes. Delays are spaced exponentially according to following equation: delay (N) = delay * 2 ^(N-1) Where N is the current entry turn. More detailed explanation follows.	Yes

Retries

In case the remote endpoint is down or does not respond with an success HTTP 2xx code, Sentilo can try to resend the data later. In order to overcome major number of remote outages, Sentilo sends the data in delay times that are exponential according to equation:

$$\text{delay (N)} = \text{delay} * 2^{(N-1)}$$

For example, if we have a subscription configured with 5 retries and 10 minutes, first retry would occur at 10 minutes, the second 20 minutes after the first, the third 40 minutes after the second, etc up to the fifth retry.

The total time used for the 5 retries would occur in 10+20+40+80+160=310 minutes after the first failed intent.

Response data

This action does not return additional data beyond the [HTTP status code](#).

Examples

Request to subscribe to alert's alarms

If we want to register a new subscription for alarms belonging to the alert with alert1 identifier, the request will be:

```
http://<your_api_server.com>/subscribe/alarm/alert1
```

and the body message:

```
{ "endpoint": "<your_endpoint_notification_server.com>" }
```

Retrieve active subscriptions

Description

This action allows to retrieve the list of all our active subscriptions. Additionally, we can retrieve only subscriptions from a specific type.

```
http://<your_api_server.com>/subscribe/<event_type>
```

Format	json
Method	GET
Permission	Read
Returns	Active subscriptions

`<event_type>` is optional and allows to filter the subscription by type.

Parameters

No additional parameters can be used.

Response data

This action, additionally to the [HTTP status code](#), will return a list of our active subscriptions:

Key	Description	Optional
subscriptions	List with all our active subscriptions	No

Each **subscription** element contains this set of attributes:

Key	Description	Optional
endpoint	URL defined in the subscription	No
type	Event type related to the subscription(data, order o alarm)	No
provider	In case the type is <i>data</i> or <i>order</i> this attribute contains the provider identifier	Yes
sensor	In case the type is <i>data</i> or <i>order</i> this attribute contains the sensor identifier	Yes
alarm	In case the type is <i>alarm</i> this attribute contains the alert identifier	Yes

Examples

Request to retrieve all active subscriptions

```
http://<your_api_server.com>/subscribe
```

As response we will obtain:

```
{
  "subscriptions":
  [{
    "endpoint":"http://<your_endpoint_notification_server.com>",
    "type":"ALARM",
    "alert":"alert1"
  }, {
    "endpoint":"http://<your_endpoint_notification_server.com>",
    "type":"DATA",
    "provider":"app_demo_provider",
    "sensor":"appdemo_sensor5_test"
  }, {
    "endpoint":"http://<your_endpoint_notification_server.com>",
    "type":"DATA",
```

(continues on next page)

(continued from previous page)

```

    "provider": "app_demo_provider",
    "sensor": "appdemo_sensor_test"
  }, {
    "endpoint": "http://<your_endpoint_notification_server.com>",
    "type": "ALARM", "alert": "11"
  }
]
}

```

Request to retrieve active subscriptions for a specific type

If we want to retrieve only the subscriptions to a specific event type:

```
http://<your_api_server.com>/subscribe/alarm
```

As response we will obtain:

```

{
  "subscriptions":
  [
    {
      "endpoint": "http://<your_endpoint_notification_server.com>",
      "type": "ALARM",
      "alert": "alert1"
    },
    {
      "endpoint": "http://<your_endpoint_notification_server.com>",
      "type": "ALARM",
      "alert": "alert11"
    }
  ]
}

```

Cancel subscriptions

Description

This action allows to cancel any or a set of our active subscriptions.

```
http://<your_api_server.com>/subscribe/<event_type>/<resource_id>
```

Format	json
Method	DELETE
Permission	Write
Returns	No additional data returned

<event_type> and **<resource_id>** are optional and allow to filter the subscription to cancel by event type or related resource.

Parameters

No additional data can be sent.

Response data

This action does not return any additional data beyond the [HTTP status code](#).

Examples

Request to cancel subscriptions

If we want to cancel all our active subscriptions, the request will be:

```
http://<your_api_server.com>/subscribe
```

Request to cancel subscriptions for a specific event type

If we want to cancel all our active subscriptions of a specific event type like order, the request will be:

```
http://<your_api_server.com>/subscribe/order
```

Request to cancel subscriptions for a specific resource

If we want to cancel all our active data subscriptions of a specific sensor like RE0012 belonging to the rec provider, the request will be:

```
http://<your_api_server.com>/subscribe/data/rec/RE0012
```

Description

The subscription service allows to the platform clients(application/modules or provider/sensors) to subscribe to system events, which can be:

- **Data:** related to data observations received by the platform
- **Order:** related to orders received by the platform
- **Alarm:** related to alarms received by the platform

It is also possible to retrieve the list of active subscriptions or cancel them.

All requests for this service will have the following format:

```
http://<your_api_server.com>/subscribe/<event_type>/<resource_id>
```

where **<resource_id>** identifies the system resource to which the request applies (providers, sensors or alerts).

Actions

The available actions for this service are:

- [Subscription to sensor data](#)
- [Subscription to orders](#)
- [Subscription to alerts](#)

- Retrieve active subscriptions
- Cancel subscription

Notifications

As mentioned before, when we subscribe to a system event, the platform will send us a notification (push process), whenever the event occurs, through a HTTP POST request to the URL configured with the subscription.

The notification message follows the following structure:

```
{
  "message": "...",
  "timestamp": "...",
  "topic": "...",
  "type": "...",
  "sensor": "...",
  "provider": "...",
  "location": "...",
  "alert": "...",
  "alertType": "...",
  "time": "...",
  "tenant": "...",
  "publisher": "...",
  "publisherTenant": "...",
  "publishedAt": "..."
}
```

where the following fields are mandatory:

- **message**: contains the event information (observation, alarm or order)
- **timestamp**: contains the timestamp associated with the event, formatted as UTC (dd/MM/yyyy'T'HH:mm:ss).
- **topic**: identifies the subscription related to the event.
- **type**: identifies the event type (DATA, ORDER or ALARM)
- **time**: same as timestamp but expressed as milliseconds

and the following are optional and depend on the event type:

- **sensor**: contains the sensor identifier related to the event.
- **provider**: contains the provider identifier related to the event.
- **location**: only added in observation notifications when the location is filled in.
- **sender**: this field has been removed in version 1.6. See *publisher* field.
- **alert**: only added in alarm notifications. Contains the alert identifier related to the alarm.
- **alertType**: only added in alarm notifications. Contains the alert type: INTERNAL or EXTERNAL.
- **retryAttempt**: if the delivery of the message fails, this number indicates a number of the retries. See for example [how to define retries in data subscription](#).
- **publisher**: identifies the entity who has published the event.
- **publishedAt**: this field differs from *time* field in that it always stores the time when the event was published on Sentilo.
- **tenant**: only added in multitenant instances. This field identifies the tenant to which the event belongs.

- `publisherTenant`: only added in multitenant instances. This field identifies the tenant to which the publisher belongs.

Here are three different examples of notification:

```
{
  "message": "8",
  "timestamp": "26/10/2016T08:50:33",
  "topic": "/data/app_demo_provider/appdemo_sensor_test",
  "type": "DATA",
  "provider": "app_demo_provider",
  "sensor": "appdemo_sensor_test",
  "retryAttempt": 1,
  "publisher": "app_demo_provider",
  "time": 1477471833000,
  "publishedAt": 1477471833000
}
```

```
{
  "message": "Stop",
  "timestamp": "16/10/2013T15:39:11",
  "topic": "/order/app_demo_provider",
  "type": "ORDER",
  "provider": "app_demo_provider",
  "publisher": "app_demo_provider",
  "time": 1477471833000,
  "publishedAt": 1477471833000
}
```

```
{
  "message": "Value greater than 34",
  "timestamp": "16/10/2013T15:40:57",
  "topic": "/alarm/internalAlarmProve",
  "type": "ALARM",
  "sensor": "app_demo",
  "alert": "ALERT_GT14",
  "alertType": "INTERNAL",
  "publisher": "sentilo",
  "time": 1477471833000,
  "publishedAt": 1477471833000
}
```

If the subscription has included a secret key, the following messages will include the **security headers** ([see more](#)).

Notifications to untrusted HTTPS

In case that remote endpoint uses a self-signed certificate, add the following configuration in the `config.properties` of the `sentilo-platform-server`:

```
#Allows Sentilo to send notifications to untrusted servers, i.e., servers with self-  
signed certificates or signed by unknown CAs  
api.subs.ssl.no-validate-certificates=false
```

The Application Programming Interface (API) of Sentilo defines a set of commands, functions and protocols that must be followed by who wants to interact with the system externally.

This area defines the Application Programming Interface (API), that any sensor or application must use to interact with the platform.

The starting capacities of the platform related to its external interface are:

- Allows to register applications/modules and providers/sensors in the platform (Catalog).
- Allow to applications/modules and sensors subscribe to services defined in the catalog as well as post events occurring (Publish/Subscribe).
- Allow you to send information from sensors to applications/modules (Data).
- Allows to send orders from applications/modules to sensors (Order).

4.1 Sentilo platform

Sentilo is a platform aimed to isolate and communicate the applications that are developed to exploit the information generated from the ground by the layer of sensors deployed across to collect and broadcast this information.

Its main modules are:

- Restful API
- Web Application which provides an administration console and some public visualizers
- Data publication & subscription system
- A memory database for storing real time data
- A non-SQL database for storing less volatile data, like the sensor's catalog
- Several agents which extend the platform features

4.1.1 Key Concepts

This section describes the main concepts of Sentilo. Many of these concepts are discussed later deeply.

You can read also some [Technical FAQs](#).

PubSub Platform

Sentilo allows customers to publish and retrieve information and to subscribe to system events. This module is a stand-alone Java process that uses Redis as a publish/subscribe mechanism.

The different types of information considered are:

- observations
- alarms

- orders

Please, check this out fore more [info](#).

RealTime storage

Primary repository where the platform stores all the information received. It is configured to do periodic backups in the file system. It is also the Publish/Subscribe engine.

REST API

The client's communication with publish/subscription mechanism is made using the REST API provided by the platform.

Services offered by the API can be classified into five main groups:

- **data**: provides operations to publish, retrieve, delete data.
- **order**: provides operations to publish, retrieve, delete orders.
- **alarm**: provides operations to publish, retrieve, delete alarms.
- **subscribe**: provides operations to subscribe, retrieve and cancel subscriptions.
- **catalog**: provides operations to insert, update, query and delete catalog resources (sensors, components and alerts).

By default, the information is transmitted using JSON format. Please, check this out fore more [info](#).

Agents

Agents are Java processes that expand the core functionality of the platform through a Plug & Play system using the Redis publish and subscribe mechanism.

Sentilo currently provides several [agents](#):

- **Relational database agent**: used to export historical data to a relational database.
- **Alert agent**: used for processing each data received by the platform and validate it with the business rules configured in the catalog.
- **Activity Monitor Agent**: used for upload the events to Elasticsearch.
- **Historian Agent**: used for upload the events to OpenTSDB.

Authentication Token

The invocation of different REST API services is secured using an authentication token. This token must be sent in every request as a header parameter of the HTTP request named IDENTITY_KEY. This token is unique for each provider or client application, and is managed by the catalog application.

Please, check this out fore more [info](#).

Permission

Permissions allow Sentilo to identify the requester and to ensure that who makes a request is authorized to do it. Permissions are managed by the catalog web app and allow to configure read or write permissions to client application on third party resources (provider or client applications). By default, every platform entity has read and write permissions on its own resources.

Please, check this out fore more [info](#).

Notification mechanism

Sentilo provides two mechanisms for nofitying events:

- If the client is capable of having an opened socket, the platform will send a notification to this socket every time an event is triggered ([push](#)).
- If the client cannot have an opened socket, then it must be doing periodic requests ([polling](#)) to the platform to retrieve last events.

Catalog

The Web Application Platform console allows to manage the following resources: providers, applications, components, sensors, sensors types, component types, alerts and users.

It also provides a public console for displaying components and sensors registered in the platform as well as the data that has been received.

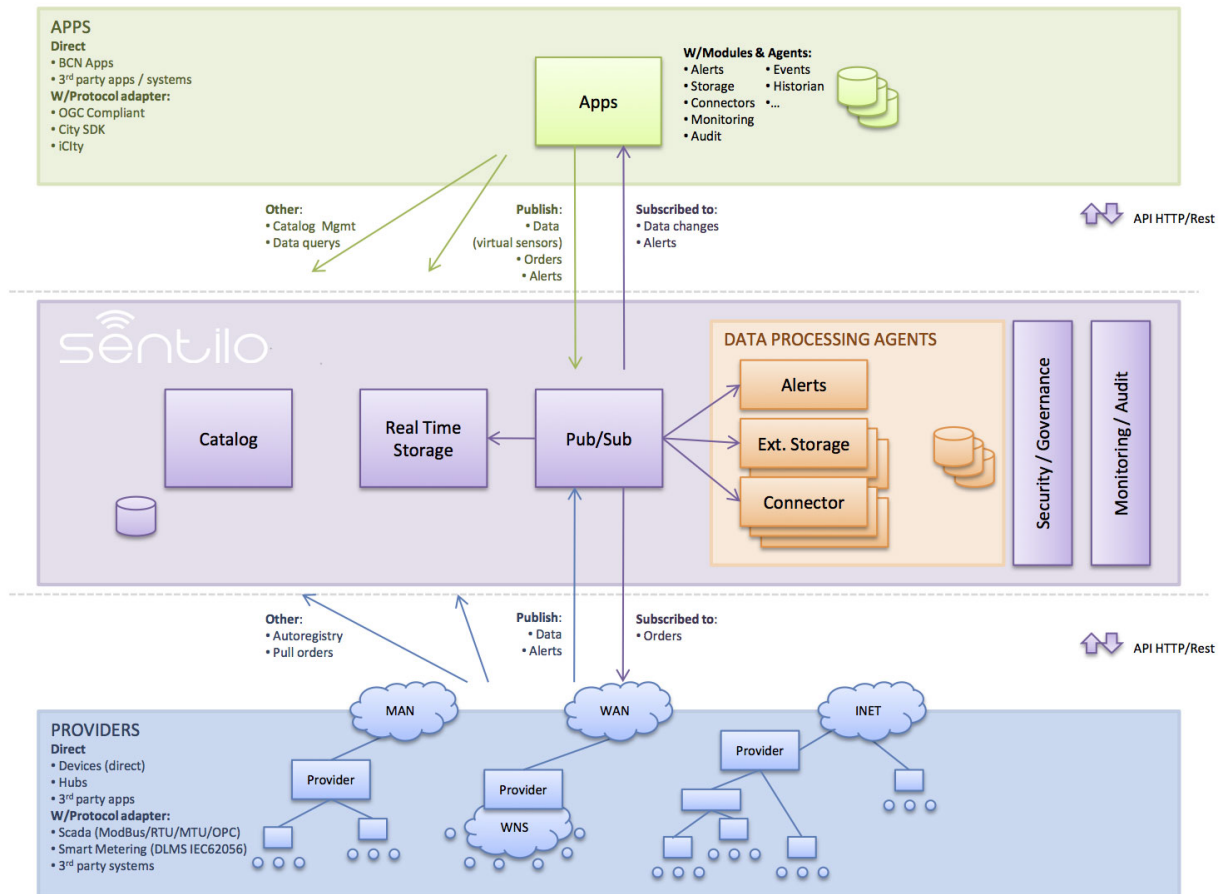
Please, check this out fore more [info](#).

Alert

Sentilo allows to manage sensor-level internal alerts, aimed to control the validity of the data received. The set of conditional operators available are: `>`, `>=`, `<`, `<=`, `=`, **any change**, **variation**, **frozen**. When the value received from a sensor doesn't met any of the conditions defined, the alert agent publishes an event (alarm) notifying it. These alerts are defined through the [console](#). There are also external alerts which can be defined and triggered externally though the [API](#).

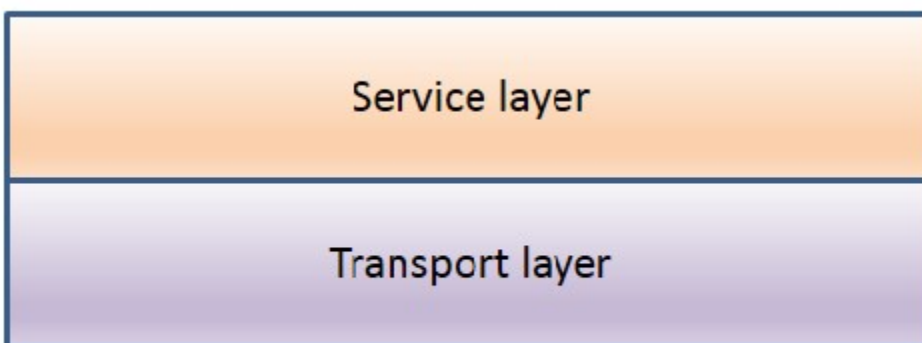
4.1.2 Platform architecture

The following diagram describes the Sentilo platform:



PubSub Server

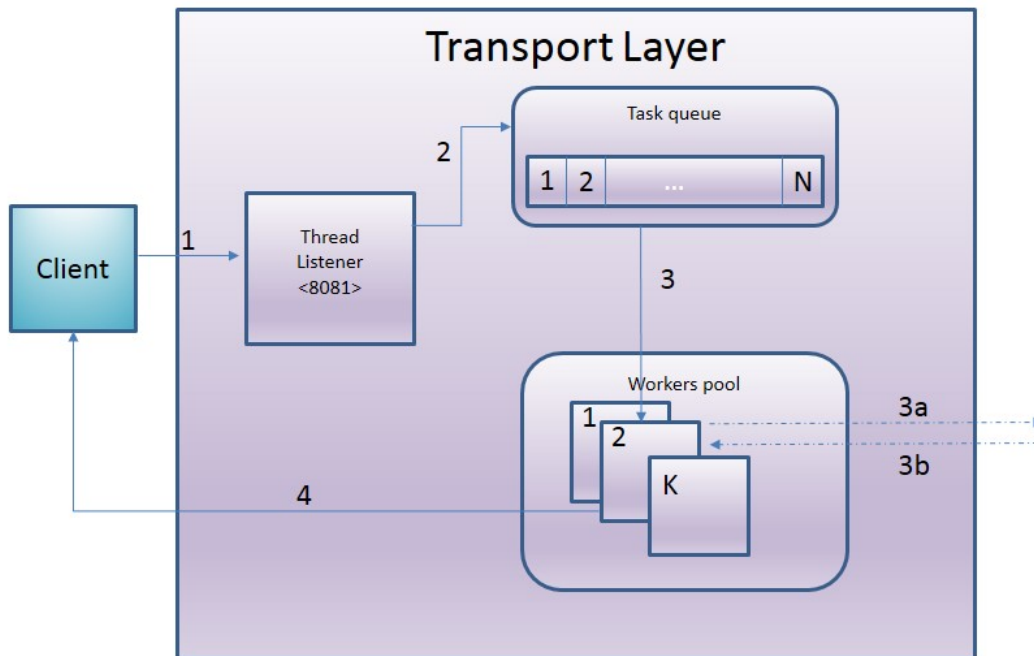
PubSub Server is a stand-alone java application whose design is divided into two layers:



- Transport Layer: designed following the Thread Pool pattern: http://en.wikipedia.org/wiki/Thread_pool_pattern
- Service Layer: Based in Spring and Redis, it's designed to provide high performance rates.

Transport Layer

The transport layer is designed following the Thread Pool pattern and is implemented with Apache HttpCore library. The following diagram shows the main flow for a request within this layer:



- The client sends a Http request to the REST platform
- The server accepts and queues it on the list of pending requests
- When a Worker is available, a pending task is assigned to it for processing (removing it from the queue)
 - delegates the request to an element of the service layer
 - and constructs the HTTP response from the information received
- Send the response to client's request

The values of the job queue and the workers' pool are fully configurable via properties file, for easily adjust to the load requirements of each environment:

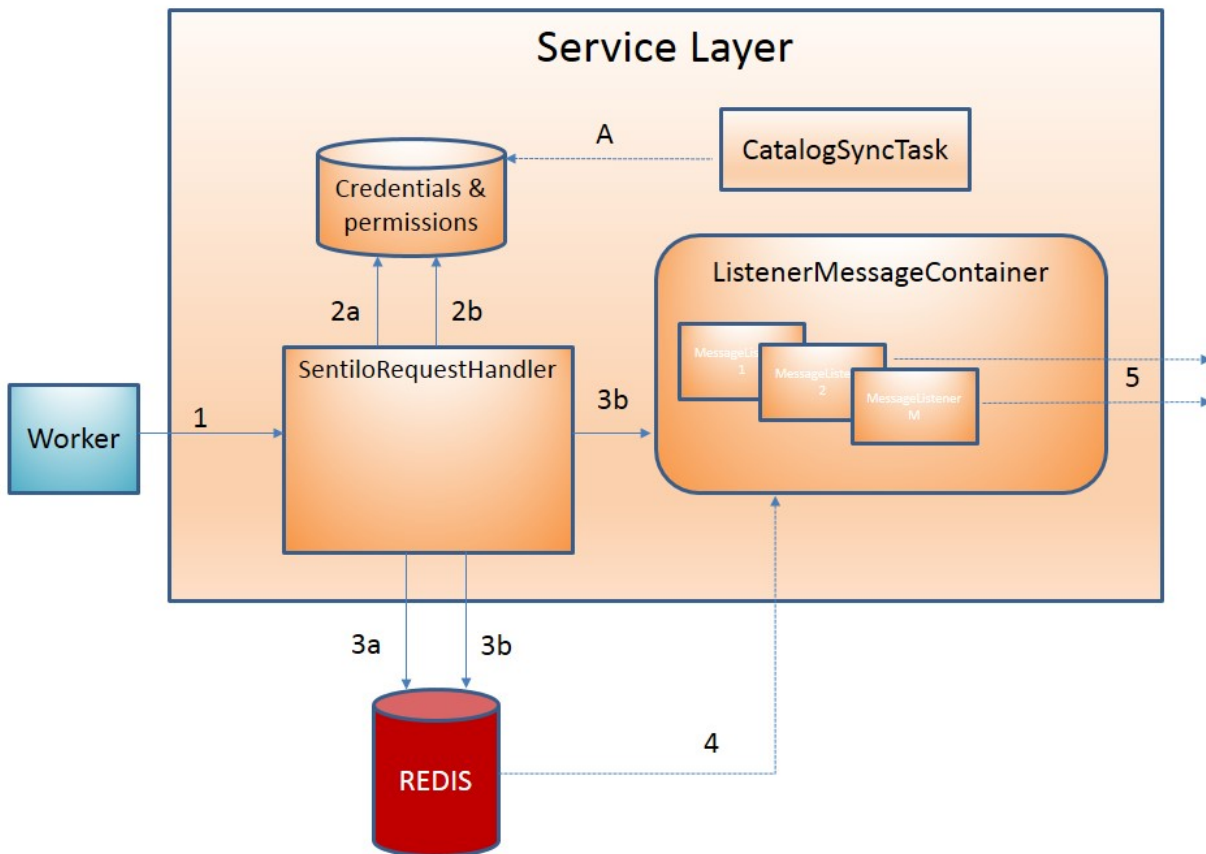
```
<bean id="ThreadPool" class="org.sentilo.platform.server.pool.ThreadPool"
  p:initialCapacity="${thread.pool.capacity.initial}"
  p:maxCapacity="${thread.pool.capacity.max}"
  p:shutdownSecondsTimeout="${thread.pool.shutdown.timeout.seconds}"
  p:QUEUE_SIZE="${thread.pool.queue.size}"
  p:groupId="${thread.pool.group.id}"
  p:groupName="${thread.pool.group.name}" />
```

```
thread.pool.queue.size=100
thread.pool.capacity.initial=4
thread.pool.capacity.max=10
```

Service Layer

The design of this layer has the main premise of minimizing the request processing time, so all the main job is held in memory(Redis). Redis stores data in a memory database but also has the possibility of disk storage to ensure the durability of the data.

The following diagram shows the main flow for a request within this layer:



NOTE: (*) Executed asynchronously to the main process.

- The Worker delegates the request to the associated handler depending on the type of request (data, order, alarm, ...)
- The following validations are performed on each request:
 - **(2a)** Integrity of credential: checks the received token sent in the header using the internal database in memory containing all active credentials in the system.
 - **(2b)** Authorization to carry out the request: validate that the requested action can be done according to the permission database.

- the validity of the request parameters: mainly, structure and typology.
- After that:
 - stores the data in Redis (in memory)
 - and depending on the type of data
 - * **(3a)** publish the data through publish mechanism
 - * **(3b)** or register of the subscription in the ListenerMessageContainer
- Redis is responsible for sending the published information to ListenerMessageContainer event, who is responsible for managing the subscription in Redis as a client for any type of event. ****(*)****
- The container notifies the event to each subscription associated with it sending a request, via HttpCallback ****(*)****

The platform registers a task that runs periodically who is responsible for credentials & permissions synchronization, stored in memory in server (A). These data is retrieved from the catalog application. This will maintain anytime an exact copy of these values in memory and allows to check credentials and permissions instantly.

Finally, access to Redis is done through a connection pool fully configurable through properties file, which allows you to adjust to the specifics of each environment.

```
<bean id="jedisPoolConfig" class="redis.clients.jedis.JedisPoolConfig"
    p:maxActive="${jedis.config.maxactive}"
    p:maxIdle="${jedis.config.maxidle}"
    p:maxWait="${jedis.config.maxwait}"
    p:testOnBorrow="true"
    p:whenExhaustedAction="1"/>
```

```
jedis.config.maxactive=50
jedis.config.maxidle=50
jedis.config.maxwait=50
```

Comments

- This design allows system scalability both vertically and horizontally:
 - vertically: increasing the boundaries of work queue & workers.
 - horizontally: distributing the load across multiple instances or server nodes.
- It also reduce response time because the process is carried out in memory.

Catalog application

The catalog application platform is a web application built with Spring on the server side (Spring MVC, Spring Security, ..) using jQuery and bootstrap as presentation layer and MongoDB as data storage database.

This webapp consists of:

- a public console for displaying public data of components and sensors and their data
- a secured part for resources management: providers, client apps, sensors, components, alerts, permissions, ...

It is fully integrated with the Publish/Subscribe platform for data synchronization:

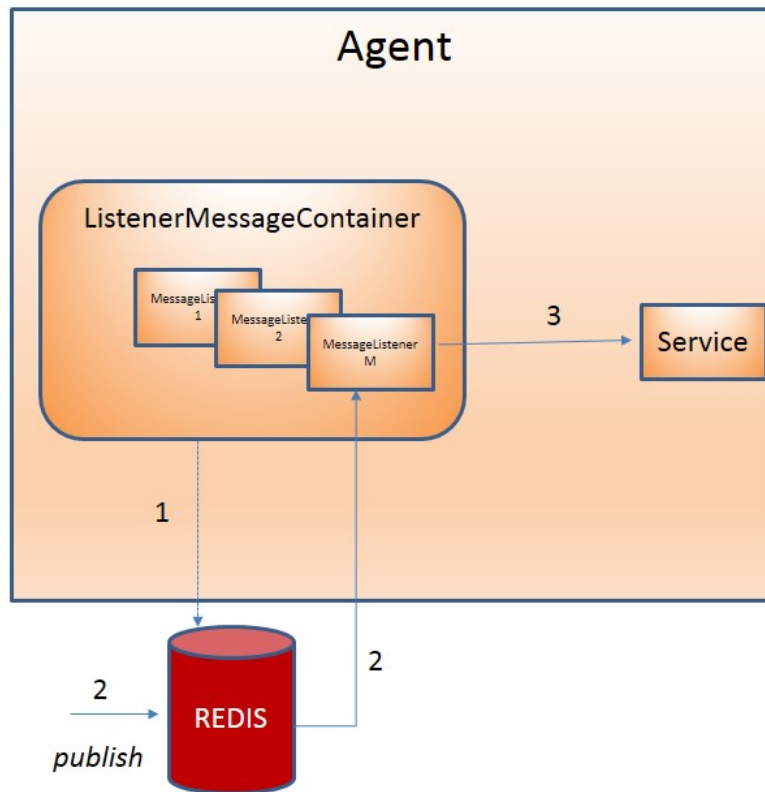
- permission and authentication data
- register statistical data and the latest data received for showing it in different graphs of the Web application.

5.1 Agents

Agents are internal modules oriented to expand its functionality without having to alter its core. The installation is based on the principle of Plug & Play: they are recognized by the system and started automatically to be up and running.

Every agent is a process that acts as a subscriber for the publish/subscribe platform. These processes will subscribe directly to Redis as a independent clients. This subscription will provide the input data to do the underlying business logic (store in a relational database, process alarms, generate statistics, ...)

The following diagram shows the design that every agent should follow:



1. When agent is started, it subscribes as client to Redis for the event that wants to receive notifications.
2. When Redis receives a publication of any of these data, the agent is automatically notified with a new message.
3. The message is processed and transferred to the corresponding agent's service responsible to carry out the underlying business logic.

As mentioned above, Sentilo currently provides two agents:

5.1.1 Relational database agent

This agent stores all information received from PubSub platform into a set of relational databases (the number of relational databases is fully configurable). It could be configured to filter the data to store according to a business rules through a configuration file.

To do this, when the agent is started it makes a subscription to the desired information in Redis (observations, orders and/or alarms), that has previously been defined in a properties file:

```
//In this example we indicate to persist any data using a DataSource with srDs_
↪identifier,
//and also to store any data from provider with PARKING identifier,
//on a different DataSource whose identifier is parkingDs.

//Finally, we can indicate more than one DataSource destination to persist the same_
↪data.
```

(continues on next page)

(continued from previous page)

```
data\:PARKING*=parkingDs
data\:*=srDs
order\:*=srDs
alarm\:*=srDs,parkingDs
```

It is imperative that the DataSources are defined in the context of the agent with the same identifier:

```
<bean id="srDs" class="org.apache.commons.dbcp.BasicDataSource"
    destroy-method="close">
    ...
</bean>

<bean id="parkingDs" class="org.apache.commons.dbcp.BasicDataSource"
    destroy-method="close">
    ...
</bean>
```

This context is defined in the file:

```
sentilo-agent-relational/src/main/resources/spring/relational-persistence-context.xml
```

5.1.2 Alarm agent

This agent processes each internal alert defined in the catalog and publish a notification (a.k.a. *alarm*) when any of the configured integrity rules are not met.

Due to the type of available rules, this validation process integrity is divided into two threads:

- An internal process that runs every minute, evaluates the status of each sensor that have associated (*frozen* type) alerts.
- Additionally, each time a Redis notification is received, alerts associated with the data received are evaluated.

Finally, an internal process regularly synchronize the alert list, to synchronize the information stored in memory with the catalog repository.

5.1.3 Activity Monitor agent

Background on Activity Monitor Agent

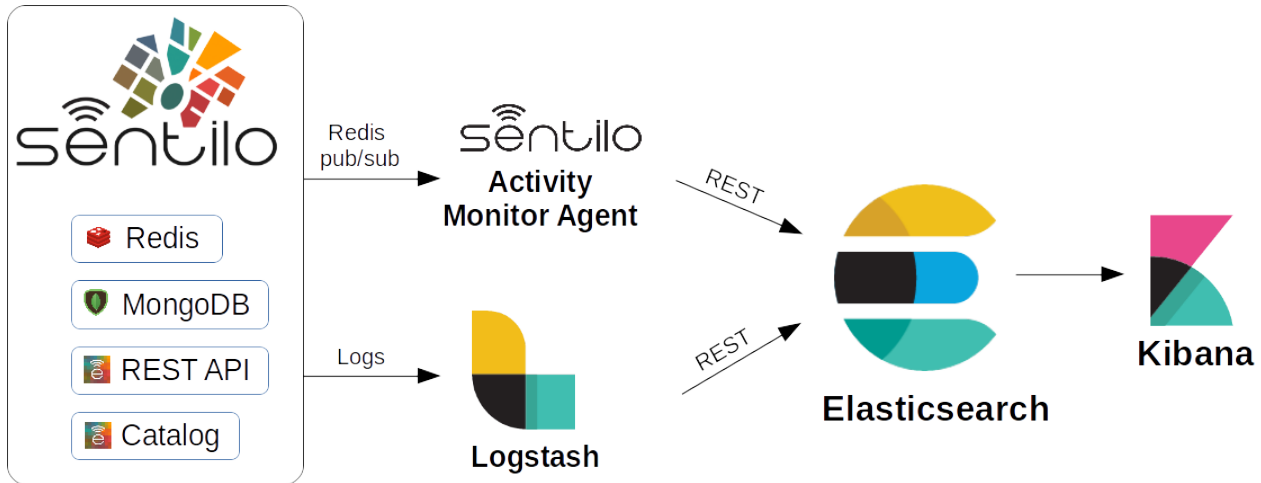
Sentilo is a publication-subscription platform. The amount of data held in the system is proportional to Redis deployment and directly depends on the amount of physical memory available for the Redis server. In another words, the data has to be probably deleted after a certain amount of time to free the Redis memory. For example, in the Barcelona deployment, the data is deleted after approximately one week.

Additionally to data expiration, Sentilo does not provide many dashboards and those dashboards are not customizable.

In order to fill the gap of historization and dashboards, we use [Elasticsearch](#) and [Kibana](#). Elasticsearch is a powerful Java-based fulltext search database with REST API. It is frequently used together with it's modules, Kibana for dashboards and Logstash for collecting of logs. The combination of Elasticsearch, Logstash and Kibana is often called the ELK stack. ELK provides a comfortable way to store and exploit historical information, and also a near-realtime monitoring of the platform. Note that Elasticsearch behaves excellently in cluster mode.

Sentilo events are uploaded to Elasticsearch through a Sentilo agent called Activity Monitor Agent. The configuration of this agent is described further in this chapter.

The following image illustrates a possible setup of Sentilo with ELK stack. Logstash is optional and can be used e.g. for monitoring of Sentilo logs (like login errors, invalid messages etc.), as well as monitoring of system resources.



The setup of the ELK stack is well documented and beyond the scope of this page.

Configuration

Activity Monitor Agent is configured with a set of .properties files in *sentilo/sentilo-agent-activity-monitor/src/main/resources/properties*.

subscription.properties

Property	Description	Comments
topics-to-index	Regex pattern on event name that enables including/excluding events	Examples of configuration: <code>/alarm/*,/data/*,/order/*</code> Subscribes to all events <code>/data/PROVIDER1/*,/data/ ↪PROVIDER2/*</code> Subscribe only to data of 2 providers

monitor-config.properties

Property	Description	Comments
elastic-search.url	URL of the ES instance	
batch.size	How many events are sent to ES at once.	Every HTTP request consumes certain amount of resources, thus is convenient to use a ES bulk API. The agent won't send events to ES until batch.size events occurred.
batch.worker.number	Number of threads the agent uses	Determines how many parallel threads communicate with ES.
batch.max.retries	Number of retries when ES is unavailable	Number of intents for upload to ES instance.

The agent will create index(es) called *sentilo-YYYY-MM*.

Configuration of Elasticsearch, Logstash and Kibana is beyond the scope of this document and can be easily followed on their respective web pages.

Compatible versions

Sentilo has been successfully used in with these versions of ELK (which does not mean other versions shouldn't work as well):

- ELK 5+

5.1.4 Historian agent

Background on Historian Agent

As you already might have learned, Sentilo does not persist data forever because of limited system resources.

Commonly used setup of a Sentilo instance is to employ one of the agents to copy the data into some external database or storage.

Since the data volumes can be fairly big and the data are mostly structured (except when the observations are text), it is convenient to use a scalable solution for time series such as [OpenTSDB](#).

OpenTSDB installs of top of HBase and HDFS. Exposes a HTTP REST API and can be used from [Grafana](#) as one of it's datasources.

Configuration

Historian Agent is configured with a set of .properties files in `sentilo/sentilo-agent-historian/src/main/resources/properties`.

subscription.properties

Property	Description	Comments
topics-to-index	Regexp pattern on event name that enables including/excluding events	<p>Examples of configuration</p> <pre>/alarm/*,/data/*,/order/*</pre> <p>Subscribes to all events</p> <pre>/data/PROVIDER1/*,/data/PROVIDER2/*</pre> <p>Subscribes only to data of 2 providers</p>

monitor-config.properties

Property	Description	Comments
opentsdb.url	URL of the OpenTSDB instance	
batch.size	How many events are sent to OpenTSDB at once.	Every HTTP request consumes certain amount of resources, thus is convenient to use a OpenTSDB bulk API. The agent won't send events to OpenTSDB until batch.size events occurred.
batch.workers	Number of threads the agent	Determines how many parallel threads communicate with OpenTSDB.
batch.max.retries	Number of retries when OpenTSDB is unavailable	Number of intents for upload to OpenTSDB instance.

Configuration of HDFS, HBase, OpenTSDB and is beyond the scope of this document and can be easily followed on their respective web pages.

Compatible versions

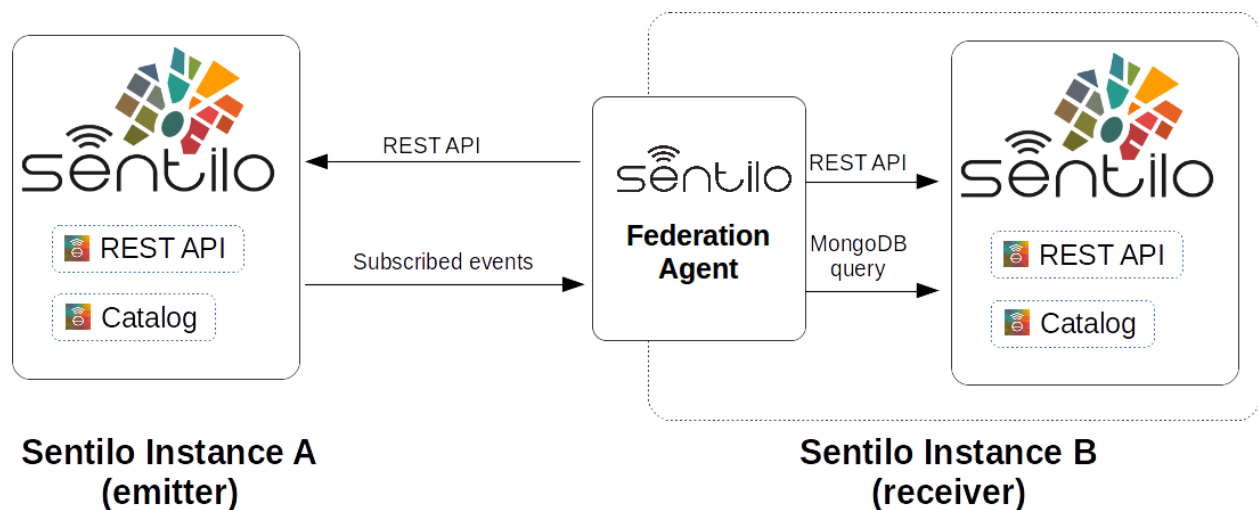
Sentilo has been successfully used in with these versions:

- Hadoop 2.7.2
- HBase 1.2.1
- Opentsdb 2.2.0, 2.3.0
- Grafana 3 +

5.1.5 Federation agent

Description

The federation agent is a module that permits to share events between two independent instances of Sentilo. The sharing is unilateral - one Sentilo instance is emitting events and the other is receiving. The agent is installed at the side of the receiving instance:



The administrator of the emitting Sentilo instance only needs to create a new application and provide the token the administrator of the receiving instance. As with any Sentilo application, the administrator is in control of which provider's data are readable by the remote federation agent.

Providers, components and sensors are created automatically in the catalog of the receiving instance by the federation agent. The agent uses its application token to query the emitting catalog API to obtain remote objects, and uses the local catalog application id to replicate the locally.

The federation agent creates subscriptions on data it has permission. It creates a HTTP endpoint and tells the emitting instance to forward the events to this endpoint URL.

Configuration

Federation Agent's configuration is in file `sentilo/sentilo-agent-federation/src/main/resources/properties/application.properties`.

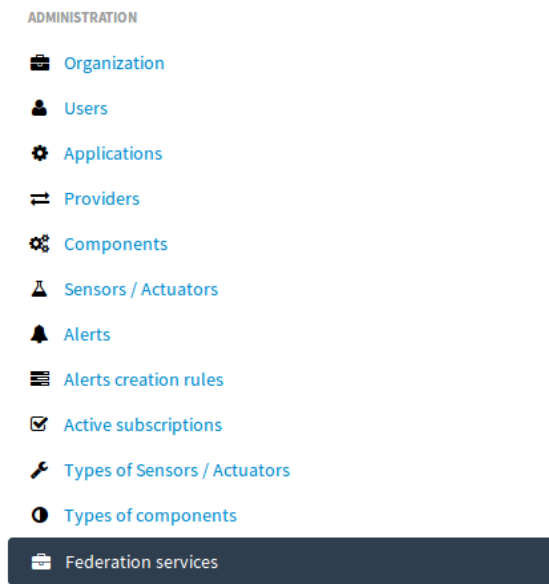
Property	Default Value	Description
server.port	8082	Agent's HTTP port
rest.client.local.host	http://127.0.0.1:8081	Local Sentilo API endpoint
sentilo.master.application.id	sentilo-catalog	Local Sentilo application Id. The agent will use the token of the application to make changes in catalog
catalog.mongodb.host	127.0.0.1	Local MongoDB host
catalog.mongodb.port	27017	Local MongoDB port
catalog.mongodb.database	sentilo	Local MongoDB database name
catalog.mongodb.user	sentilo	Local MongoDB user
catalog.mongodb.password	sentilo	Local MongoDB password
federation.subscription.endpoint	http://localhost:8082/data/federated/	Agent URL that will be used in subscriptions in the remote Sentilo instance.
federation.subscription.secret.key	secret-callback-key-change-it	HMAC secret used for incoming subscription.
federation.subscription.max.retries	3	Number of retries used for subscription
federation.subscription.max.delay	5	Delay used for subscription

Further configuration of the agent is available in the “Federation services” menu.

The menu is available when running Tomcat with the option:

```
-Dsentilo.federation.enabled=true
```

The “Client application token” input is the token created in the emitting Sentilo instance:



New federation

Identifier	<input type="text" value="FED-SENITO-A-SENITO-B"/>
Name	<input type="text" value="FED-SENITO-A-SENITO-B"/>
Description	<input type="text" value="Federation with Sentilo A"/>

Sentilo federate configuration

Client application name	<input type="text" value="senatilo-a@FFED-SENITOA-SE"/>
Client application token	<input type="text" value="123qwe456rty789uio123qwe456"/>
Endpoint	<input type="text" value="http://sentilo-a.com:8081"/>

Contact info

Name	<input type="text" value="Admin Sentilo A"/>
e-mail	<input type="text" value="admin@sentilo-a.com"/>
<input type="button" value="Back"/> <input type="button" value="Save"/>	

5.1.6 Kafka agent

Description

The Kafka agent publishes Sentilo events to Kafka.

Configuration

Property	Default Value	Description
kafka.bootstrap.servers	localhost:9092	Comma-separated list of Kafka brokers
zookeeper.connect	localhost:2181	Comma-separated list of Zookeeper nodes
batch.worker.size	10	Number of worker threads
batch.max.retries	4	How many times will the agent try to resend the message to Kafka until it gives up
kafka.request.timeout.ms	30000	
kafka.linger.ms	100	Milliseconds before the contents of buffer are sent or until batch fills up, whichever comes first.
kafka.batch.size	2000	Number of bytes of internal buffer. If the size fills up before , contents are sent to Kafka, . Otherwise contents are sent once kafka.linger.ms passed.
kafka.topic.prefix	sentilo	Topics in Kafka will start with following prefix. May be left blank
kafka.topic.separator	.	The compound name of topic in Kafka will be separated with this string.
kafka.topicNameMode	topicPerSensor	Possible values of topicNameMode for the “data” event type: * topicPerSensor: sentilo.data.providerName.sensorName * topicPerProvider: sentilo.data.providerName * topicPerSensorType: sentilo.data.temperature * topicPerMessageType: sentilo.data * singleTopic: sentilo

Compatible versions

Sentilo has been successfully used in with these versions:

- Kafka 0.11.0

5.2 Node-red

Node-RED offers a fast integration and prototyping ecosystem for Sentilo. There’s a Sentilo ad-hoc node in /sentilo-node-red. In order to activate it to your local Node-RED installation procede with two simple steps:

1. From the directory /sentilo-node-red, type:

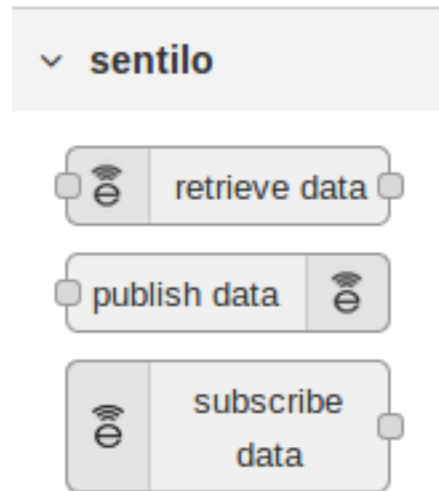
```
npm link
```

This command registers the package node-red-contrib-sentilo to the NodeJS node_modules.

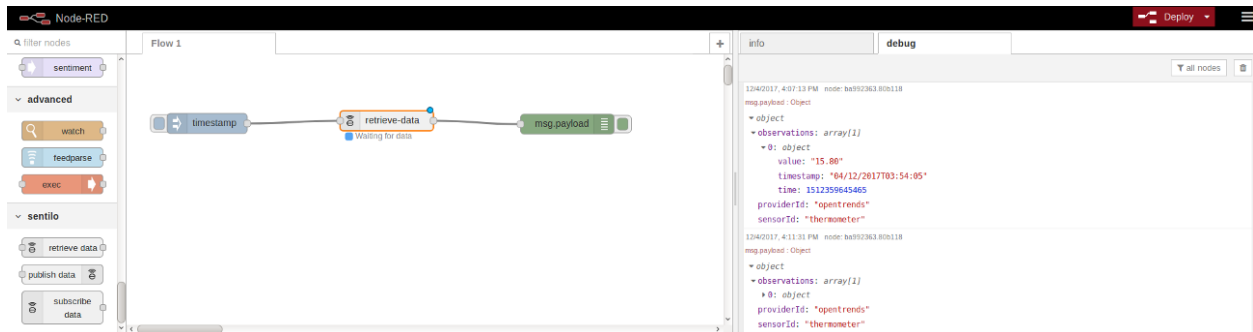
2. In order to add the module to your NOdeRED installation, type:

```
cd ~/.node-red
npm link node-red-contrib-sentilo
```

Then, following nodes should appear in the nodes palette:



Now, you should be able to use Sentilo from Node-RED:



6.1 Introduction

The Catalog is a web application that enables you to administer, rule and monitor the Sentilo platform resources and activity. On this page, you will learn how to manage and administer the Sentilo resources and how to monitor its activity.

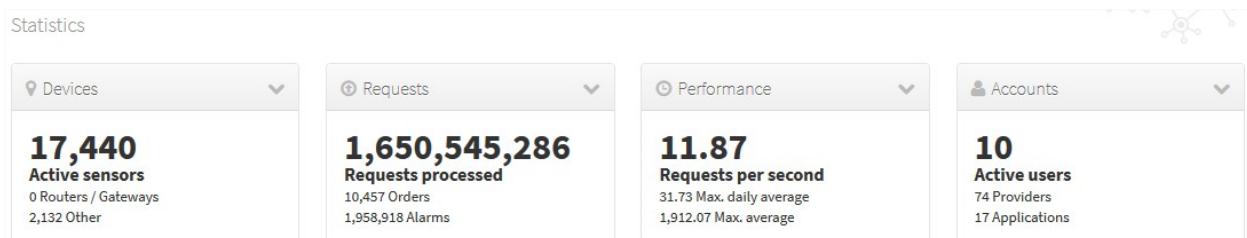
We will begin with the monitoring and then follow with the administration section.

6.2 Sentilo monitoring

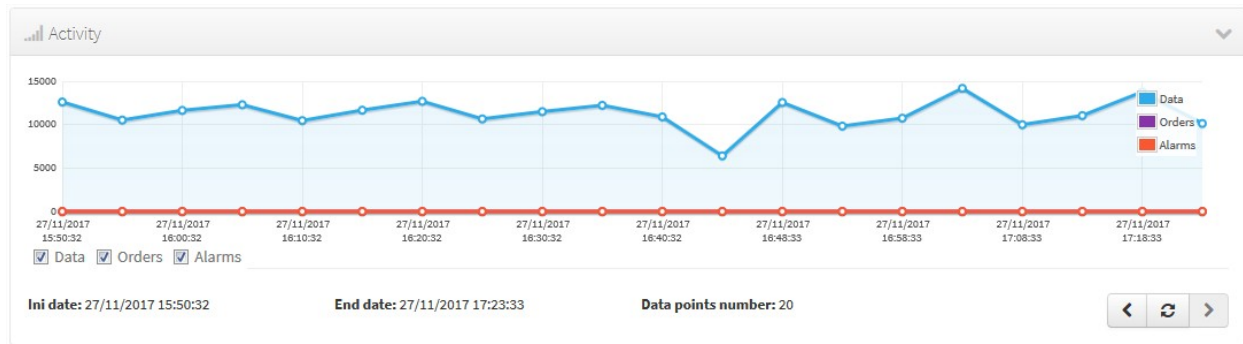
The Catalog allows us to display some Sentilo statistics through a set of features/pages which allow us to inspect the current platform activity and to display the components/sensors over a map.

6.2.1 Statistics

The statistic dashboard, which is accessible from the top menu bar, displays some basic use indicators, like total requests processed, number of sensors registered, current requests per second, max daily average and max average requests per second, These values are automatically updated every 30 seconds.



It also shows a time-series graph which displays the platform activity (such as observations, orders and alarms) for the last 100 minutes. This graph is automatically updated every 5 minutes.



Navigate the last data chart

You can navigate along the dates of the graph by using the buttons located in the lower right corner of it:



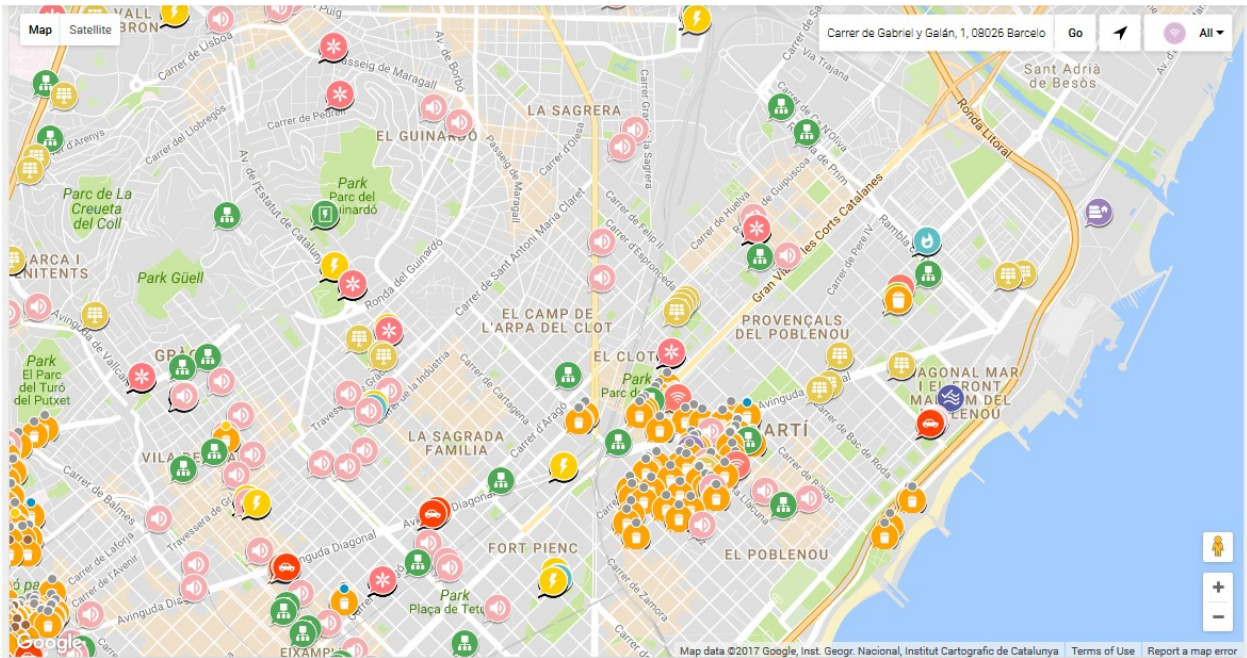
- **left arrow:** navigate to the past (only if there are older data)
- **reload data (center button):** reload last data / reset chart data
- **right arrow:** navigate to the future (only if you have navigated or gone into the past before)

You could also monitor the current activity of each sensor from the different viewers available which are accessible from the *Explore* item at the top menu bar (*Universal viewer* and *Route viewer*).

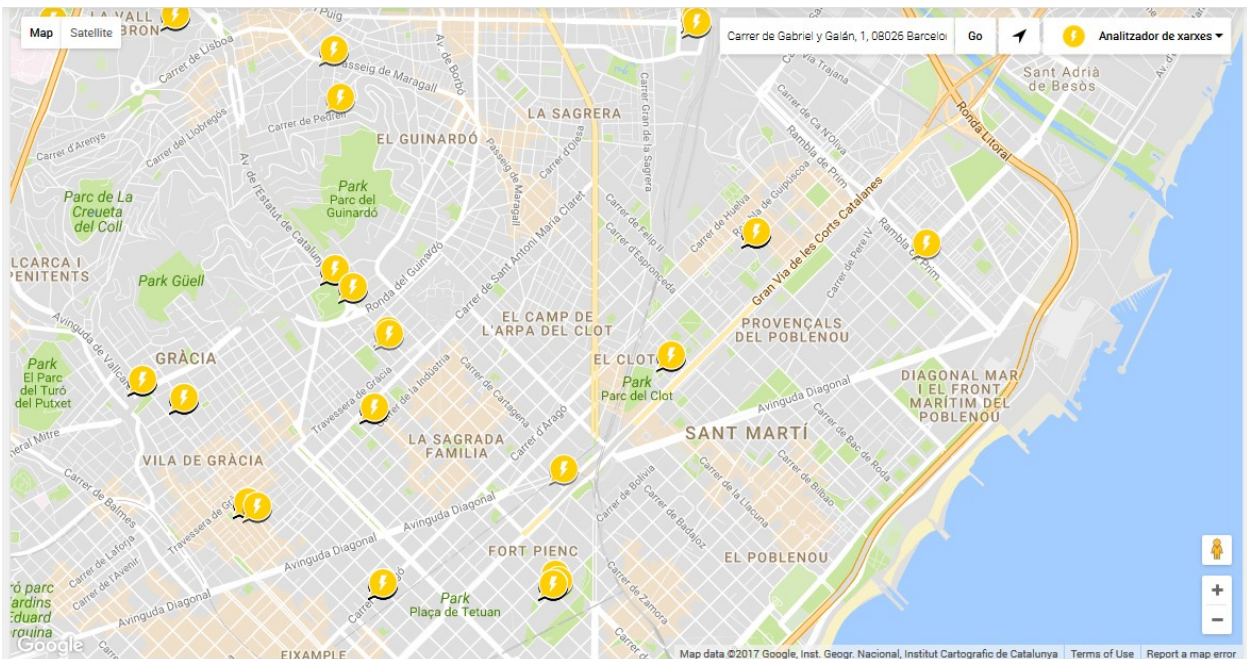
6.2.2 Universal viewer

Components map

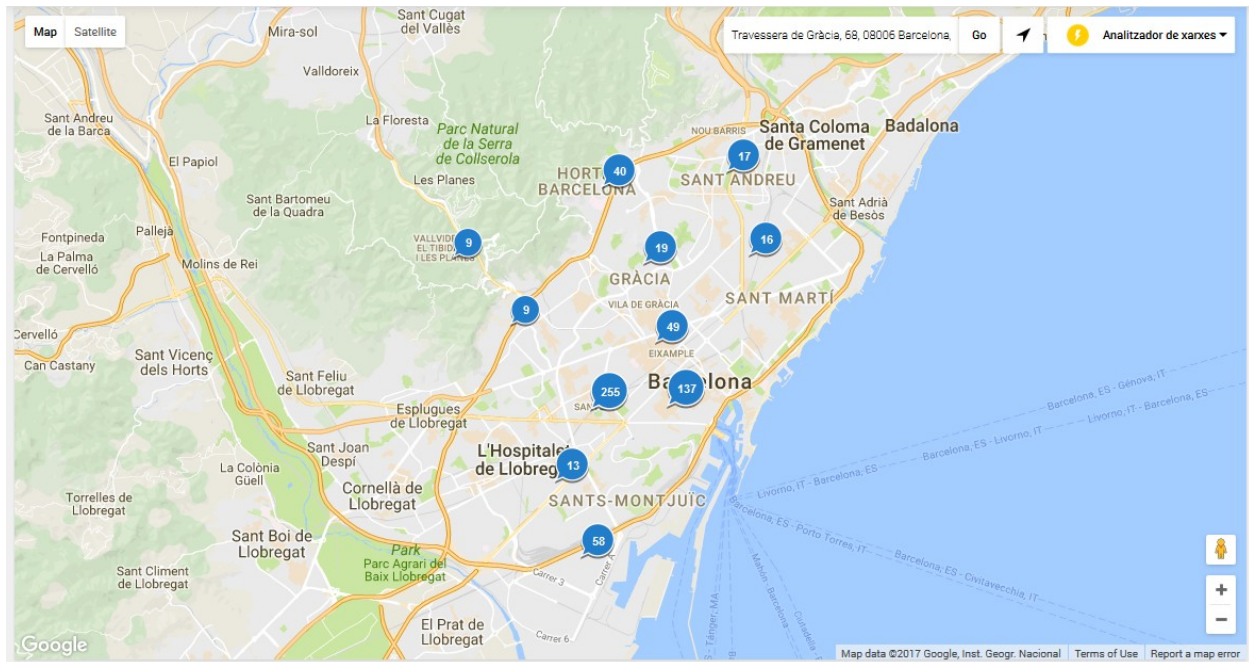
The catalog provides a default map, based on Google Maps, which shows all the public components registered at the platform. If the user is logged as administrator, all the private components will be displayed as well.



On this page, you can filter the components to show by selecting a *component type* from the top left select.



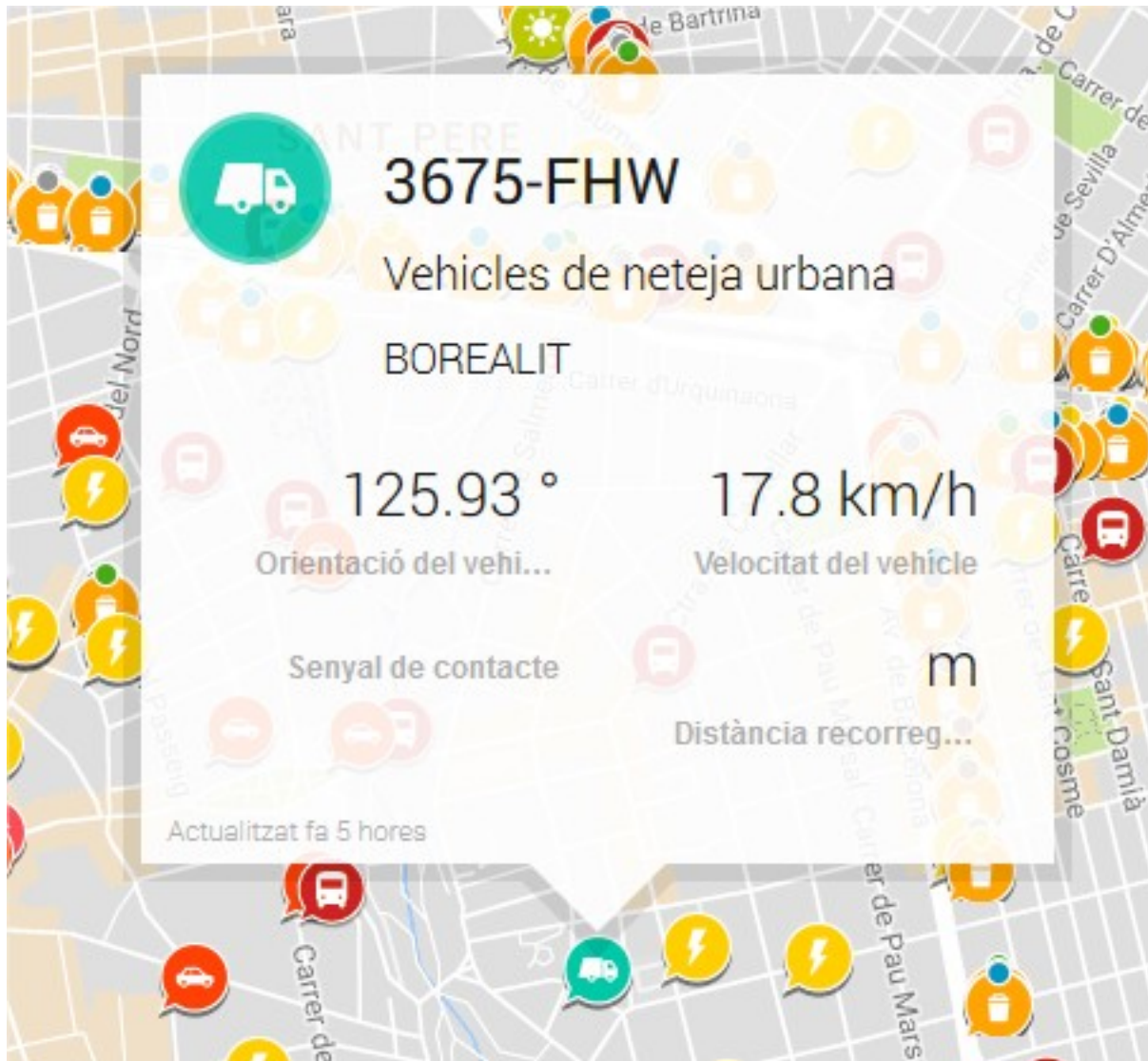
Depending on the zoom level, the map will display the elements as individuals POIs or grouped in clusters, showing the number of components in each group.



Component details

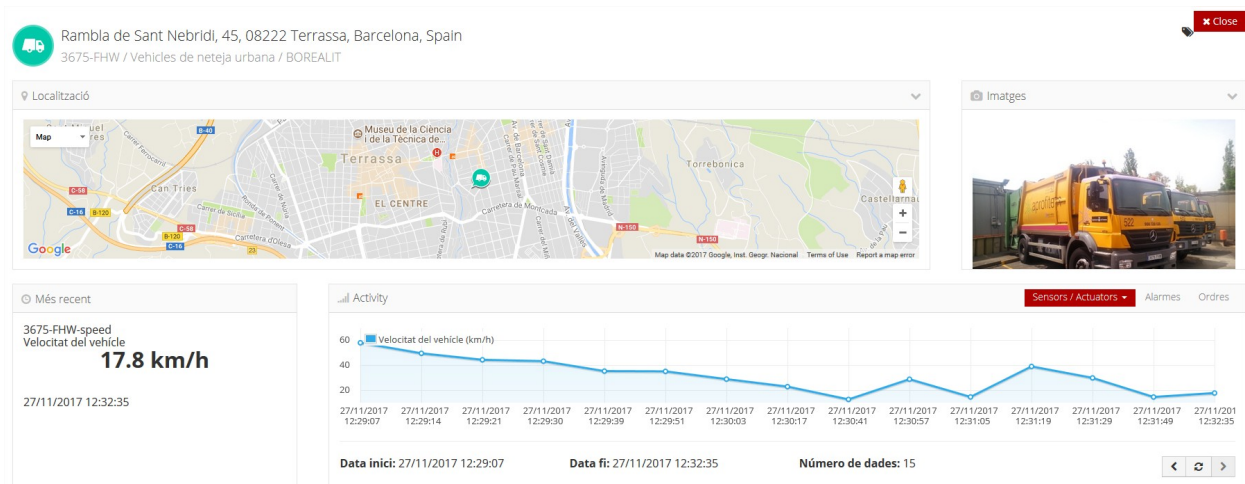
Sensors list

When you select a component, a popup window is opened above the map and displays the list of sensors related to it with the last activity for each one of them (as noted above, the private sensors will be displayed only for logged users):



Sensors last activity view

If you click into the content area of the popup window, a new page is open displaying some basic details about the component, and a time-series graph with the last activity of each of its sensors:



Navigate the last data chart

You can navigate along the dates of the graph by using the buttons located in the lower right corner of it:



- **left arrow**: navigate to the past (only if there are older data)
- **reload data (center button)**: reload last data / reset chart data
- **right arrow**: navigate to the future (only if you have navigated or gone into the past before)

Default configuration

The initial map settings are configured in the file:

```
sentilo-catalog-web/src/main/webapp/WEB-INF/jsp/common/include_script_maps.jsp
```

and includes settings such as the default map center, the default zoom,

```
// the initial map center (Barcelona city)
var defaultMapCenter [ 41.4001221, 2.172839 ];
var defaultZoomLevel 14;
var defaultInputLabelZoomLevel 17;
// the maximum zoom level beyond which pois are not grouped into clusters
var defaultMaxZoomCluster 13;
// flag for control if routes must be displayed or no on the current map
var showRoutes false;
// the minimum zoom level beyond which routes are displayed
var minRouteZoomLevel 15;
// flag for control if only components that fit into bounds's map must be searched on
// the server
var filterByBounds true;
...
```

You could change these configuration parameters or customize the look and feel to meet your requirements, but instead of overwrite the existing values you could add the new values to the file:

```
sentilo-catalog-web/src/main/webapp/WEB-INF/jsp/common/include_script_maps_config.jsp
```

For example, add the following line to the `include_script_maps_config` file if you would change the initial map center to London:

```
var defaultMapCenter [ 51.4991257, -0.11325074];
```

When Catalog starts, properties in this file overwrites existing ones having the same name in the file `include_script_maps.jsp`

Displaying complex data

In some cases, you may want to inform **complex data** as an observation on Sentilo, such like a large json object. For these cases, Sentilo will detect that the text is a json object and then it will be shown to you as a prettyfied json value:

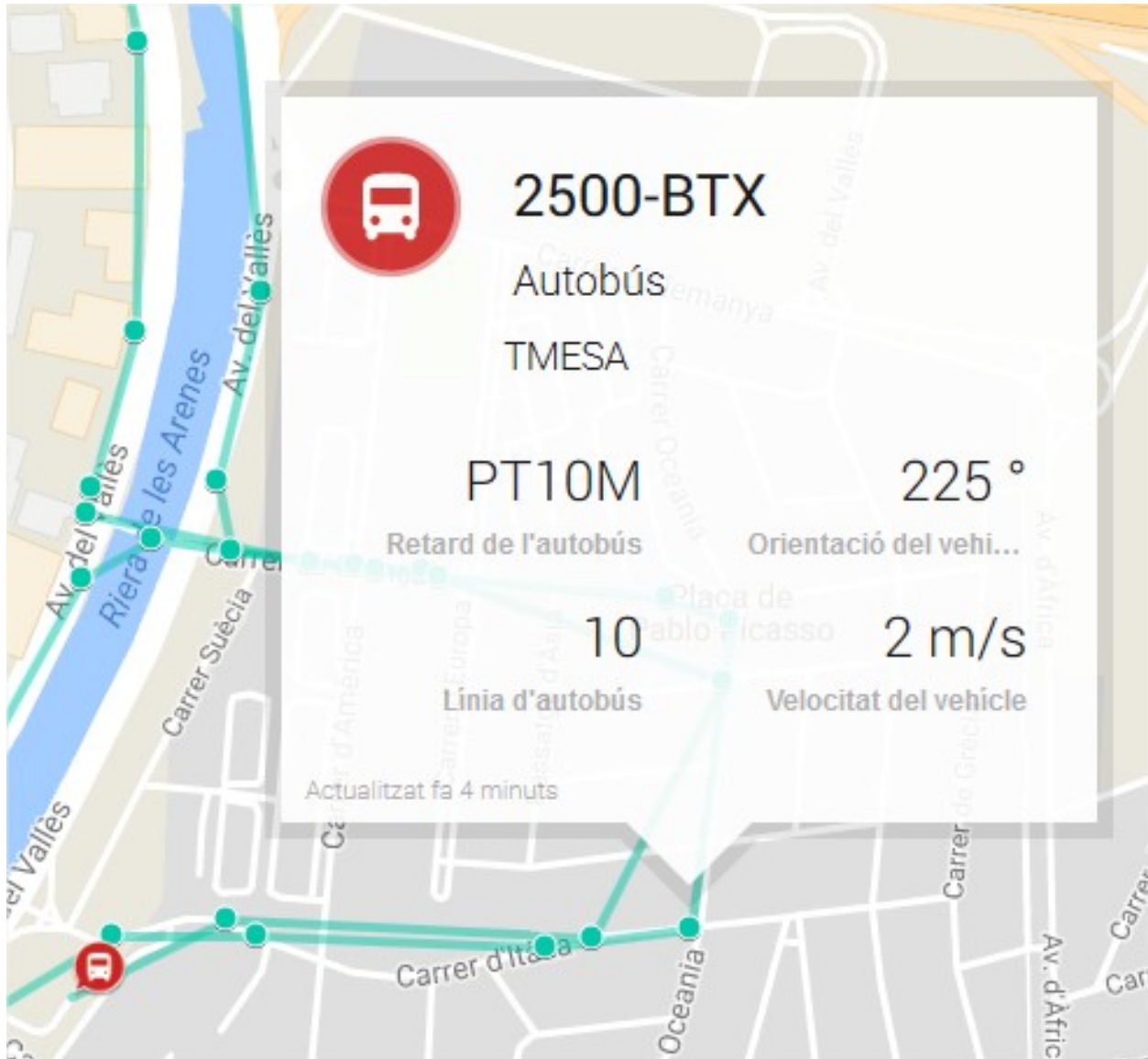
The screenshot shows the Sentilo web interface. At the top, there's a header with the Sentilo logo, navigation links (Statistics, Explore), and a user profile (admin). Below the header, the main content area is divided into several sections:

- Location:** A map of Barcelona, Spain, centered on Carrer de Josep Pla, 93. The map shows various landmarks like EL CAMP DE L'ARPA DEL CLOT, PROVENÇALS DEL POBLENOU, and Parc del Fòrum.
- Images:** A section labeled "No photo yet" with a camera icon.
- Most recent:** A section showing a complex JSON data object for a sensor status. The JSON is prettyfied and includes fields like id, date, and sensors.
- Activity:** A section showing a list of sensor status updates. Each entry includes a timestamp, a status object, and a list of sensors. The sensors include wind speed, direction, and humidity.
- Summary:** A section at the bottom showing the initial date, end date, and data points number for the selected activity.

You can expand or compress the prettified json with the bottom buttons under the status field,

6.2.3 Route viewer

As the name suggest, the route viewer is a specific map that shows the routes followed by the mobile components (keep in mind that only the last 20 points are displayed for each route):



6.3 Administration console

The administration console is composed of several CRUDs used to maintain all the entities of the Catalog such as providers, components, sensors, users, ... Only registered users can access it, so you must be logged before starting to manage the Catalog (the login access is located at the top right menu bar). Remember that, by default, the admin user has admin/1234 as access credentials.

All admin pages follow the same structure and layout for ease of use and to facilitate future maintenance. Therefore, below there is only a brief description of each admin page rather than to repeat the same things over and over in every section. In these sections will focus only on the particularities of each one.

When you select any option of the menu admin, the first page that you will see will be a list with the resources of this type already registered on the Catalog. These lists are very intuitive and extremely easy-to-use: you could filter, page and order it. You could delete an existing resource selecting the corresponding checkbox and clicking the *Delete selected* button; you could add new resources selecting the corresponding button and you could edit anyone clicking over the corresponding row.

Component's typology

10 items per page

Filter

Identifier	Name	Description	Creation date
electricity_meter	Electricity meter		08/11/2013 00:00
generic	Generic	Generic component type	08/11/2013 00:00
meteo	Meteo		08/11/2013 00:00
noise	Noise meter		08/11/2013 00:00

Showing 1 to 4, from 4 records

Export to Excel

Delete selected New typology

When you select to add a new resource, a traditional form page is displayed. Here, you must have filled in the mandatory fields before clicking the *Save* button. If some mandatory field is not filled in or it have a no valid value, the page shows you information about what is wrong:

New provider

Identifier Required
Only letters, numbers, hyphens and underscores are allowed

Name

Description

Contact info

Contact name Required

Contact email Required

Back Save

Otherwise, the resource will be registered into the Catalog and you will be redirect to the list page (at the top right corner you will see a confirmation message that the resource have been succesfully created):

Component's typology

10 items per page

Filter

Identifier	Name	Description	Creation date
air_quality	Air quality	Lorem ipsum dolor si amet	06/11/2015 09:09
electricity_meter	Electricity meter		08/11/2013 00:00
generic	Generic	Generic component type	08/11/2013 00:00
meteo	Meteo		08/11/2013 00:00
noise	Noise meter		08/11/2013 00:00

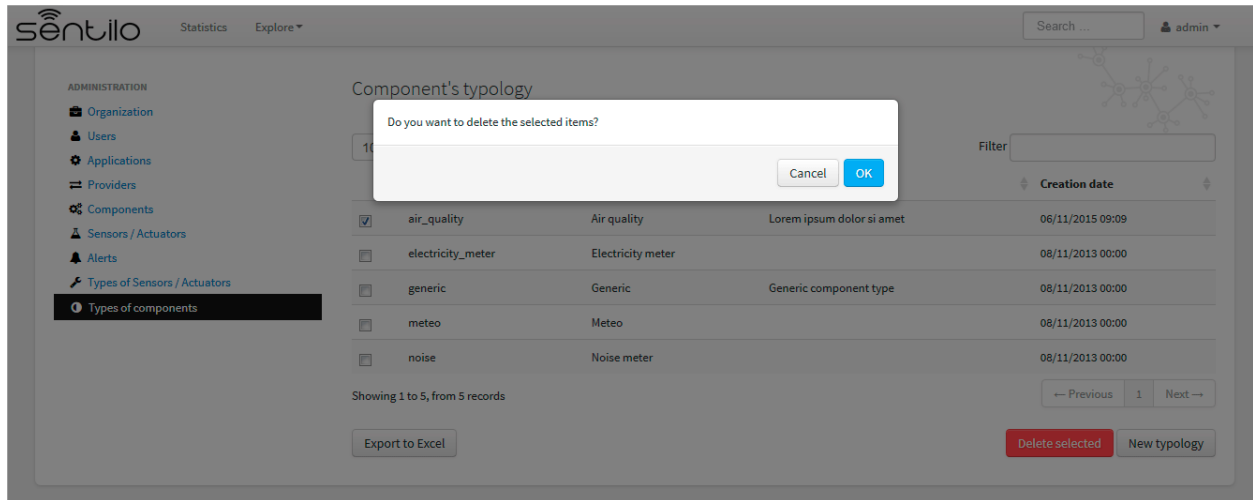
Showing 1 to 5, from 5 records

Export to Excel

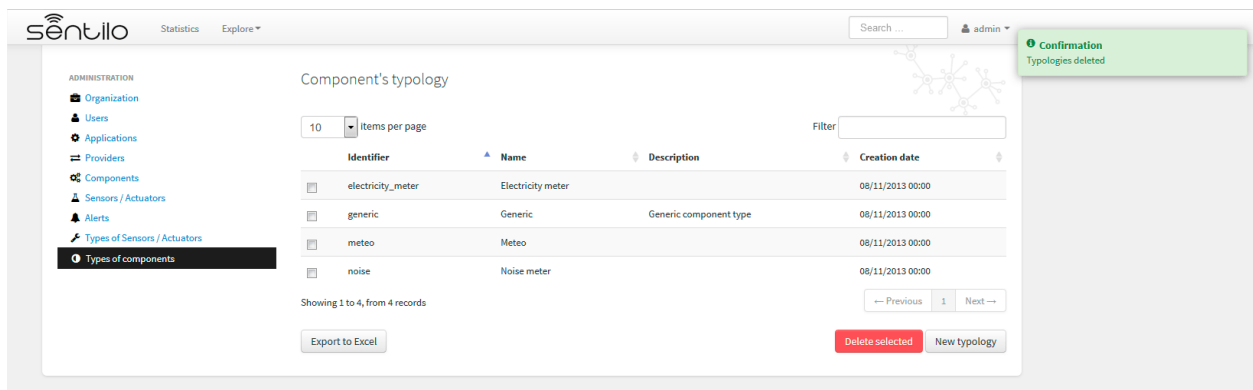
Delete selected New typology

Confirmation
Typology added

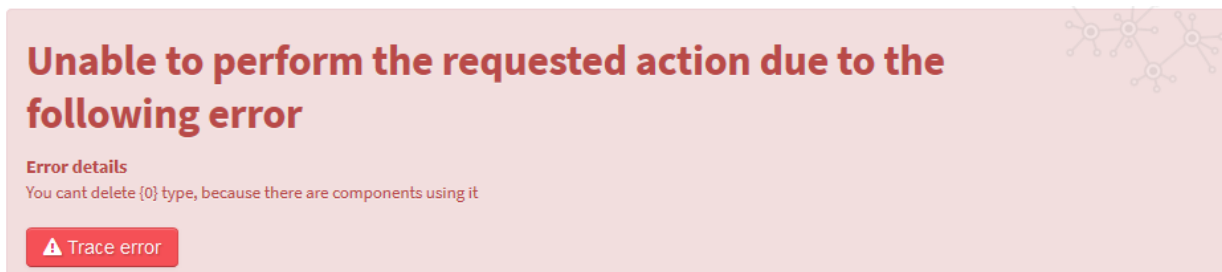
The same applies when you try to delete a resource, but with the peculiarity that the browser will always ask for your confirmation before deleting it:



If the resource has been successfully removed, the list is reloaded and a confirmation message is displayed at the top right corner:



Otherwise, you will see an error page with a description about what is wrong. For example, if you try to delete a component type that is associated with an existing component the response will be :

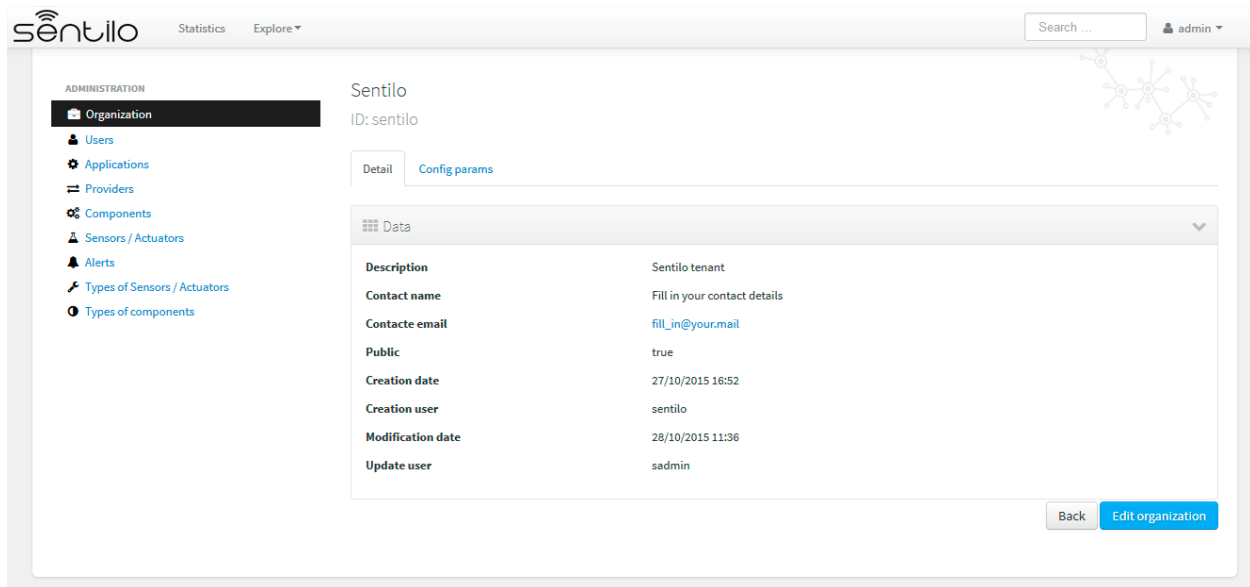


6.3.1 Organization

The organization is the entity that describes the Sentilo instance.

Detail

By default, this organization is created and its identifier is **sentilo**.



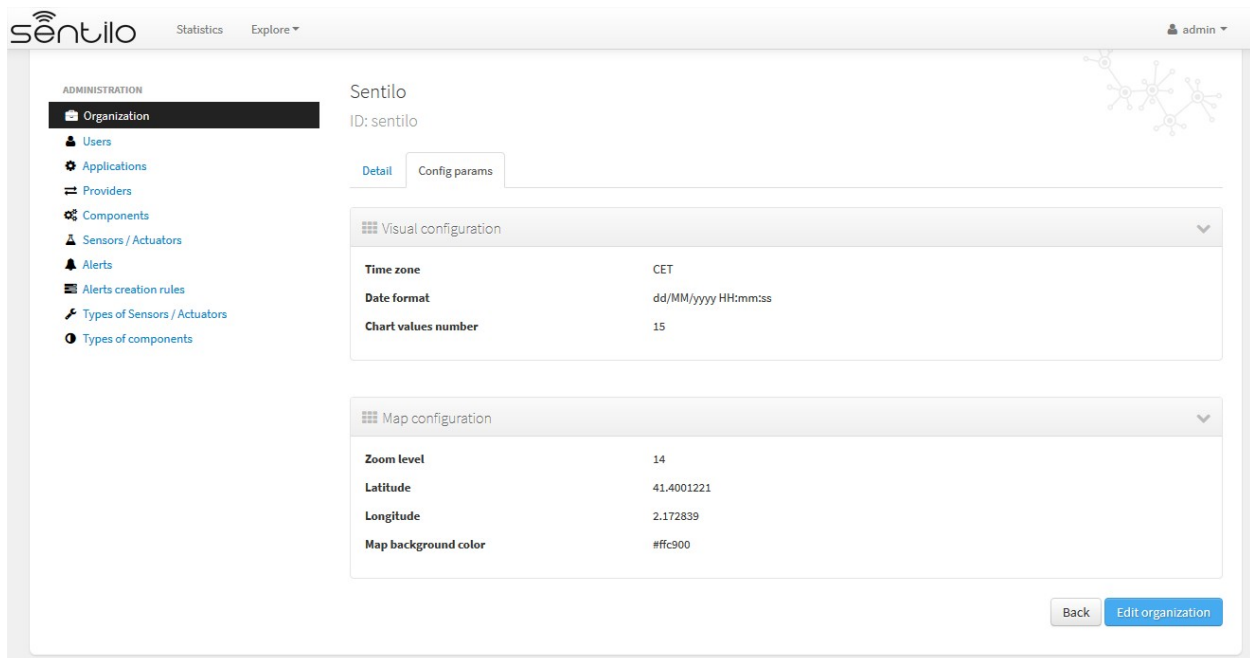
The screenshot shows the Sentilo web interface. The top navigation bar includes the Sentilo logo, 'Statistics', 'Explore', a search bar, and a user profile 'admin'. The left sidebar lists administrative options: Organization (selected), Users, Applications, Providers, Components, Sensors / Actuators, Alerts, Types of Sensors / Actuators, and Types of components. The main content area displays the 'Sentilo' organization details. The 'Detail' tab is active, showing a table of organization data. The 'Config params' tab is also visible. At the bottom right, there are 'Back' and 'Edit organization' buttons.

Data	
Description	Sentilo tenant
Contact name	Fill in your contact details
Contact email	fill_in@your.mail
Public	true
Creation date	27/10/2015 16:52
Creation user	sentilo
Modification date	28/10/2015 11:36
Update user	sadmin

We can also edit the organization's name other contact details, except the.

Config params

In addition, we can edit the visualization formats and public map settings, using the **Config params** tab:



The screenshot shows the Sentilo web interface with the 'Config params' tab selected. The main content area displays two configuration sections: 'Visual configuration' and 'Map configuration'. The 'Visual configuration' section includes settings for Time zone (CET), Date format (dd/MM/yyyy HH:mm:ss), and Chart values number (15). The 'Map configuration' section includes settings for Zoom level (14), Latitude (41.4001221), Longitude (2.172839), and Map background color (#ff9900). At the bottom right, there are 'Back' and 'Edit organization' buttons.

Visual configuration	
Time zone	CET
Date format	dd/MM/yyyy HH:mm:ss
Chart values number	15

Map configuration	
Zoom level	14
Latitude	41.4001221
Longitude	2.172839
Map background color	#ff9900

There we can configure the Visual configuration and the Map configuration.

Visual configuration

These params will apply to the entire catalog application visual customization, and how the user will see the data. Note that time zone & date format are directly relationated.


Property	Description	Comments
Time zone	Defines the time zone of the organization, and modifies the way to display data on screen, such as dates	You can define hourly difference or time zone abbreviations: CET, UTC, +001...
Date format	Defines the date format with which the data will be displayed in the application (lists, details...)	Example: dd/MM/yyyy HH:mm:ss = 30/11/2017 15:34:56 See all possible formats as Java Date Format, at: Java Date Format
Chart values number	Number of observations displayed on chart	It must be a positive integer number greater or equals to 10. If blank, it will be a default value of 10. This value will be overwritten by sensor's configuration one.

Map configuration

These params configure the universal map visualization.

Property	Description	Comments
Zoom level	Zoom level of the universal map	Default value is 14. And you can define a value between 1 and 20. See possible values in: https://developers.google.com/maps/documentation/static-maps/intro#Zoomlevels
Latitude / Longitude	Defines the map center in latitude & longitude values format	
Map background color	Define the background color of the map	Possible values applies with the colorpicker, or input a valid css / html color value

For example, set the map background color to #ffc900:


 Map configuration

Zoom level

Latitude

Longitude

Map background color



will result in:



6.3.2 Applications

Applications are the data clients of the Sentilo platform and, by default, if you have loaded the default data, you will see two applications registered into the Catalog:

- **sentilo-catalog**: it is a internal application, used by the catalog to make calls to the API REST and therefore **MUST NOT** be removed.
- **testApp**: as the name suggest, this application is used for testing the platform status.

List

Access the Application list. This is the main Application page. From here you'll can access to the desired application to show its details by click on it.

The screenshot shows the Sentilo Administration console. On the left is a sidebar menu with options: Organization, Users, Applications (selected), Providers, Components, Sensors / Actuators, Alerts, Alerts creation rules, Types of Sensors / Actuators, and Types of components. The main area is titled 'Applications' and contains a table of 13 applications. Above the table is a filter input and a '10 items per page' dropdown. Below the table is a pagination control showing 'Showing 1 to 10, from 13 records' and buttons for 'Export to Excel', 'Delete selected', and 'New application'.

Identifier	Name	Description	Creation date
appDemo	Demo de la plataforma	App demo de la plataforma	22/02/2013 11:20:00 CET
GestioMediAmbient	Gestio MediAmbient		21/03/2013 14:44:50 CET
GestioTransit	Gestio Transit		21/03/2013 11:20:59 CET
GestioRec	Gestió del Rec	Gestió del Rec	20/03/2013 17:48:45 CET
OT_STRESS_APP_1	OT_STRESS_APP_1	Aplicació a les proves de carrega	06/06/2016 16:55:28 CEST
OT_STRESS_APP_2	OT_STRESS_APP_1	Aplicació a les proves de carrega	06/06/2016 16:55:28 CEST
RegIntelligent	RegIntelligent	App de reg intelligent	19/06/2013 14:53:00 CEST
samples-application	Samples Application	Aplicació de prova per a fer servir els codi d'exemple	09/04/2015 13:01:33 CEST
connecta-catalog	connecta-catalog	Aplicació del catalog	28/02/2013 14:15:00 CET
nimbus-monitor	nimbus-monitor	No borrar. Corresponde a la aplicación monitor-web	18/07/2016 15:14:43 CEST

You'll be able to list, filter, show application details, create (*New application* button) and delete selected applications (select from left checkbox, and apply by *Delete selected* button).

Further, you'll be able to export the list to Excel, by clicking on *Export to Excel* button. The result file will contain the list columns and a number of extra ones from internal database use.

Use the button panel at the bottom right to navigate through the list (first page, previous page, page number, next page and last page, respectively).

Details tab

The detail page is structured into three tabs:

The screenshot shows the Sentilo web interface. On the left is a navigation menu under 'ADMINISTRATION' with links for Organization, Users, Applications (selected), Providers, Components, Sensors / Actuators, Alerts, Alerts creation rules, Types of Sensors / Actuators, and Types of components. The main content area is titled 'sentilo-catalog' with ID 'sentilo-catalog'. It has three tabs: 'Details' (active), 'Permissions', and 'Active subscriptions'. The 'Details' tab shows a 'Data' section with the following properties:

Authorization Token	c956c302086a042dd0426b4e62652273e05a6ce74d0b77f8b5602e0811025066
Description	Catalog application
HTTPS API REST	<input type="checkbox"/>
Creation date	08/11/2013 13:15:01
Updated date	
Contact email	sentilo@sentilo.org

At the bottom right of the details section are two buttons: 'Back' and 'Edit application'.

where:

- the *Details* tab contains the main properties of the application (described below).
- the *Permissions* tab allows to manage the permissions for other entities (applications or providers)
- the *Active subscriptions* tab displays a list with all the active subscriptions for the current application (from version 1.5).

The main properties of the *Details* tab are the following:

Property	Description	Comments
Id	Application Identifier	Mandatory. After its creation it can't be modified. It is the identifier used in the API calls.
Name	Display name	If not filled in by the user, its default value will be the <i>Id</i> .
Token	Access key	Automatically generated by the system when application is created. It is the <i>identity_key</i> value used in the API calls. <i>NOTE: only users with ADMIN role will show the entire token chain, other user roles only will see obfuscated text at this place (see below)</i>
Description	Description	Optional. The application description text.
HTTPS API REST	Application accepts data over HTTPS	The Sentilo Server itself does not support SSL at the moment, however you can put a reverse proxy such as Nginx in front of the Sentilo Server. If this option is checked, the Sentilo Server expects the standard header <code>X-Forwarded-Proto</code> Please note that when configuring Nginx, you should also use the parameter <code>underscores_in_headers on;</code> so Nginx would forward sentilo headers to the Sentilo Server.
Contact email	Email address of the person responsible for the application	Mandatory.

How users that has not ADMIN role see the detail section:

The screenshot shows the Sentilo Administration console interface. On the left is a sidebar with navigation links: Applications (selected), Providers, Components, Sensors / Actuators, Alerts, Alerts creation rules, Types of Sensors / Actuators, and Types of components. The main content area displays the details for the 'sentilo-catalog' application. At the top, it shows the application name and ID. Below this are tabs for 'Details' (selected), 'Permissions', and 'Active subscriptions'. A 'Data' section is visible, containing a table with the following information:

Authorization Token	*****
Description	Catalog application
HTTPS API REST	<input type="checkbox"/>
Creation date	08/11/2013 13:15:01
Updated date	
Contact email	sentilo@sentilo.org

A 'Back' button is located at the bottom right of the details section.

Permissions tab

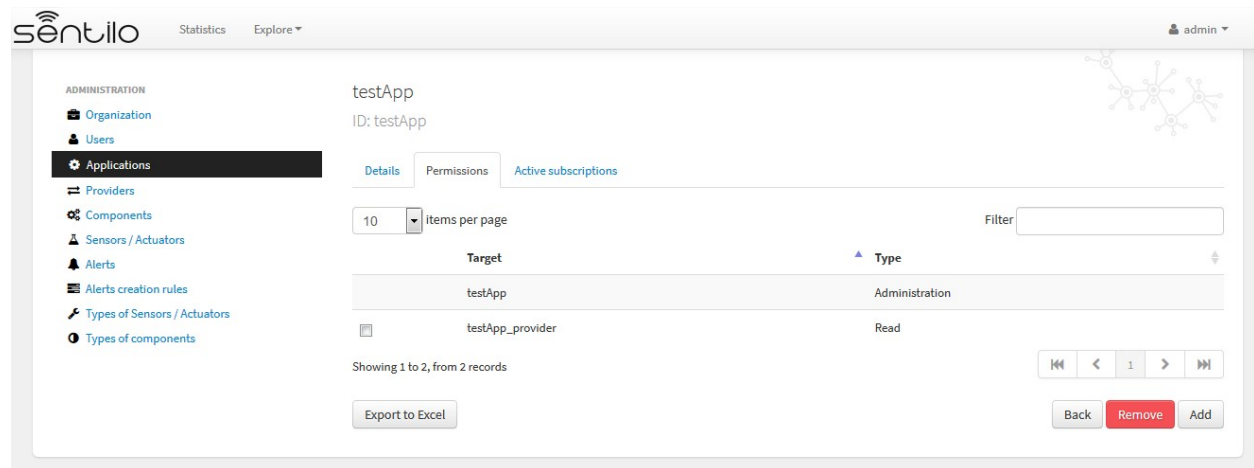
As commented before, the *Permissions* tab allows you to define and manage the authorization privileges that are granted to an application (such privileges are named *permissions*) which are required for access to the data from other entities.

There are 3 possible permissions:

- *Read*: Only allows to read the data but not modify it (e.g. cannot publish orders to sensors/actuators).
- *Read-Write*: allows to read and write data over the resources of an entity, but not administer them (e.g.. cannot create new sensors for a provider)
- *Administration*: full control over an entity and its resources.

By default, **the application sentilo-catalog has granted the Administration permission over all entities registered into Catalog** and, as you would expect, an application has full control over itself.

For example, at the following case where the permissions of the application *testApp* are displayed:



We will see the following:

- The application *testApp* could administer the entity *testApp* (obviously!)
- The application *testApp* could read any data from the entity *testApp_provider*.

Active subscriptions tab

This tab allows you to inspect the subscriptions that an application has registered on the platform (remember that subscriptions are [created with the API REST](./api_docs/services/subscription/subscription.html)), as shown in the following picture:

6.3.3 Providers

In Sentilo, providers are those who send data, i.e. those who publish the data (in contrast to applications, which consume the data). If you have loaded the default data, you will see one default provider registered into the Catalog:

- **testApp_provider**: as the name suggests, this provider is used for checking platform status.

One singularity of the providers list is the *Delete* action: **if you remove a provider, not only the provider will be deleted from the backend, but also all its related resources** such as components, sensors, alerts ... and any data published by its sensors, **so be very careful with this command**.

List

Access the Providers list. This is the main Provider page. From here you'll can access to the desired provider to show its details by click on it.

Providers

10 items per page

Filter

Identifier	Name	Description	Creation date
AGBAR	AGBAR	AGBAR	23/12/2013 14:32:23 CET
ASPB	ASPB	Agència Salut Publica Barcelona: unitat ambiental	05/10/2017 10:57:22 CEST
BCASA	BCASA	BCASA	06/11/2017 16:57:25 CET
JMC_TEST_PROVIDER	JMC_TEST_PROVIDER	JMC_TEST_PROVIDER	09/07/2015 11:44:00 CEST
METEO	METEO	Proveïdor Meteo per proves Reg Terrassa	14/01/2016 10:07:00 CET
MediAmbient	Medi Ambient	Proveïdor de control de Medi Ambient	20/03/2013 00:00:00 CET
OT_STRESS_PROV_1	OT_STRESS_PROV_1	Proveïr per a les proves de carrega	06/06/2016 16:55:28 CEST
OT_STRESS_PROV_2	OT_STRESS_PROV_2	Proveïr per a les proves de carrega	06/06/2016 16:55:28 CEST
OT_STRESS_PROV_3	OT_STRESS_PROV_3	Proveïr per a les proves de carrega	06/06/2016 16:55:28 CEST
SAMCLA	SAMCLA	SAMCLA	10/06/2013 17:15:00 CEST

Showing 1 to 10, from 22 records

Export to Excel

Delete selected

New provider

You'll be able to list, filter, show provider details, create (*New provider* button) and delete selected providers (select from left checkbox, and apply by *Delete selected* button).

Further, you'll be able to export the list to Excel, by clicking on *Export to Excel* button. The result file will contain the list columns and a number of extra ones from internal database use.

Use the button panel at the bottom right to navigate through the list (first page, [previous page](#), page number, [next page](#) and last page, respectively).

Details tab

The detail page of a provider is structured into five tabs:

testApp_provider

ID: testApp_provider

Details | Sensors / Actuators | Components | Active subscriptions | Documentation

Data

Authorization Token	563093ec5252147edc8860c2d667be5db0c010325b6953ed5b323724bcc0e05
Description	Provider to do integration tests
HTTPS API REST	<input type="checkbox"/>
Creation date	15/03/2013 08:48:42
Updated date	
Contact name	Sentilo
Contact email	sentilo@sentilo.org

Back

Edit provider

where

- The *Details* tab contains the main properties of the provider (described below).
- The *Sensors/Actuators* tab displays a list with all sensors owned by the current provider (i.e. associated with this provider).
- The *Components* tab displays a list with all components owned by the current provider (from version 1.5).
- The *Active subscriptions* tab displays a list with all the active subscriptions for the current provider.
- The *Documentation* In this tab you can upload any files relevant to provider, such as a maintenance guide, etc.

The main properties of the *Details* tab are the following:

Property	Description	Comments
Identifier	Provider identifier	Mandatory. After its creation can't be modified. It is the identifier used in the API calls.
Name	Display name	If not filled in by the user, its default value will be the <i>Id</i> .
Authorization Token	Access key	Automatically generated by the system when application is created. It is the <i>* identity_key*</i> value used in the API calls. <i>NOTE: only users with ADMIN role will show the entire token chain, other user roles only will see obfuscated text at this place (see below)</i>
Description	Description	Optional. The provider description text.
HTTPS API REST	Provider sends data over HTTPS	The Sentilo Server itself does not support SSL at the moment, however you can put a reverse proxy such as Nginx in front of the Sentilo Server. If this option is checked, the Sentilo Server expects the standard header <code>X-Forwarded-Proto</code> Please note that when configuring Nginx, you should also use the parameter <code>underscores_in_headers on;</code> so Nginx would forward sentilo headers to the Sentilo Server.
Contact name	Name of the person responsible for the provider	Mandatory
Contact email	Email address of the person responsible for the application	Mandatory.

How users that has not ADMIN role see the detail section:

ADMINISTRATION

- Applications
- Providers**
- Components
- Sensors / Actuators
- Alerts
- Alerts creation rules
- Types of Sensors / Actuators
- Types of components

testApp_provider
ID: testApp_provider

Details Sensors / Actuators Components Active subscriptions

Data

Authorization Token	*****
Description	Provider to do integration tests
HTTPS API REST	<input type="checkbox"/>
Creation date	15/03/2013 08:48:42
Updated date	
Contact name	Sentilo
Contact email	sentilo@sentilo.org

Back

Sensors/Actuators tab

As mentioned before, this tab displays a list with all sensors associated with the current provider, as shown in the picture below where the sensors of the provider CINERGIA are listed:

ADMINISTRATION

- Organization
- Users
- Applications
- Providers**
- Components
- Sensors / Actuators
- Alerts
- Alerts creation rules
- Types of Sensors / Actuators
- Types of components

Provider001
ID: Provider001

Details Sensors / Actuators Components Active subscriptions Documentation

10 items per page Filter

Sensor / Actuator	Provider	Type	Public	State	Substate	Creation date
D5_DV10	Provider001	wind_direction_10_m	true	online		27/01/2014 20:13:14
D5_ETo	Provider001	eto	true	online		27/01/2014 20:13:14
D5_HR	Provider001	humidity	true	online		27/01/2014 20:13:14
D5_HRn	Provider001	humidity	true	online		27/01/2014 20:13:14
D5_P	Provider001	atmospheric_pressure	true	online		27/01/2014 20:13:14
D5_PPT	Provider001	pluviometer	true	online		27/01/2014 20:13:14
D5_RS	Provider001	global_solar_irradiance	true	online		27/01/2014 20:13:14
D5_T	Provider001	temperature	true	online		27/01/2014 20:13:14
D5_Tn	Provider001	temperature	true	online		27/01/2014 20:13:14
D5_Tx	Provider001	temperature	true	online		27/01/2014 20:13:14

Showing 1 to 10, from 48 records

Export to Excel Back

You could filter, page and order the list but you cannot access to the sensor detail: it must be done from the sensor list administration.

Components tab

As explained early, this list is very similar to the previous one but with components.

Active subscriptions tab

The meaning of this tab is the same as described for the applications.

Documentation tab

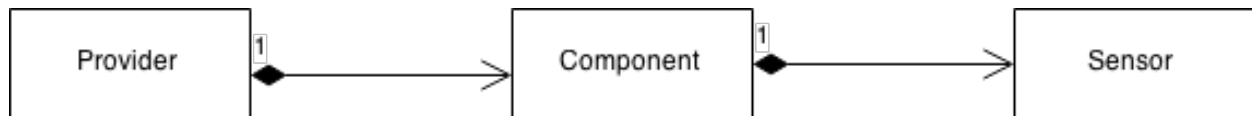
In this tab you can upload any files relevant to provider (up to 4MB each). The documents in total should not surpass ~16MB, which the [limit of MongoDB](#).

6.3.4 Components

Within the context of Sentilo, components have a special meaning: they are not linked to the API REST (except for the [catalog](#) service), i.e., components are not required to publish or read data. We use components into Catalog to group together sensors sharing a set of properties (such as location, provider, power, connectivity, ...).

You could think of them as physical devices with a set of sensors, like a weather station or a microcontroller, with multiple sensors connected. But not necessarily a component needs to have sensors physically connected to it. A gateway could also be modeled as a component: you could have a wireless sensor network ([WSN](#)) where each sensor sends data to a gateway and then it sends data to Sentilo using its Ethernet/WiFi/.. connection . In this case, the gateway will be a *component*. And finally, if you have a sensor that connects to Sentilo directly then you will have a component with only one sensor.

In short: into Sentilo, a sensor always need to be related to a component and providers have its sensors grouped by components, as shown in the following picture:



List

One singularity of the components list page are the two buttons that allows us to change the visibility of a set of components from *public* to *private* and vice versa. These buttons apply on the selected rows.

The screenshot shows the 'Components' management interface in Sentilo. The left sidebar has a menu with 'Components' highlighted. The main content area shows a table of components. The table has columns: Name, Description, Provider, Location, Type, Public, and Creation date. The table lists 10 components, all with Provider 'AGBAR' and Type 'water_meter'. Below the table, there is a pagination bar showing 'Showing 81 to 90, from 259 records' and a button panel with 'Export to Excel', 'Delete selected', 'Change access to public', 'Change access to private', and 'New component'.

Name	Description	Provider	Location	Type	Public	Creation date
11340325		AGBAR	Static	water_meter	true	10/03/2014 01:00:00 CET
11340440		AGBAR	Static	water_meter	false	10/03/2014 10:16:26 CET
11340509		AGBAR	Static	water_meter	false	10/03/2014 10:16:26 CET
11340562		AGBAR	Static	water_meter	false	10/03/2014 10:16:26 CET
11340630		AGBAR	Static	water_meter	false	10/03/2014 01:00:00 CET
11340631		AGBAR	Static	water_meter	false	10/03/2014 01:00:00 CET
11340632		AGBAR	Static	water_meter	false	10/03/2014 10:16:26 CET
11340658		AGBAR	Static	water_meter	false	10/03/2014 01:00:00 CET
11340671		AGBAR	Static	water_meter	false	10/03/2014 10:16:26 CET
11340673		AGBAR	Static	water_meter	false	10/03/2014 10:16:26 CET

You'll be able to list, filter, show components details and create (*New component* button). Like with the providers list, the component list have a *Delete* button that works as follows: ** if you remove a component, not only the component will be deleted from the backend, but also all its related resources will be deleted** such as sensors, alerts ... and any data published by its sensors, **so be very careful with this command**.

Further, you'll be able to export the list to Excel, by clicking on *Export to Excel* button. The result file will contain the list columns and a number of extra ones from internal database use.

Use the button panel at the bottom right to navigate through the `list` (first page, `<` previous page, page number, `>` next page **and** last page, respectively).

Details tab

The detail page of a component is structured into five tabs:

The screenshot displays the 'Administration' console interface. On the left is a sidebar menu with options: Organization, Users, Applications, Providers, Components (selected), Sensors / Actuators, Alerts, Alerts creation rules, Types of Sensors / Actuators, and Types of components. The main content area shows the details for component '001_D1' (ID: DEMO.001_D1). It features a tabbed interface with 'Details' selected, showing a table of properties: Organization, Type (electricity_meter), Description, Provider (DEMO), Access type (Private), Creation date (28/11/2017 09:54:57), Update date (28/11/2017 09:54:57), Tags, and Location (Static). Below the table are 'Back' and 'Edit component' buttons. A second tab, 'Location', shows a Google Map of Barcelona with a red pin indicating the component's location near the Ciutadella area.

where:

- The *Details* tab displays the main properties of the component.
- The *Technical details* tab displays several categorized properties of the component.
- The *Additional information* tab displays custom properties of the component which are not predefined by Sentilo. See the parameter `additionalInfo` of the API docs
- The *Related components* tab shows other components linked with the current component .
- The *Sensors/Actuators* tab shows the sensor element located in the current component.

The main properties of the *Details* tab are the following:

Property	Description	Comments
Name	Display name	Mandatory. After its creation can't be modified. It is the identifier used in the API calls.
Type	Component type.	Mandatory. Select from a list of available types.
Description	Description	Optional. The component description text.
Provider	Component owner	Mandatory.
Photo	URL of the component photograph	It could be defined for each component or it will be inherited using the defined one for the component type.
Access type	Checkbox to set the component visibility as public or private in the viewer	
Creation date	Creation date	Automatically generated
Update date	Last update date	Automatically generated
Tags	Related custom tags of the component	Are displayed at the public page
Static or Mobile	To mark the component as static or mobile	If the component is static then location is mandatory
Address	Address where the component is located	The address, longitude and latitude fields work together with the location list field. It's possible to use the map to set the points adding new locations.
Latitude	Latitude in decimal format	
Longitude	Longitude in decimal format	
Locations List	Location/s of the component	You can configure the component as a POI, a polyline or a polygon (<i>future feature</i>) depending the location composition.

Technical details tab

As noted above, this tab displays a set of properties related to the technical details of the component such as manufacturer, serial number, ...

[Detail](#)
[Technical details](#)
[Additional information](#)
[Related components](#)
[Sensors / Actuators](#)

Data

Producer	Solar devices
Model	So120
Serial number	S2E1195
MAC	00:FF:A4:B2:2A:31
Power type	220VAC
Connectivity type	ET_RJ45

[Back](#)
[Edit component](#)

where:

Property	Description	Comments
Producer	Manufacturer	
Model	Component model	
Serial number	Serial number	
MAC	Mac address of the device	
Power type	Energy type used by the device	Select from a list of available values (see the API for details)
Connectivity type	Connection type used by the device	Select from a list of available values (see the API for details)

Additional information tab

This tab displays the set of additional properties related to the component See the parameter `additionalInfo` of the API docs.

These fields are not categorized, i.e., here you could stored any device information which will be of interest.

For each property, it will be displayed as a *label-value* entry where the property's key will be the label and the property's value will be the value, as shown in the following picture:

[Details](#)
[Technical details](#)
[Additional information](#)
[Related components](#)
[Sensors / Actuators](#)

Data

Comarca	Alt Empordà
Terme municipal	COLERA
Provincia	Girona

where the following map, stored on the backend, has been rendered `{“Comarca”:“Alt Empordà”,“Terme municipal”:“COLERA”,“Provincia”:“Girona”}`

Sensors/actuators tab

The meaning of this tab is the same as for the providers, but restricted to the current component.

6.3.5 Sensors

This section is used for creating, updating or deleting sensors or actuators. Usually these elements are created by the provider autonomously using the API.

The sensors list page follows the same structure as described for components (you could change the public/private visibility or delete sensors massively through the list).

List

It is possible to full-text search the list in the “Filter” box. The filter works for all filter attributes except the creation date. The Filter field is case-sensitive. Only search by the substate’s code is possible at the moment.

The screenshot shows the Sentilo web interface. On the left is a sidebar with navigation links: ADMINISTRATION, Organization, Users, Applications, Providers, Components, Sensors / Actuators (selected), Alerts, Alerts creation rules, Types of Sensors / Actuators, and Types of components. The main area is titled "Sensors / Actuators" and displays a table of sensors. At the top right of the main area is a "Filter" box containing the text "Traffic". Below the table, there is a pagination bar showing "Showing 1 to 10, from 464 records" and a set of navigation buttons (first, previous, 1, 2, 3, 4, 5, next, last). At the bottom of the main area is a panel with buttons: "Export to Excel", "Delete selected" (in red), "Change state", "Change access to public", "Change access to private", and "New sensor".

Sensor / Actuator	Provider	Type	Public	State	Substate	Creation date
001_T1_TRAFFIC	Traffic	traffic	false	online	CO (Connecting)	12/09/2014 02:00:00 CEST
001_T2_TRAFFIC	Traffic	traffic	false	online	CO (Connecting)	12/09/2014 14:47:28 CEST
001_T3_TRAFFIC	Traffic	traffic	false	online	RE (Repairing)	12/09/2014 14:47:28 CEST
001_T4_TRAFFIC	Traffic	traffic	true	online	RU (Running)	12/09/2014 14:47:28 CEST
001_T5_TRAFFIC	Traffic	traffic	true	online	RU (Running)	12/09/2014 14:47:28 CEST
002_T1_TRAFFIC	Traffic	traffic	true	online		12/09/2014 14:47:00 CEST
002_T2_TRAFFIC	Traffic	traffic	true	online		12/09/2014 14:47:28 CEST
003_T1_TRAFFIC	Traffic	traffic	true	online		12/09/2014 14:47:28 CEST
003_T2_TRAFFIC	Traffic	traffic	true	online		12/09/2014 14:47:00 CEST
003_T3_TRAFFIC	Traffic	traffic	true	online		12/09/2014 14:47:28 CEST

You’ll be able to list, filter, show sensors details, and create (*New application* button) and delete selected sensors (select from left checkbox, and apply by *Delete selected* button).

Further, you’ll be able to export the list to Excel, by clicking on *Export to Excel* button. The result file will contain the list columns and a number of extra ones from internal database use.

Use the button panel at the bottom right to navigate through the list (first page, previous page, page number, next page and last page, respectively).

Details Tab

The detail page of a sensor is structured into four tabs:

The screenshot displays the 'demo_temperature_sensor' configuration page in the Sentilo Administration console. The left sidebar shows the 'Sensors / Actuators' menu item selected. The main content area has tabs for 'Details', 'Technical details', 'Visual configuration', 'Additional information', and 'Latest data'. The 'Details' tab is active, showing a table of sensor properties. The table includes fields like Organization, Sensor / Actuator, Description, Component, Access type, Creation date, Updated date, Type (temperature), Tags, Data type (Numerical), Measurement unit (°C), Time zone, State (online), and TTL (min) (1). At the bottom right, there are 'Back' and 'Edit sensor' buttons.

Data	
Organization	
Sensor / Actuator	demo_temperature_sensor
Description	
Component	provider.component
Access type	Public
Creation date	07/05/19 13:10:10 CEST
Updated date	07/05/19 13:15:39 CEST
Type	temperature
Tags	
Data type	Numerical
Measurement unit	°C
Time zone	
State	online
TTL (min)	1

where

- The *Details* tab displays the main properties of the sensor.
- The *Technical details* tab displays several categorized properties of the sensor.
- The *Additional information* tab displays the custom properties of the sensor.
- The *Latest data* tab shows the latest observations received from the sensor.

The main properties of the *Details* tab are the following:

Property	Description	Comments
Sensor / Actuator	Name of the sensor/actuator.	Mandatory. After its creation can't be modified. It is the identifier used in the API calls.
Provider	Sensor provider owner	Mandatory
Description	Description	
Component	Component to which the sensor belongs	Mandatory
Access type	Checkbox to set the sensor visibility to public or private	
Creation date	Creation date	Automatically generated
Update date	Last update date	Automatically generated
Type	Sensor type	Mandatory. Select from a list of available types
Data type	Type of data published by the sensor	Mandatory. Possible values are: <ul style="list-style-type: none"> • Audio Link • Boolean • File link • Image link • JSON • Link • Numerical • Text • Video Link
Unit	Measurement unit	
Time zone	Time zone for the data sent by the sensor	
Tags	Related custom tags of the sensor	
State	State of the sensor	Possible values: online offline. If the sensor is configured as offline the API will reject any data publication, the alerts will be disabled and the sensor won't be visible in the map. Likewise, offline sensors are excluded from the /catalog GET request. Default value is online.
Substate	Substate of the sensor	The list of possible values that have informational purpose and are specific for every deployment. You can customize the list of possible substate values editing the contents of table sensorSubstate in mongoDB. No default value.
TTL (min)	Time of expitation of the sensor in minutes	This value can be configured only from the catalog, Only admin should control this value. The default value is <code>redis.expire.data.seconds</code> from the platform server <code>jedis-config.properties</code>

Technical details tab

As noted above, this tab displays a set of properties related to the technical details of the sensor (such as the *manufacturer*, the *model*, the *serial number* and the *power type* , all of which are described in the component section) as shown in the following picture:

Data	
Producer	DEMO
Model	TAFS0
Serial number	12345
Power type	220VAC

Back Edit sensor

Visual configuration tab

The only configurable option in this tab is “Chart values number”. This integer indicates how many measures will be show in the observation chart of the sensor.

Additional information tab

The meaning of this tab is the same as for the *components*.

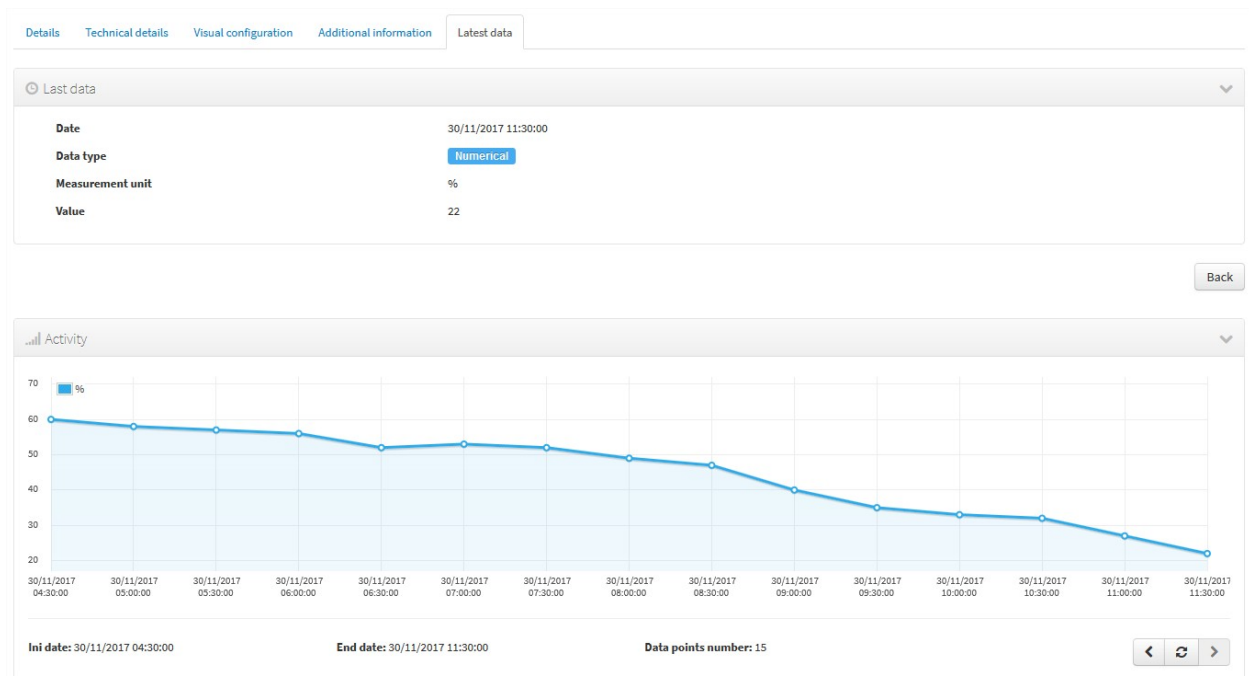
This tab displays the set of additional properties related to the component See the parameter [additionalInfo](#) of the API docs.

These fields are not categorized, i.e., here you could stored any device information which will be of interest.

For each property, it will be displayed as a *label-value* entry where the property’s key will be the label and the property’s value will be the value.

Latest data tab

This tab, as shown in the following picture:



displays both the latest observation published by the sensor and a graph with its last activity.

Navigate the last data chart

You can navigate along the dates of the graph by using the buttons located in the lower right corner of it:



- **left arrow:** navigate to the past (only if there are older data)
- **reload data (center button):** reload last data / reset chart data
- **right arrow:** navigate to the future (only if you have navigated or gone into the past before)

Number of chart observations at chart

You can change the number of values shown in the graph. To do this, within the sensor editing tabs, go to “**Visual configuration**”, and there edit the value of the “**Chart values number**” field

Edit sensor

[Details](#)
[Technical details](#)
[Visual configuration](#)
[Additional information](#)

Visual configuration

Chart values number

[Back](#)
[Save](#)

You must inform a positive value number. If blank, then default value shall be applied as that has been configured in the organization visual configuration.

Showing complex data

If your sensor data type is text, and it contains a complex data in json format, Sentilo will show it as a prettified value:

ADMINISTRATION

- Organization
- Users
- Applications
- Providers
- Components
- Sensors / Actuators**
- Alerts
- Alerts creation rules
- Types of Sensors / Actuators
- Types of components

complex-data-sensor

ID: samples-provider.sample-component.complex-data-sensor

[Details](#)
[Technical details](#)
[Visual configuration](#)
[Additional information](#)
[Latest data](#)

Last data

Date

01/12/2017 13:57:31 CE

Data type

Text

Measurement unit

Value

```
{
  id: 478952645,
  date: "2017-12-01T12:57:00.595Z",
  sensors: [
    {
      id: 1,
      type: "meteo",
      wind: {
        speed: 34,
        direction: "SW",
        date: "2017-12-01T12:57:00.595Z"
      },
      humidity: {
        level: 21
      }
    }
  ]
}
```

Activity

01/12/2017 13:57:31 CE

: { "id": 478952645, "date": "2017-12-01T12:57:00.595Z", "sensors": [{ "id": 1, "type": "meteo", "wind": { "speed": 34, "direction": "SW", "date": "2017-12-01T12:57:00.595Z", "humidity": { "level": 21 } }, { "id": 2, "type": "meteo", "wind": { "speed": 674, "direction": "SW", "date": "2017-12-01T12:57:00.596Z", "humidity": { "level": 19 } }, { "id": 3, "type": "meteo", "wind": { "speed": 4, "direction": "SW", "date": "2017-12-01T12:57:00.596Z", "humidity": { "level": 87 } }, { "id": 4, "type": "meteo", "wind": { "speed": 57, "direction": "SW", "date": "2017-12-01T12:57:00.597Z", "humidity": { "level": 65 } }, { "id": 4, "type": "meteo", "wind": { "speed": 76, "direction": "SW", "date": "2017-12-01T12:57:00.597Z", "humidity": { "level": 21 } }] }

01/12/2017 13:41:23 CE

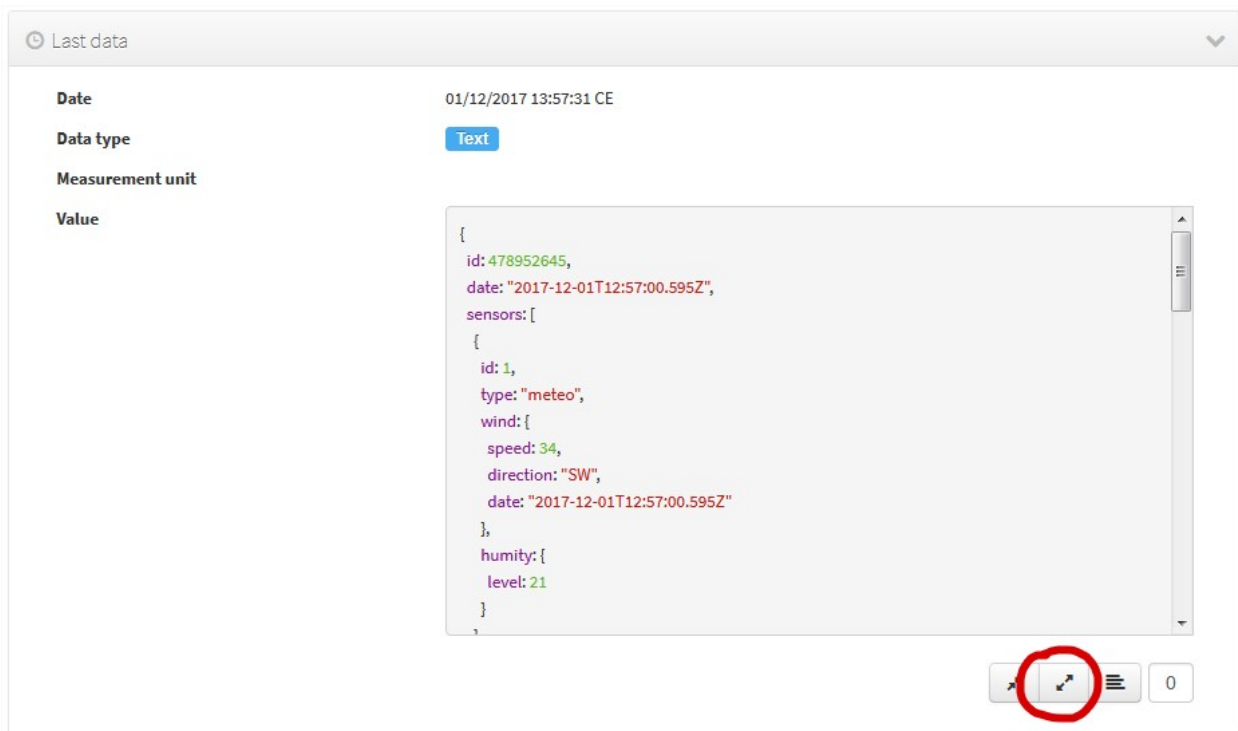
: { "id": 478952645, "date": "2017-12-01T12:37:16.610Z", "sensor": { "wind": { "speed": 34, "direction": "SW", "date": "2017-12-01T12:37:16.611Z", "humidity": { "level": 21 } }, { "speed": 674, "direction": "SW", "date": "2017-12-01T12:37:16.611Z", "humidity": { "level": 21 } }] }

in this case you will have the possibility to inspect, expand or contract the json map shown as a value using the navigation buttons:

Collapse data: the json map will be collapsed at all



Expand data: the json map will be expanded at all (default view)



Collapse to level X: insert a correct value for the X, and click the button to collapse to the specified level (default level is 0, first level)

List

Access the Alerts list. This is the main Alert page. From here you'll can access to the desired alert to show its details by click on it.

The screenshot displays the 'Alerts' management page in the Sentilo application. The sidebar on the left contains navigation links for Administration, Organization, Users, Applications, Providers, Components, Sensors / Actuators, and Alerts (which is currently selected). The main area shows a table of alerts with columns for Identifier, Typology, Trigger type, Active, and Creation date. The table lists 10 alerts, with the first one being an 'External' alert and the others being 'Internal' alerts with a trigger type of 'GT(500)'. The bottom panel includes a navigation bar with buttons for 'First', 'Previous', 'Page 1', 'Page 2', 'Page 3', 'Page 4', 'Page 5', 'Next', and 'Last', as well as an 'Export to Excel' button and a 'Delete selected' button.

Identifier	Typology	Trigger type	Active	Creation date
11	External		true	25/07/2013 00:00:00 CEST
AGBAR_D11UF034835B_GT_1460361495645	Internal	GT(500)	false	11/04/2016 09:58:15 CEST
AGBAR_D11UF150490W_GT_1460361491098	Internal	GT(500)	false	11/04/2016 09:58:11 CEST
AGBAR_D11UF150494A_GT_1460361496301	Internal	GT(500)	false	11/04/2016 09:58:16 CEST
AGBAR_D11UF150522N_GT_1460361489457	Internal	GT(500)	false	11/04/2016 09:58:09 CEST
AGBAR_D11UH030717U_GT_1460361495395	Internal	GT(500)	false	11/04/2016 09:58:15 CEST
AGBAR_D11UI050668T_GT_1460361495566	Internal	GT(500)	false	11/04/2016 09:58:15 CEST
AGBAR_D12TD107585S_GT_1460361482659	Internal	GT(500)	false	11/04/2016 09:58:02 CEST
AGBAR_D12UG029430P_GT_1460361496395	Internal	GT(500)	false	11/04/2016 09:58:16 CEST
AGBAR_D12UG081631U_GT_1460361492363	Internal	GT(500)	false	11/04/2016 09:58:12 CEST

You'll be able to list, filter, show alerts details, create (*New alert* button) and delete selected alerts (select from left checkbox, and apply by *Delete selected* button).

Further, you'll be able to export the list to Excel, by clicking on *Export to Excel* button. The result file will contain the list columns and a number of extra ones from internal database use.

Use the button panel at the bottom right to navigate through the list (first page, previous page, page number, next page and last page, respectively).

Filtering the alerts list

It is possible to full-text search the list in the "filter" box. The field is case-sensitive. That means that you can search for full or partial text contained in the identifier, type, trigger or status field. If you want to search for certain trigger type, currently only searching by trigger type's code is possible (e.g. a search for "GT" would return results in the above screen, whereas a search for "GT(40)" wouldn't).

The screenshot shows the 'Alerts' page in the Sentilo Administration console. The left sidebar contains the 'ADMINISTRATION' menu with options: Organization, Users, Applications, Providers, Components, Sensors / Actuators, Alerts (selected), Alerts creation rules, Types of Sensors / Actuators, and Types of components. The main area displays a table of alerts with columns: Identifier, Typology, Trigger type, Active, and Creation date. There are 4 records shown. A filter box contains 'test'. Below the table, it says 'Showing 1 to 4, from 4 records'. There are buttons for 'Export to Excel', 'Delete selected', and 'New alert'.

Identifier	Typology	Trigger type	Active	Creation date
<input type="checkbox"/> app_demo_provider_testSensorWind_CHANGE_1435918374799	Internal	CHANGE()	true	03/07/2015 12:12
<input type="checkbox"/> app_demo_provider_testSensorWind_FROZEN_150630155035	Internal	FROZEN(15000)	true	30/06/2015 15:50
<input type="checkbox"/> stresstest_alert	External		true	20/07/2016 10:12
<input type="checkbox"/> testAlert	External		true	25/09/2014 15:46

The screenshot shows the 'New alert' form in the Sentilo Administration console. The left sidebar is the same as the previous screenshot. The main area contains a form with the following fields:

- Identifier: testAlert1
- Name: testAlert 1
- Description: (empty text area)
- Active: ☒
- Typology: Internal (dropdown)
- Provider: app_demo_provider (dropdown)
- Component: testWind_Component (dropdown)
- Sensor id / Actuator: testSensorWind
- Expression:
 - Trigger type: Greater than (dropdown)
 - Expression to evaluate: 45

At the bottom of the form are 'Back' and 'Save' buttons.

6.3.7 Alerts creation rules

It is possible to bulk-create alerts for a group of sensors. For example, attach a rain alert rule to all pluviometers of certain provider.

List

Accessing “Alert creation rules” menu option opens a list of existing Alert Rules.

The screenshot shows the 'Alerts creation rules' page in the Sentilo application. The sidebar on the left contains the following menu items: Organization, Users, Applications, Providers, Components, Sensors / Actuators, Alerts, and Alerts creation rules (which is currently selected). The main content area is titled 'Alerts creation rules' and features a table with the following data:

Name	Provider	Component type	Sensor type	Creation date
ALERT_RULE_001	SAMCLA	meteo		01/12/2015 16:26:33 CET
ALERT_RULE_002	SMC	meteo	wind_direction_10_m	02/12/2015 12:16:13 CET
ALERT_RULE_SAMPLE	samples-provider		temperature	18/10/2016 11:57:02 CEST
ALERT_RULE_SAMPLE_2	samples-provider		luminosity	18/10/2016 12:05:45 CEST
ALERT_RULE_TITAN	TITAN		humidity	18/10/2016 11:35:44 CEST
RULE MOCK_1	METEO	meteo	atmospheric_pressure	26/10/2017 14:07:33 CEST
WATER_RULE	AGBAR	water_meter	water_meter	14/03/2016 14:43:10 CET

Below the table, it says 'Showing 1 to 7, from 7 records'. At the bottom of the page, there are three buttons: 'Export to Excel', 'Delete selected', and 'New rules'.

You’ll be able to list, filter, show alert rules details, create (*New rules* button) and delete selected rules group (select from left checkbox, and apply by *Delete selected* button).

Further, you’ll be able to export the list to Excel, by clicking on *Export to Excel* button. The result file will contain the list columns and a number of extra ones from internal database use.

Use the button panel at the bottom right to navigate through the list (first page, previous page, page number, next page and last page, respectively).

Create rules

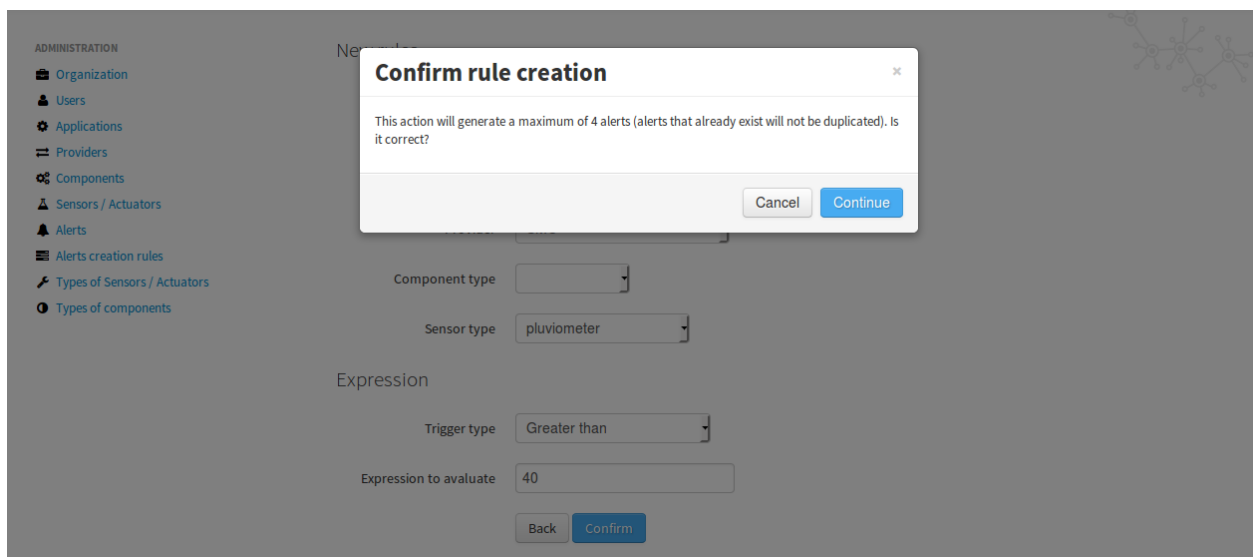
To create new alerts, use the “New Rules” button.

The screenshot shows the 'New rules' form in the Administration console. On the left is a sidebar menu under 'ADMINISTRATION' with items: Organization, Users, Applications, Providers, Components, Sensors / Actuators, Alerts, Alerts creation rules, Types of Sensors / Actuators, and Types of components. The main form has the following fields:

- Name:** rain_more_than_50
- Description:** Triggers when it rains more than 40mm.
- Provider:** SMC (selected from a dropdown)
- Component type:** (empty dropdown)
- Sensor type:** pluviometer (selected from a dropdown)
- Expression:**
 - Trigger type:** Greater than (selected from a dropdown)
 - Expression to evaluate:** 40

At the bottom of the form are two buttons: 'Back' and 'Confirm'.

After pressing the “Confirm” button, a modal window will inform on how many alerts will be created for given combination of provider, component type and sensor type.



Subsequently, alerts are created, all having the same rule. At the moment it is not possible to bulk-create alerts without specifying the provider.

To bulk-delete alerts with associated with a particular rule, just select the item from the Alert Rule list and press Delete.

6.3.8 Users

Used for creating, updating or deleting console users. It's possible to delete users massively through the elements list. There are three available roles:

- **Admin:** role for administration purposes.
- **Platform:** platform role for internal use.
- **User:** visualisation role, they could access to the administration console and read all the data, but they haven't permission for changing anything.

Properties

Id	Name	Description
Id	User identifier	After its creation can't be modified
Password	Password	
Repeat	Password check	
Name	User name	
Description	Description	
Creation date	Creation date	Automatically generated
Update date	Last update date	Automatically generated
E-Mail	User e-mail	
Active	Checkbox for removing access	
Role	Related role	Value list

List

The screenshot shows the Sentilo administration console. The top navigation bar includes the Sentilo logo, 'Statistics', 'Explore', and a user profile 'admin'. The left sidebar lists various administration options: Organization, Users (selected), Applications, Providers, Components, Sensors / Actuators, Alerts, Alerts creation rules, Types of Sensors / Actuators, and Types of components. The main content area is titled 'Users' and displays a table of user records. Above the table, there is a dropdown for '10 items per page' and a 'Filter' input field. The table has columns for 'Identifier', 'Name', 'eMail', and 'Creation date'. It lists four users: 'admin' (Administrator), 'platform_user' (Platform user), 'sadmin' (SuperAdmin user), and 'user' (User). Below the table, it indicates 'Showing 1 to 4, from 4 records'. At the bottom, there are buttons for 'Export to Excel', 'Delete selected', and 'New user'.

Identifier	Name	eMail	Creation date
<input type="checkbox"/> admin	Administrador	sentilo@sentilo.org	08/11/2013 09:26:22
<input type="checkbox"/> platform_user	Platform user	sentilo@sentilo.org	08/11/2013 14:42:40
<input type="checkbox"/> sadmin	SuperAdmin user	fill_in@your.mail	13/11/2015 13:34:38
<input type="checkbox"/> user	User	user@sentilo.org	28/11/2017 09:07:18

The screenshot shows the Sentilo Administration console interface. The top navigation bar includes the Sentilo logo, 'Statistics', 'Explore', and a user profile 'admin'. The left sidebar lists various administration options: Organization, Users (selected), Applications, Providers, Components, Sensors / Actuators, Alerts, Alerts creation rules, Types of Sensors / Actuators, and Types of components. The main content area is titled 'Edit user' and has two tabs: 'Details' and 'Config params'. The 'Details' tab is active, showing a form for editing the 'platform_user'. The form fields include: Identifier (platform_user), Password (masked with dots), Repeat (masked with dots), Name (Platform user), Description (PubSub platform user. Do not remove it!), eMail (sentilo@sentilo.org), Active (checked checkbox), and Roles (PLATFORM dropdown). At the bottom of the form are 'Back' and 'Save' buttons.

6.3.9 Sensor types

Used for creating, updating or deleting sensor types. The sensor types should be defined through the administrator console before adding elements to the catalog.

It's possible to delete elements massively through the sensor list.

Properties

Id	Name	Description
Id	Type identifier	After its creation can't be modified
Name	Display name	
Description	Description	
Creation date	Creation date	Automatically generated
Update date	Last update date	Automatically generated

List

Access the main Type of Sensors / Actuators list page, will show you a complete list of type of sensors.

The screenshot shows the 'Types of Sensors / Actuators' page in the Sentilo administration console. The sidebar on the left contains a menu with items like Organization, Users, Applications, Providers, Components, Sensors / Actuators (selected), Alerts, and Alerts creation rules. The main area displays a table with columns: Identifier, Name, Description, and Creation date. The table lists eight sensor types: anemometer, humidity, noise, pluviometer, rain, status, temperature, and wind. Each row has a checkbox on the left. Above the table, there is a 'Filter' input field and a '10 items per page' dropdown. Below the table, there is a 'Showing 1 to 8, from 8 records' message, an 'Export to Excel' button, a 'Delete selected' button, and a 'New typology' button. A pagination control at the bottom right shows '1' as the current page.

Identifier	Name	Description	Creation date
<input type="checkbox"/> anemometer	Anenometer		08/11/2013 11:27:36
<input type="checkbox"/> humidity	Humidity		08/11/2013 11:27:36
<input type="checkbox"/> noise	Noisemeter		08/11/2013 11:27:36
<input type="checkbox"/> pluviometer	Pluviometer		08/11/2013 11:27:36
<input type="checkbox"/> rain	Rain		08/11/2013 11:27:36
<input type="checkbox"/> status	status	status	10/07/2015 08:44:07
<input type="checkbox"/> temperature	Temperature		08/11/2013 11:27:36
<input type="checkbox"/> wind	Wind		08/11/2013 11:27:36

You'll be able to list, filter, show typologies details, create (*New typology* button) and delete selected typology (select from left checkbox, and apply by *Delete selected* button).

Further, you'll be able to export the list to Excel, by clicking on *Export to Excel* button. The result file will contain the list columns and a number of extra ones from internal database use.

Use the button panel at the bottom right to navigate through the list (first page, previous page, page number, next page and last page, respectively).

New

Access to create new typology pressing *New typology* button. You must inform an identifier, name and description (optional) for the new typology.

The screenshot shows the 'New typology' form in the Sentilo administration console. The sidebar on the left is the same as in the previous screenshot. The main area contains a form with three input fields: 'Identifier', 'Name', and 'Description'. Below the fields are 'Back' and 'Save' buttons. The title 'New typology' is centered at the top of the form area.

6.3.10 Component types

Used for creating, updating or deleting component types. The component types should be defined through the administrator console before adding elements to the catalog.

It's possible to delete elements massively through the component list.

Properties

Id	Name	Description
Id	Type identifier	After its creation can't be modified
Name	Display name	
Description	Description	
Creation date	Creation date	Automatically generated
Update date	Last update date	Automatically generated
Photo	Related photo	Generic picture for the component type, will be used if there isn't any specified for the component itself
Icon	Related icon	Value list from the deployed icon list. Used in the maps for representing the component

List

Access the main Component's typology list page, will show you a complete list of available type of components.

The screenshot shows the 'Component's typology' page in the Sentilo Administration console. The sidebar on the left contains links for Organization, Users, Applications, Providers, Components, Sensors / Actuators, Alerts, Alerts creation rules, Types of Sensors / Actuators, and Types of components (which is currently selected). The main content area displays a table with the following data:

Identifier	Name	Description	Creation date
electricity_meter	Electricity meter		08/11/2013 11:28:01
generic	Generic	Generic component type	08/11/2013 11:28:01
meteo	Meteo		08/11/2013 11:28:01
noise	Noise meter		08/11/2013 11:28:01

Below the table, it indicates 'Showing 1 to 4, from 4 records'. At the bottom right, there are buttons for 'Export to Excel', 'Delete selected', and 'New typology'. A filter input field is also present at the top right of the table area.


You'll be able to list, filter, show typology details, create (*New application* button) and delete selected typologies (select from left checkbox, and apply by *Delete selected* button).

Further, you'll be able to export the list to Excel, by clicking on *Export to Excel* button. The result file will contain the list columns and a number of extra ones from internal database use.

Use the button panel at the bottom right to navigate through the list (first page, previous page, page number, next page and last page, respectively).

New

Access to create new typology pressing *New typology* button. You must inform an identifier, name, description (optional), photo (optional) and icon for the new typology.

 Statistics Explore ▾

admin ▾

ADMINISTRATION

- Organization
- Users
- Applications
- Providers
- Components
- Sensors / Actuators
- Alerts
- Alerts creation rules
- Types of Sensors / Actuators
- Types of components**

New typology


Identifier

Name

Description

Photo

Icon

 ▾

Back

Save

7.1 Introduction

The new **Sentilo Multi Tenant** release provides the capacity of creating and managing virtual Sentilo instances related with different organizations(e.g. cities). Every organization has its own context, entities and data, and it can share information with third parties at its will, even it's possible to have different look & feel for every tenant.

Above all the tenants, a new role emerges for administer the platform, manage the organizations and create the necessary users for administer each one. Additionally, the platform can provide additional services to its organizations, like common integrations and offer a single map showing the public information of all its organization.

The Sentilo multi-tennancy model implements level 3 of SAAS maturity model, which offers a good levels of efficiency and scalability balanced with a complexity and a reasonable operational costs:

- Single instance for all the tenants.
- Same software deployed version for all the entities.
- Unique typologies for components and sensors.
- Common data repositories for all the tenants.
- Personalization and access control for entities through admin console.
- Personalization of look & feel for tenants.
- Delegated administration for each entity, allowing them to administer its own data, devices, users and to share data a their will.

After configuring it, every organization has its own virtual Sentilo instance and can be administered autonomously.

7.1.1 The Organization concept

The Organizations represent the different entities, usually cities, that owns a virtual Sentilo instance. Every one can manage autonomously its own applications, providers, components and sensors. All these elements are property

of the organization, and nobody outside the organization can access to them, unless the organization grants access permissions to other organizations.

Organizations are administered through the various existing user roles, and according to them, be managed in different ways:

Role	Access type
Super Admin	The Super Admin user can create and administer organizations, users and typologies
Admin	The Admin user can only manage its own organization parameters and has capacity for creating his own users, applications, providers, components, sensors and alerts, which will be automatically related to its organization
User	The User can only access to public information data about the applications, providers, components, sensors and alerts which belong to its own organization

Below you can see an organization list from a multi tenant Sentilo instance, when connecting as super admin user:

The screenshot shows the Sentilo web application interface. On the left, there is a sidebar with the 'ADMINISTRATION' menu, where 'Organizations' is selected. The main content area is titled 'Organizations' and displays a table of organization records. The table has four columns: 'Identifier', 'Name', 'Description', and 'Creation date'. It lists 10 organizations, including 'olost', 'mataro', 'vilafranca', 'vilanova', 'castelldefels', 'cornella', 'diba', 'esplugues', 'test', and 'sample_organization'. Below the table, there are pagination controls indicating 'Showing 1 to 10, from 19 records'. At the bottom, there are buttons for 'Export to Excel', 'Delete selected', and 'New organization'.

7.2 Sentilo contexts

There are several virtual contexts(URL paths) for a multi tenant Sentilo instance, one for every organization and one for the public common area. It's important to remark that for accessing to each organization console, you should choose the correct path, otherwise you won't be able to access, even using correct credentials.

7.2.1 Organization console

You should access to the administration console through the corresponding url address, adding the organization id as a last parameter, as follows.

- [http://sentilo_instance_host\[{}\]:port\[{}\]/sentilo-catalog-web/organizationId](http://sentilo_instance_host[{}]:port[{}]/sentilo-catalog-web/organizationId)

In the parameter **organizationId** you should inform the organization identifier where you want access to. For example, we could access to an organization named **Sample Organization**, with a **sample_organization** as organization identifier in a Senilo instance deployed in a host with name **example.com**:

- http://example.com/sentilo-catalog-web/sample_organization

7.2.2 Platform console

Super Admin users should access to the catalog console without informing any organization identifier in the url. In this case, no data is filtered by organization, and all the public information is visible in the public map and statistics:

- http://your_sentilo_server_ip/sentilo-catalog-web

Super Admin users are responsables of configure the platform organizations and its users, and also to define the component and sensor typologies.

7.2.3 Anonymous access

Anonymous users(not logged) can access the universal viewer directly without informing organization in the url. In this case, no data is filtered by organization, and all public information is displayed in the public maps and statistics, using the platform common look & feel.

- [http://sentilo_instance_host{\[\]:port{\[\]}/sentilo-catalog-web](http://sentilo_instance_host{[]:port{[]}/sentilo-catalog-web)

In this case, the user will see all the public information provided for the instance organizations.

Alternatively, the users can access to a specific organization public information, specifying a different URL context:

- [http://sentilo_instance_host{\[\]:port{\[\]}/sentilo-catalog-web/organizationId](http://sentilo_instance_host{[]:port{[]}/sentilo-catalog-web/organizationId)

For example, we could access to an organization named **Sample Organization**, with a **sample_organization** as organization identifier in a Senilo instance deployed in a host with name **example.com**:

- http://example.com/sentilo-catalog-web/sample_organization

Then the user will see all the public data offered by the Sample Organization, displayed using the organization custom look & feel.

For the rest of it, the features and behaviour of the public area is the same as described in [Catalog and Maps section](#).

7.3 Platform administration

Super Admin users are responsables of configure the platform organizations and its users, and also to define the component and sensor typologies. They cannot see any organization data, such as components, sensors, alerts.

7.3.1 Organization administration

List

Only the **Super Admin** user can *list, create and delete organizations*. After the organization is created, an **Admin** user can edit its own configuration settings. **User** role don't have access to this information.

ADMINISTRATION

- Organizations
- Users
- Types of Sensors / Actuators
- Types of components

Organizations

10 Items per page

Filter

Identifier	Name	Description	Creation date
olost	Aj. d'Olost	Ajuntament d'Olost	04/03/2016 14:20 CET
mataro	Aj. de Mataró	Ajuntament de Mataró	04/03/2016 11:15 CET
vilafranca	Ajuntament de Vilafranca del Penedès	Ajuntament de Vilafranca del Penedès	08/10/2015 13:54 CEST
vilanova	Ajuntament de Vilanova i la Geltrú	Ajuntament de Vilanova i la Geltrú	08/10/2015 13:53 CEST
castelldefels	Castelldefels	Ajuntament de Castelldefels	30/09/2015 15:26 CEST
cornella	Cornellà	Ajuntament de Cornellà de Llobregat	30/09/2015 15:58 CEST
diba	DIBA	Diputació de Barcelona	08/10/2015 13:32 CEST
esplugues	Esplugues de Llobregat	Ajuntament d'Esplugues de Llobregat	30/09/2015 00:00 CEST
test	Organització de test	Organització de test	03/12/2015 10:53 CET
sample_organization	Sample Organization	A sample organization	09/11/2015 12:19 CET

Showing 1 to 10, from 19 records

Export to Excel

Delete selected New organization

Details

Below, the organization creation form, as a Super Admin:

ADMINISTRATION

- Organizations
- Users
- Types of Sensors / Actuators
- Types of components

New organization

Details Config params

Identifier some_identifier

Name some_name

Description The description

Contact name somebody

Contact email somebody@mail.com

Public ☒

Back Save

In order to create an organization, we must inform, at least, these parameters:

- **identifier:** an unique organization identifier
- **name:** the organization name
- **contact name:** the name of the responsible person
- **contact email:** the email of the responsible person

Some other parameters are optional:

- **description:** some description about the organization

Config params

There are some additional parameters for customizing the public & private behavior.

Visual configuration

These params will apply to the entire catalog application visual customization, and how the user will see the data. Note that time zone & date format are directly relationated.

Property	Description	Comments
Time zone	Defines the time zone of the organization, and modifies the way to display data on screen, such as dates	You can define hourly difference or time zone abbreviations: CET, UTC, +001...
Date format	Defines the date format with which the data will be displayed in the application (lists, details...)	Example: dd/MM/yyyy HH:mm:ss = 30/11/2017 15:34:56 See all possible formats as Java Date Format, at: Java Date Format
Chart values number	Number of observations displayed on chart	It must be a positive integer number greater or equals to 10. If blank, it will be a default value of 10. This value will be overwritten by sensor's configuration one.

Map configuration

These params configure the universal map visualization.

Property	Description	Comments
Zoom level	Zoom level of the universal map	Default value is 14. And you can define a value between 1 and 20. See possible values in: https://developers.google.com/maps/documentation/static-maps/intro#Zoomlevels
Latitude / Longitude	Defines the map center in latitude & longitude values format	
Map background color	Define the background color of the map	Possible values applies with the colorpicker, or input a valid css / html color value

7.3.2 Users administration

The Super Admin user can create, edit and delete any user from any Organization whatever role they have. In Addition, Super Admin role is the unique user role that can create additional Super Admin users.

In a multi tenant instance, except for Super Admin users, when creating users, it's mandatory to specify the related organization.

List

The screenshot displays the 'Users' management page in the Sentilo application. The sidebar on the left shows the 'Users' menu item selected. The main area contains a table of users with columns for Identifier, Name, eMail, and Creation date. The table lists 10 users, including 'admin', 'asentilo', 'aterrassa', 'acornella', 'aesplugues', 'asantjoandespi', 'asantjust', 'avallirana', 'avilafranca', and 'avilanova'. The table is paginated, showing 1 to 10 of 34 records. At the bottom, there are buttons for 'Export to Excel', 'Delete selected', and 'New user'.

Identifier	Name	eMail	Creation date
admin	Admin	sentilo@sentilo.org	20/05/2015 16:29 CEST
asentilo	Admin Sentilo	xxx@xxx.com	10/11/2015 16:45 CET
aterrassa	Admin de l'Ajuntament de Terrassa	admin@ajterrassa.cat	01/10/2015 15:54 CEST
acornella	Administrador Aj. Cornellà	admin@admin.com	30/06/2017 11:28 CEST
aesplugues	Administrador Aj. Esplugues	admin@ajesplugues.cat	05/10/2015 15:17 CEST
asantjoandespi	Administrador Aj. Sant Joan Despí	admin@ajsjtoandespi.cat	22/09/2015 13:51 CEST
asantjust	Administrador Aj. Sant Just Desvern	admin@ajsjantjust.cat	29/09/2015 16:01 CEST
avallirana	Administrador Aj. Vallirana	admin@ajvallirana.cat	30/09/2015 00:00 CEST
avilafranca	Administrador Aj. Vilafranca	admin@ajvilafranca.cat	08/10/2015 13:56 CEST
avilanova	Administrador Aj. Vilanova	admin@ajvilanova.cat	08/10/2015 13:55 CEST

New user

Details

The next image shows how the new user's form is:

ADMINISTRATION

- Organizations
- Users**
- Types of Sensors / Actuators
- Types of components

New user

Details | Config params

Identifier: sample_admin

Password: *****

Repeat: *****

Name: Sample Admin User

Description: Sample Admin User

eMail: sample@admin.com

Active: ☒

Roles: ADMIN

Organization: Sample Organization

Alternatively, we can inform some configuration params that will modify the catalog visualization for this user:

ADMINISTRATION

- Organizations
- Users**
- Types of Sensors / Actuators
- Types of components

New user

Details | **Config params**

Visual configuration

Time zone: CET

Date format: dd/MM/yyyy HH:mm:ss

Back Save

These params will apply to the entire catalog application visual customization, and how the user will see the data. Note that time zone & date format are directly relationated.

Property	Description	Comments
Time zone	Defines the time zone of the user, and modifies the way to display data on screen, such as dates	You can define hourly difference or time zone abbreviations: CET, UTC, +001... Example: dd/MM/yyyy HH:mm:ss = 30/11/2017 15:34:56* Note that this value overrides the organization's one, if informed*
Date format	Defines the date format with which the data will be displayed in the application (lists, details...)	See all possible formats as Java Date Format, at: Java Date Format Note that this value overrides the organization's one, if informed

7.3.3 Component and Sensor types administration

Only the Super Admin user can administer the components and sensor types. In this case, the behaviour is the same like for a normal Sentilo instance.

See more information about it in the [\[\[Catalog and Maps section>>doc:Catalog & Maps\]\]](#).

7.4 Tenant administration

Admin role users are directly related with an specific organization. They are the only ones who are able to administer the organization private data, such as its providers, applications, components, sensors and alarms. They can also manage its own users. The admin users will also be capable of seeng all the defined component and sensor typologies, but they wont be able to modify them.

Basically, the only difference between the simple Sentilo instance and a Multi Tenant instace version is that only users from one organization can see and access to information from its own organization. It's also possible to share information with another organizations, as described later. This data isolation make possible taking advantage of user and organization hierarchy.

Below, we review the specific behaviour of tenant administration, remarking is specificities. For more information, you can read the [\[\[Catalog and Maps>>doc:Catalog & Maps\]\]](#) section.

7.4.1 Organization administration

Admin users can only manage its own organization information.

Data	
Description	A sample organization
Contact name	Somebody
Contact email	somebody@example.com
Public	true
Creation date	09/11/2015 11:19:56
Creation user	sadmin
Modification date	09/11/2015 11:19:56
Update user	sadmin

Back Edit organization

Alternatively, Admin user can manage their config params and third party from/to permissions for sharing information purposes. You'll find them in the two last tabs that located in the top of the detail section.

Permission administration

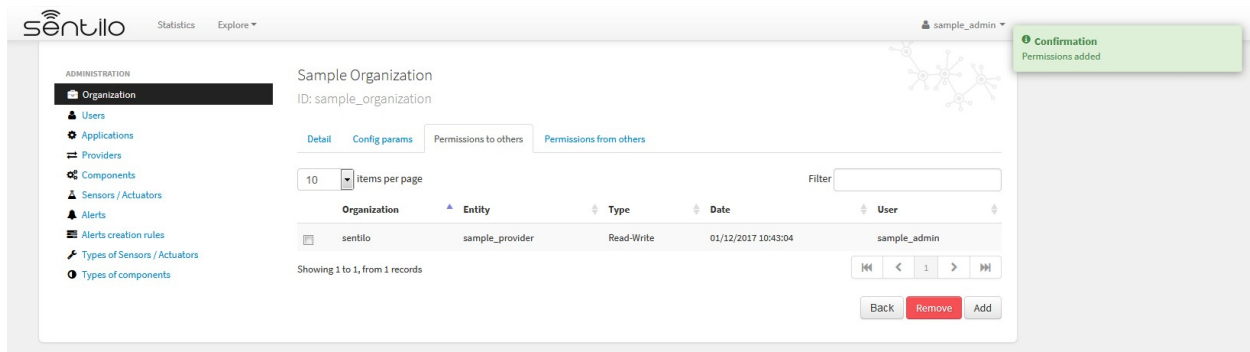
Permission type	Functionality
To third party organizations from us	Grant read / write permissions to other organizations over our providers (and dependent components / sensors / alerts). We can add and drop these permissions.
From third party organizations to us	Read / Write permissions from third party organizations granted to us. We can only make them visible or not in the universal map.

Permissions list

Adding permissions

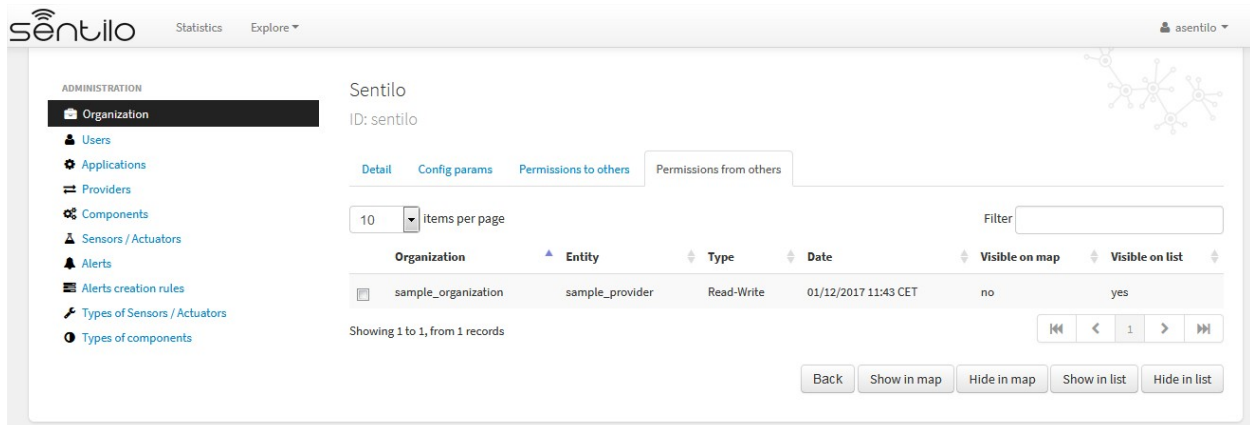
Adding to third party read & write permission:

Response upon permission has been created:



In this case we had granted **read&write** permissions from our organization and our provider **sample_provider** to third party organization named **Sentilo**. So, now the Sentilo organization can access to the sample_provider data and manage it (publish data).

In the other side, the Sentilo organization can see these permissions in the second tab, *Permissions from others*:



And now, from this tab, we can change the permission visibility on the map. Simply select the checkbox from the permission and click on **Show in map** or **Hide in map**.

When sharing providers with other organizations, their related entities(providers, components, sensors), will appear on the other tenant console, but only in read mode.

7.5 Tenant resources administration: unique identifiers

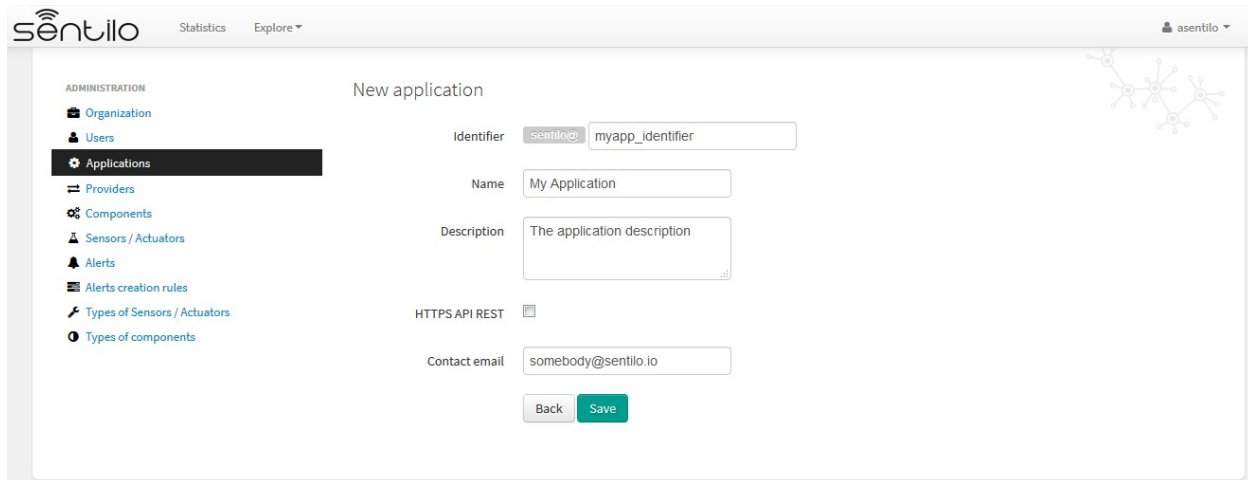
Resources related to a tenant, such like *providers* and *applications*, must have unique identifier into a Sentilo instance. But, in a multi tenant instance, it is possible to repeat it identifier, based on its tenant. So tenant resources are completely independent between their tenants.

Multi tenant instances offers to the user a little visual difference. You will inform the resource identified with its own tenant identifier as prefix.

It is transparent for users, but in administration console you'll see a flag that informs you that you're in a multi tenant instance: **sentilo@the_identifier**, is related to an identifier from Sentilo tenant organization

7.5.1 Applications

For application creation form you'll see this in the Identifier field:

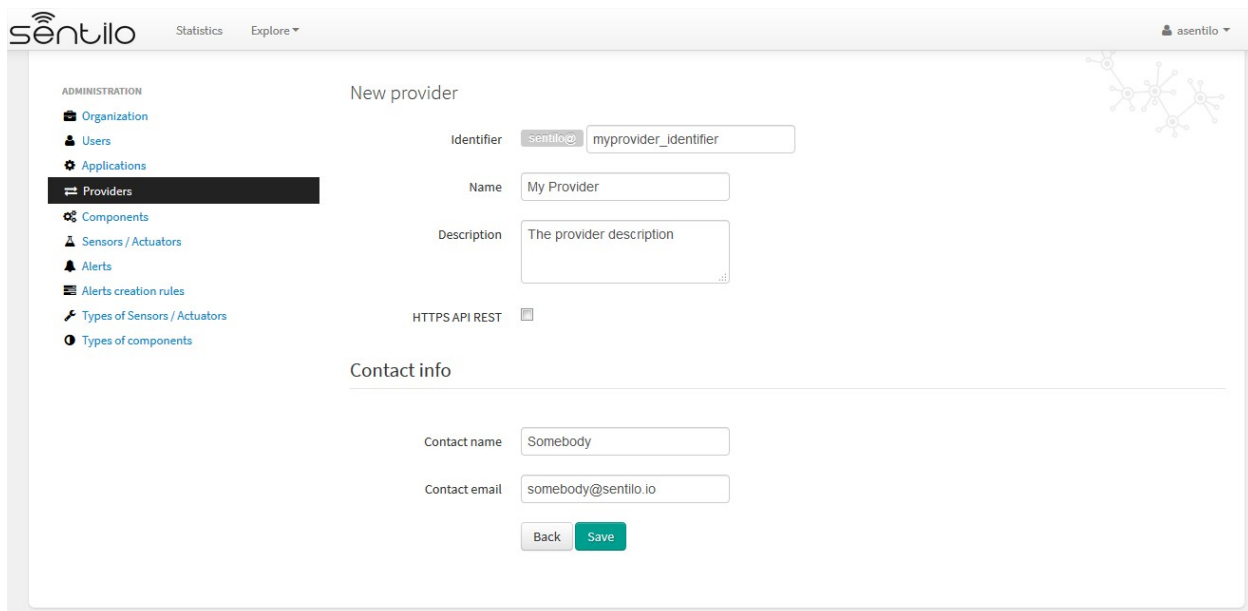


The screenshot shows the 'New application' form in the Sentilo administration interface. The left sidebar contains a list of administrative options: Organization, Users, Applications (highlighted), Providers, Components, Sensors / Actuators, Alerts, Alerts creation rules, Types of Sensors / Actuators, and Types of components. The main form area has the following fields: Identifier (with a dropdown set to 'sentilo@' and a text input 'myapp_identifier'), Name (text input 'My Application'), Description (text area 'The application description'), HTTPS API REST (checkbox, unchecked), and Contact email (text input 'somebody@sentilo.io'). At the bottom are 'Back' and 'Save' buttons.

In this case, we're creating a application named **My Application** with identifier **myapp_identifier**.

7.5.2 Providers

For the providers, we would be facing the same case as for the Applications. Therefore, you can choose the desired identifier, regardless of the tenant you are managing.



The screenshot shows the 'New provider' form in the Sentilo administration interface. The left sidebar contains the same list of administrative options as the previous screenshot, with 'Providers' highlighted. The main form area has the following fields: Identifier (with a dropdown set to 'sentilo@' and a text input 'myprovider_identifier'), Name (text input 'My Provider'), Description (text area 'The provider description'), HTTPS API REST (checkbox, unchecked), and a 'Contact info' section with Contact name (text input 'Somebody') and Contact email (text input 'somebody@sentilo.io'). At the bottom are 'Back' and 'Save' buttons.

In this case, we're creating a provider named **My Provider**, with identifier **myprovider_identifier**.

Contents:

8.1 Java Client



The Sentilo Java Client is a library developed for working with webapps or standalone java applications. You can download the source code from <https://github.com/sentilo/sentilo-client-sample-java>.

For this example, we'll use a basic maven based web application, that retrieve some system data and send it as a sensor observation to the Sentilo Platform. This webapp is named **sentilo-client-java-sample** and you'll find it into the source code (see above).

8.1.1 Hardware

We don't need any specific hardware for running this example, only one PC with Internet connection.

8.1.2 Software

You'll need some software packages, as you're developing in Java environment:

- Java SE 1.8
- Eclipse IDE or STS Spring IDE
- The **Sentilo Client Java Library**, which you can download and install it as a *Maven dependency* into your project (please, see the **pom file** in the project form more information)
- Tomcat 7+
- Some other Maven dependencies (you can see them in the **pom file** of the example project):
 - Hibernate
 - Spring
 - And some other...

8.1.3 The example

The code

You must download the sample webapp project from Git repository: <https://github.com/sentilo/sentilo-client-sample-java>

Once you have the project, open it with Eclipse or another IDE and construct it using *Maven* goals: **clean package**, for downloading dependencies, compile the code and package it.

After that your project is compiled and packaged, you can deploy it in a Tomcat 7 webapp container. We recommend that you use Eclipse or STS Spring IDE to develop and open this example, and deploy it with the Apache deployment plugins.

Now you can then navigate into the project and edit the source code.

The properties file

You must modify the properties file **application.properties** located in **src/main/resources/properties** in order to provide your correct Sentilo Platform Client configurations. There're some values that are for testing purposes, and they may not be changed.

```
# Sentilo Platform API Services IP
rest.client.host=YOUR_SENTILO_PLATFORM_CLIENT_ADDRESS

# User configurations
rest.client.identityKey=YOUR_IDENTITY_KEY
rest.client.provider=samples-provider
rest.client.component=sample-component
rest.client.component.type=generic
rest.client.component.location=41.387015 2.170047
rest.client.sensor=sample-sensor-java
rest.client.sensor.type=status
rest.client.sensor.dataType=TEXT
rest.client.sensor.location=41.387015,2.170047
```

This settings should be updated:

- **rest.client.host**: provide a correct host or ip address of your Sentilo Platform Client, and replace the *YOUR_SENTILO_PLATFORM_CLIENT_ADDRESS* with it

- **rest.client.identityKey:** provide your correct application security token, and replace the *YOUR_IDENTITY_KEY* with it
- *optionally*, you can provide your component / sensor locations, modifying the values **rest.client.component.location** and **rest.client.sensor.location**

The samples controller

There's a Spring MVC controller which displays a view with the sensor data retrieved from system and the publish result. Navigate to **src/main/java** and open this resource **org.sentilo.samples.controller.SamplesController**.

This is a Spring Framework Controller that creates a view where you'll see a sample data value obtained from the System, and then send it as observation to your Sentilo Platform instance. The webapp is based on Maven & Spring foundations, so you must modify and provide some configurations before start the example execution (see above).

```
/**
 * SamplesController
 *
 * Executes a basic Sentilo Java Client Platform example which connects to the server,
 * and publish some data to a sample sensor.
 * In this case we're getting info from the system with the runtime properties object
 *
 * @author openTrends
 */
@Controller
public class SamplesController {

    private final Logger logger = LoggerFactory.getLogger(SamplesController.class);

    private static final String VIEW_SAMPLES_RESPONSE = "samples";

    private static final int SLEEP_TIME = 1;

    @Autowired
    private PlatformTemplate platformTemplate;

    @Resource
    private Properties samplesProperties;

    @RequestMapping(value = {"/", "/home"})
    public String runSamples(final Model model) {

        // All this data must be created in the Catalog Application before starting this
        // sample execution. At least the identity key and the provider id must be
        // declared in the system
        String restClientIdentityKey = samplesProperties.getProperty("rest.client.
        identityKey");
        String providerId = samplesProperties.getProperty("rest.client.provider");

        // For this example we have created a generic component with a status sensor that
        // accepts text
        // type observations, only for test purpose
        String componentId = samplesProperties.getProperty("rest.client.component");
        String sensorId = samplesProperties.getProperty("rest.client.sensor");

        logger.info("Starting samples execution...");
```

(continues on next page)

(continued from previous page)

```

String observationsValue = null;
String errorMessage = null;

try {
    // Get some system data from runtime
    Runtime runtime = Runtime.getRuntime();
    NumberFormat format = NumberFormat.getInstance();
    StringBuilder sb = new StringBuilder();
    long maxMemory = runtime.maxMemory();
    long allocatedMemory = runtime.totalMemory();
    long freeMemory = runtime.freeMemory();

    sb.append("free memory: " + format.format(freeMemory / 1024) + "<br/>");
    sb.append("allocated memory: " + format.format(allocatedMemory / 1024) + "<br/>");
    sb.append("max memory: " + format.format(maxMemory / 1024) + "<br/>");
    sb.append("total free memory: " + format.format((freeMemory + (maxMemory -
    allocatedMemory)) / 1024) + "<br/>");

    // In this case, we're getting CPU status in text mode
    observationsValue = sb.toString();

    logger.info("Observations values: " + observationsValue);

    // Create the sample sensor, only if it doesn't exists in the catalog
    createSensorIfNotExists(restClientIdentityKey, providerId, componentId,
    sensorId);

    // Publish observations to the sample sensor
    sendObservations(restClientIdentityKey, providerId, componentId, sensorId,
    observationsValue);
} catch (Exception e) {
    logger.error("Error publishing sensor observations: " + e.getMessage(), e);
    errorMessage = e.getMessage();
}

logger.info("Samples execution ended!");

model.addAttribute("restClientIdentityKey", restClientIdentityKey);
model.addAttribute("providerId", providerId);
model.addAttribute("componentId", componentId);
model.addAttribute("sensorId", sensorId);
model.addAttribute("observations", observationsValue);

ObjectMapper mapper = new ObjectMapper();

try {
    if (errorMessage != null && errorMessage.length() > 0) {
        Object json = mapper.readValue(errorMessage, Object.class);
        model.addAttribute("errorMsg", mapper.writerWithDefaultPrettyPrinter().
    writeValueAsString(json));
    } else {
        model.addAttribute("successMsg", "Observations sended successfully");
    }
} catch (Exception e) {
    logger.error("Error parsing JSON: {}", e.getMessage(), e);
    errorMessage += (errorMessage.length() > 0) ? "<br/>" : "" + e.getMessage();
}

```

(continues on next page)

(continued from previous page)

```

        model.addAttribute("errorMsg", errorMessage);
    }

    return VIEW_SAMPLES_RESPONSE;
}

/**
 * Retrieve catalog information about the sample provider. If the component and/or
 * sensor doesn't
 * exists, it will create them
 *
 * @param identityToken Sample identity token
 * @param providerId Samples provider id
 * @param componentId Samples component id
 * @param sensorId Samples sensor id
 * @return {@link CatalogOutputMessage} object with provider's catalog data
 */
private CatalogOutputMessage createSensorIfNotExists(String identityToken, String
providerId, String componentId, String sensorId) {
    List<String> sensorsIdList = new ArrayList<String>();
    sensorsIdList.add(sensorId);

    // Create a CatalogInputMessage object for retrieve server data
    CatalogInputMessage getSensorsInputMsg = new CatalogInputMessage();
    getSensorsInputMsg.setProviderId(providerId);
    getSensorsInputMsg.setIdentityToken(identityToken);
    getSensorsInputMsg.setComponents(createComponentsList(componentId));
    getSensorsInputMsg.setSensors(createSensorsList(providerId, componentId,
sensorsIdList));

    // Obtain the sensors list from provider within a CatalogOutputMessage response
    CatalogOutputMessage getSensorsOutputMsg = platformTemplate.getCatalogOps().
getSensors(getSensorsInputMsg);

    // Search for the sensor in the list
    boolean existsSensor = false;
    if (getSensorsOutputMsg.getProviders() != null && !getSensorsOutputMsg.
getProviders().isEmpty()) {
        for (AuthorizedProvider provider : getSensorsOutputMsg.getProviders()) {
            if (provider.getSensors() != null && !provider.getSensors().isEmpty()) {
                for (CatalogSensor sensor : provider.getSensors()) {
                    logger.debug("Retrieved sensor: " + sensor.getComponent() + " - " +
sensor.getSensor());
                    existsSensor |= sensorId.equals(sensor.getSensor());
                    if (existsSensor) {
                        break;
                    }
                }
            }
        }
    }

    // If the sensor doesn't exists in the retrieved list, we must create it before
    publishing the
    // observations
    if (!existsSensor) {

```

(continues on next page)

(continued from previous page)

```

        // Create a CatalogInputMessage object for retrieve server data
        CatalogInputMessage registerSensorsInputMsg = new
↪CatalogInputMessage(providerId);
        registerSensorsInputMsg.setIdentityToken(identityToken);
        registerSensorsInputMsg.setComponents(createComponentsList(componentId));
        registerSensorsInputMsg.setSensors(createSensorsList(providerId, componentId,
↪sensorsIdList));

        // Register the new sensor in the server
        platformTemplate.getCatalogOps().registerSensors(registerSensorsInputMsg);
    }

    return getSensorsOutputMsg;
}

/**
 * Publish some observations from a sensor
 *
 * @param identityToken Samples Application identity token for manage the rest
↪connections
 * @param providerId Samples provider id
 * @param componentId Samples component id
 * @param sensorId Samples sensor id
 * @param value Observations value, in our case, a String type
 */
private void sendObservations(String identityToken, String providerId, String
↪componentId, String sensorId, String value) {
    List<String> sensorsIdList = new ArrayList<String>();
    sensorsIdList.add(sensorId);
    createSensorsList(providerId, componentId, sensorsIdList);

    List<Observation> observations = new ArrayList<Observation>();
    Observation observation = new Observation(value, new Date());
    observations.add(observation);

    SensorObservations sensorObservations = new SensorObservations(sensorId);
    sensorObservations.setObservations(observations);

    DataInputMessage dataInputMessage = new DataInputMessage(providerId, sensorId);
    dataInputMessage.setIdentityToken(identityToken);
    dataInputMessage.setSensorObservations(sensorObservations);

    platformTemplate.getDataOps().sendObservations(dataInputMessage);
}

/**
 * Create a component list
 *
 * @param componentId Component identifier
 * @return A {@link CatalogComponent} list
 */
private List<CatalogComponent> createComponentsList(String componentId) {
    List<CatalogComponent> catalogComponentList = new ArrayList<CatalogComponent>();
    CatalogComponent catalogComponent = new CatalogComponent();
    catalogComponent.setComponent(componentId);
    catalogComponent.setComponentType(samplesProperties.getProperty("rest.client.
↪component.type"));

```

(continues on next page)

(continued from previous page)

```

        catalogComponent.setLocation(samplesProperties.getProperty("rest.client.component.
↪location"));
        catalogComponentList.add(catalogComponent);
        return catalogComponentList;
    }

    /**
     * Create a sensor list
     *
     * @param componentId The Sample Component Id
     * @param sensorsIdList A list with the sensor ids to create
     * @return A {@link CatalogSensor} list
     */
    private List<CatalogSensor> createSensorsList(String providerId, String componentId,
↪ List<String> sensorsIdList) {
        List<CatalogSensor> catalogSensorsList = new ArrayList<CatalogSensor>();
        for (String sensorId : sensorsIdList) {
            CatalogSensor catalogSensor = new CatalogSensor();
            catalogSensor.setComponent(componentId);
            catalogSensor.setSensor(sensorId);
            catalogSensor.setProvider(providerId);
            catalogSensor.setType(samplesProperties.getProperty("rest.client.sensor.type"));
            catalogSensor.setDataType(samplesProperties.getProperty("rest.client.sensor.
↪dataType"));
            catalogSensor.setLocation(samplesProperties.getProperty("rest.client.sensor.
↪location"));
            catalogSensorsList.add(catalogSensor);
        }
        return catalogSensorsList;
    }
}

```

What's happening?

- First of all, we're looking for some configuration settings, like the component and sensor names
- Next, we're using some runtime status values, so we can the publish them as a observations (mem status, for example)
- First of all, we check if the sensor has been created before in the Catalog, and if it doesn't exists we add it
- After that, we'll publish the sensor observations
- Then, we pass all this information to the view for displaying it the navigator window

This is an observation sample:

```
CPU states: 5.8% user, 1.9% system, 0.0% nice, 0.0% wait, 91.7% idle
```

The samples page view

And finally, this is the source code of the view:

```

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ page contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>

```

(continues on next page)

(continued from previous page)

```
<html>

<head>

</head>

<body>

  <h3>Observations:</h3>
  <p>${observations}</p>

  <br />

  <c:if test="${not empty successMsg}">
    <h3>Success:</h3>
    <p>${successMsg}</p>
  </c:if>

  <c:if test="${not empty errorMsg}">
    <h3>Error:</h3>
    <pre>${errorMsg}</pre>
  </c:if>

  <br />

  <button onclick="location.reload();">Send observations</button>

</body>

</html>
```

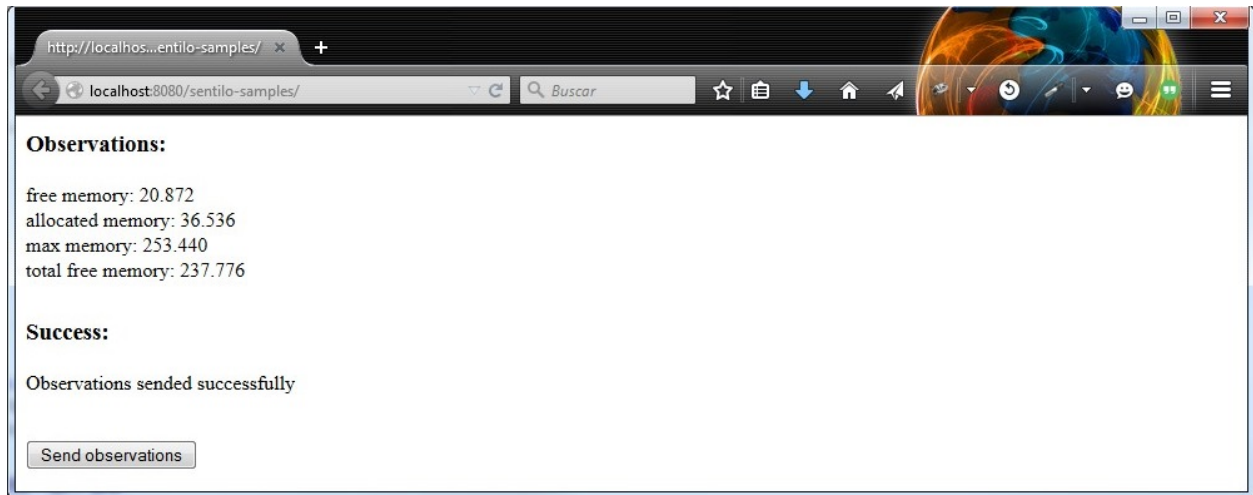
This source code is quite easy, so don't need to comment it.

Executing the sample application

Using the Eclipse IDE or copying the WAR file, deploy your webapp into the Tomcat deployments directory, and start it.

You must access to this url (we assume that you're in your localhost and your port is the 8080, the default values): <http://localhost:8080/sentilo-samples>

And then you must see a result page like this:



As you can see, there's a button named *Send observations*. You can use it to re-send observations and reload the page. Every page reload sends the observations to the Sentilo Platform Client.

8.2 RaspberryPi Client



The **SentiloClientNodej** is a library written in javascript that brings to you the possibility to connect any device and embedded device with Node.js to a Sentilo instance easily.

For this example we'll use a Raspberry pi device, because its special features, like cheap price, small dimensions and the great input-output possibilities.

8.2.1 Hardware

We need some hardware to execute this example:

Material	Description
Raspberry Pi	Of course, we need a Raspberry Pi, doesn't matters its version. You need a Raspbian distro installed too
Ethernet cable or wifi dongle	Doesn't matter what network hardware do you use, it's up to you, but you'll need the correct drivers correctly configured for connecting to Internet
Some cables	Some "dupont" male-to-female cables to make breadboard connections
A LED	We'll use a LED as an output for simulating order execution, you can pick any color
A resistor	A 470 Ohms 1/4W resistance
A breadboard	You'll need an electronic connections base to make some circuits, so we'll recommend you to get a breadboard

8.2.2 Software

We need to install some node.js alternative modules or libraries. We'll install them through *npm* or download them from the Internet. See the next section for more information.

8.2.3 Setup the Raspi

In this example we'll show to you how to implement a sensor and a actuator with several capabilities such as:

- get values from one o more sensors connected to it GPIO port and publish them in Sentilo
- receive orders from a Sentilo application and actuate on the GPIO in consequence

We're assuming that you have yet installed Node.js in your system and your user has privileges to manage the GPIO. If not, you can try an easy tutorial like the described in [this link](#).

NOTE: we recommend that you have installed latest versions of **Node.js** and **npm**. You can test it as follow:

```
pi@raspberrypi ~/sentilo $ node -v
v0.12.1
pi@raspberrypi ~/sentilo $ npm -v
2.5.1
```

The software

Create the workspace

First of all, create a directory named **sentilo** (for example) in our user home and change to it:

```
pi@raspberrypi ~ $ mkdir sentilo && cd sentilo
```

Here we'll work with all our files.

Install the SentiloClientNodejs library

Download the library, that allow to us to access some Sentilo operations easily.

You may download it from the [Sentilo Git repository](#), and clone it into your working directory:

```
pi@raspberrypi ~/sentilo $ git clone https://github.com/sentilo/sentilo-client-nodejs
```

After that, you may have a directory structure like this into the **sentilo-client-nodejs** directory:

```
pi@raspberrypi ~/sentilo $ ls -la sentilo-client-nodejs
total 28
drwxr-xr-x 4 pi pi 4096 nov 17 13:00 .
drwxr-xr-x 3 pi pi 4096 nov 17 13:00 ..
drwxr-xr-x 8 pi pi 4096 nov 17 13:00 .git
-rw-r--r-- 1 pi pi 1352 nov 17 13:00 LICENSE
-rw-r--r-- 1 pi pi 257 nov 17 13:00 package.json
-rw-r--r-- 1 pi pi 3079 nov 17 13:00 README.md
drwxr-xr-x 3 pi pi 4096 nov 17 13:00 src
```

Here we have the core library (the **src** directory) and now we can write down the example files (see below).

Install some library dependencies

We need some Node.js modules to work with our library. So you need to download and install the through *npm*.

You can download via npm update:

```
pi@raspberrypi ~/sentilo/sentilo-client-nodejs $ npm
```

Or install them directly:

```
pi@raspberrypi ~/sentilo/sentilo-client-nodejs $ npm install restify
pi@raspberrypi ~/sentilo/sentilo-client-nodejs $ npm install sync-request
pi@raspberrypi ~/sentilo/sentilo-client-nodejs $ npm install onoff
```

Ok, but, what are they?

- **restify** is a rest server interface for Node.js that allow to us to create a rest server easily ([see this link](#) for more information)
- **sync-request**, allow to us to create synchronous http calls ([see this link](#) for more information)
- **onoff**, a GPIO driver that allow to us to turn on and off a LED very easily! ([see this link](#) for more information)

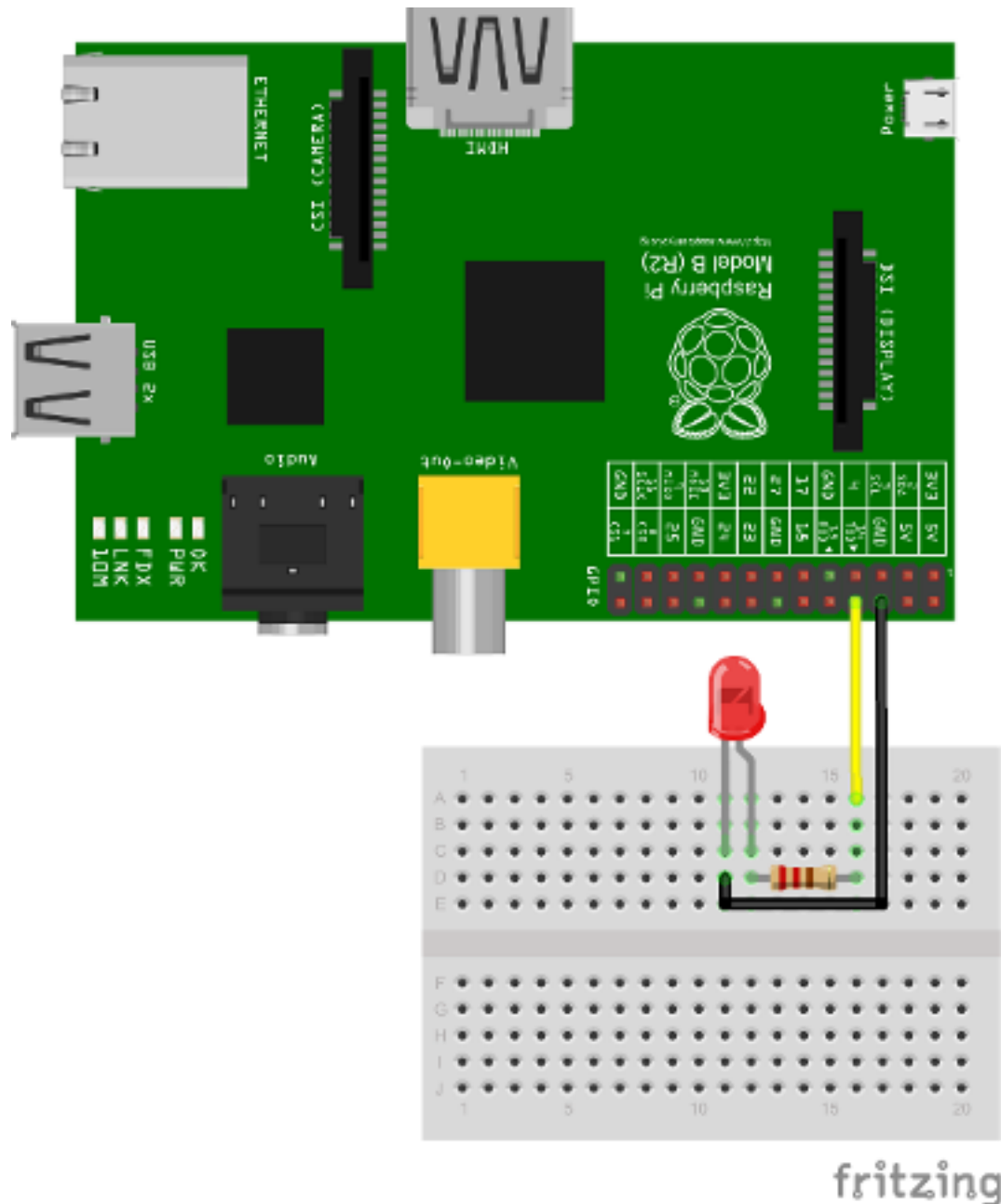
Now, we can start to develop with node in our working directory

Connecting the hardware

We'll assume that you have your Raspberry Pi connected to the Internet, a Raspbian system correctly installed and enough permissions to create and runs scripts in your user home. Usually, we will use the **pi** user.

To simulate a sensor's actuator, we'll use a LED connected to the GPIO 14 from the Raspberry Pi.

Let see it below:



As you can see, we only connect the **LED anode** to the **14 GPIO** pin, and the **kathode** to the **GND** pin. We're ready to turn ON and OFF this LED from the Sentilo platform.

8.2.4 The example

Our example is a complete suit to test the `SentiloClientNodejs` library.

The library allow to you to performs these operations:

- Catalog
 - Get sensors from a provider
 - Register sensors for a provider

- Update sensors configuration
- Register alerts
- Alarm
 - Publish alarms
- Data
 - Retrieve last observations from a sensor
 - Publish observations
- Subscription
 - Subscribe orders from a sensor
 - Subscribe orders for all the sensors of a provider

In our example, we'll use almost of the to show you how interact with Sentilo platform

The code

We've created our working directory, and connected all the necessary hardware. Now we can take a look to the example code.

The next files was included into the library directory, so you have they into your working directory.

Note that you can download these all example files from <https://github.com/sentilo/sentilo-client-sample-nodejs>.

actuator.js

This file contains code to control the output LED, that performs a simulation of a possible actuator controlled by the Raspberry Pi and orders published on the Sentilo platform. In the example, the system is being subscribed as endpoint to receive incoming order actuation calls.

```
var Gpio = require('onoff').Gpio;
var led;

/**
 * Sample module that perfoms operations over the actuator
 */
module.exports = {

  /**
   * Initializes the actuator. For this example, we've connected a LED as a
   * output in the GPIO 14
   */
  init : function() {
    // Configure the GPIO as OUTPUT
    led = new Gpio(14, 'out');

    console.log('Turning OFF the LED');

    // Turn off the LED on startup
    led.writeSync(0);
  },
```

(continues on next page)

(continued from previous page)

```

/**
 * Execute an order in the actuator. For this example, we've mounted a LED
 * in GPIO 14, where we can turn it ON (order=ON) or OFF (order=OFF)
 *
 * @param order
 *       A Sentilo order object structure
 */
executeOrder : function(order) {
    console.log('Executing order: ' + order.message);

    if (order.message === 'ON') {
        console.log('Turning ON the LED');
        led.writeSync(1);
    } else if (order.message === 'OFF') {
        console.log('Turning OFF the LED');
        led.writeSync(0);
    }

}

};

```

What's happening here?

- The **init** function configures the GPIO 14 as an output to control the LED (don't forget to call it!)
- The **executeOrder** function performs the actuator order execution, for our case, lets turn ON and OFF the LED if the order was ON or OFF. The main server code will access to this function to actuate over the sensor actuator.

sensor.js

This file contains the code that performs a possible sensor data read and return its value. For our example, we only emulate a random value as a possible sensor data value. You may develop an data input sensor with a GPIO library.

```

module.exports = {

    /**
     * Retrieve data from the sensor
     *
     * @returns {String}
     */
    readSensorValue : function() {

        // TODO: Implement this method

        // Return some random value between 0 and 255
        var sensorValue = Math.floor((Math.random() * 255));

        return sensorValue;
    }

};

```

What's happening here?

- The **readSensorValue** function reads a possible sensor input data value from an external way. The main server code will access to this function to read the sensor data.

NOTE: as you see, you must implement this function to complete your requirements

sentilo.js

This file is a little interface that wraps the general calls to the library. Its function is make more easy the interaction between the server and the library. It isn't really necessary, but is a good method to modularize the code.

```
var servicesConfig = require('./sentiloclient/ServicesConfiguration');
var logger = require('./sentiloclient/Utils/SentiloLogs');
var utils = require('./sentiloclient/Utils/SentiloUtils');

var catalog = require('./sentiloclient/CatalogServiceOperations');
var data = require('./sentiloclient/DataServiceOperations');
var alarm = require('./sentiloclient/AlarmServiceOperations');
var subscribe = require('./sentiloclient/SubscriptionServiceOperations.js');

module.exports = {

  /**
   * Initialize the services with default and custom options
   */
  init : function(initOptions) {
    // Initialize the services
    catalog.init(initOptions);
    data.init(initOptions);
    alarm.init(initOptions);
    subscribe.init(initOptions);

    logger.debug("Samples module initialization successful");
  },

  /**
   * Search a sensor in the catalog
   *
   * @return boolean
   */
  existsSensorInCatalog : function(options) {
    // Get all sensors from provider
    var response = catalog.getSensors(options);

    // The params of the example
    var provider = options.provider;
    var sensor = options.sensor;

    // Look the desired sensor in the catalog...
    var existsSensor = false;
    if (response && response.providers) {
      var providers = response.providers;
      for (var p = 0; p < providers.length; p++) {
        var provider = providers[p];
        if (provider.sensors && provider.sensors.length > 0) {
          var sensors = provider.sensors;
          for (var s = 0; s < sensors.length; s++) {
            var sensor = sensors[s];
            if (sensor === options.sensor) {
              existsSensor = true;
              break;
            }
          }
        }
      }
    }
  }
}
```

(continues on next page)

(continued from previous page)

```

    }
  }
}

logger.debug('Exists the \'' + provider + '\'' and \'' + sensor + '\'' in the_
↪catalog? ' + existsSensor);

return existsSensor;
},

/**
 * Create a sensor
 */
createSensor : function(options) {
  logger.debug('Adding the sensor \'' + options.sensor + '\'' to the catalog...
↪');

  // Create an input message to inform the new sensor data
  // We are using the sample data, defined in ServicesConfiguration module
  var inputMessage = {
    body : {
      sensors : [ {
        sensor : options.sensor,
        description : options.sensorDesc,
        type : options.sensorType,
        dataType : options.sensorDataType,
        unit : options.sensorUnit,
        component : options.component,
        componentType : options.componentType,
        location : options.sensorLocation
      } ]
    }
  };

  logger.debug(inputMessage);

  var response = catalog.registerSensors(inputMessage);
  if (response && response.code && response.code === 400) {
    logger.error('Error registering the sensors');
    logger.error(response);
    return false;
  } else {
    return true;
  }
},

/**
 * Publish observations
 */
publishObservations : function(value, options) {
  var observationsInputMessage = {
    body : {
      observations : [ {
        value : value
      } ]
    }
  }
}

```

(continues on next page)

(continued from previous page)

```

    }
    };

    observationsInputMessage = utils.mergeOptions(observationsInputMessage, ↵
↵options);

    var response = data.sendObservations(observationsInputMessage);
    if (response && response.code && response.code === 400) {
        logger.error('Error publishing observations');
        logger.error(response);
        return false;
    } else {
        return true;
    }
},

/**
 * Create an alert list
 */
createAlerts : function(alertsList) {
    var alertsInputMessage = {
        body : {
            alerts : alertsList.alerts
        }
    };

    var response = catalog.registerAlerts(alertsInputMessage);
    if (response && response.code && response.code === 400) {
        logger.error('Error registering alerts');
        logger.error(response);
        return false;
    } else {
        return true;
    }
},

/**
 * Publish an alarm
 */
publishAlarm : function(alert, inputMessage) {
    var alarmInputMessage = {
        body : {
            message : inputMessage.message
        }
    };

    var response = alarm.publish(alert, alarmInputMessage);
    if (response && response.code && response.code === 400) {
        logger.error('Error publishing alarm');
        logger.error(response);
        return false;
    } else {
        return true;
    }
},

/**

```

(continues on next page)

(continued from previous page)

```

    * Subscribe to a sensor order
    */
    subscribeOrder : function(inputMessage) {
        var subscriptionInputMessage = {
            body : {
                endpoint : inputMessage.endpoint
            }
        };

        var response = subscribe.subscribe(subscriptionInputMessage);
        if (response && response.code && response.code === 400) {
            logger.error('Error subscribing order');
            logger.error(response);
            return false;
        } else {
            return true;
        }
    },

    /**
     * Subscribe to all sensors orders from a provider
     */
    subscribeOrderToAll : function(inputMessage) {
        var subscriptionInputMessage = {
            body : {
                endpoint : inputMessage.endpoint
            }
        };

        var response = subscribe.subscribeToAll(subscriptionInputMessage);
        if (response && response.code && response.code === 400) {
            logger.error('Error subscribing order');
            logger.error(response);
            return false;
        } else {
            return true;
        }
    }
};

```

Here you can see direct calls to the library, so the only differences are several logs and the initialization of some configuration variables.

server.js

This is the main node file. This file performs a rest server and still waiting for incoming calls that apply to the actuator.

```

// A very simple RESTful server module
var restify = require('restify');

// The OS information module
var os = require('os');

// Include some Sentilo operations
var sentilo = require('./sentilo.js');

```

(continues on next page)

(continued from previous page)

```
// Module that interacts with the local sensor
var sensor = require('./sensor.js');

// Module that interacts with the local actuator
var actuator = require('./actuator.js');
actuator.init();

// Get some OS values, like the sensor IP
var interfaces = os.networkInterfaces();
var addresses = [];
for (var k in interfaces) {
  for (var k2 in interfaces[k]) {
    var address = interfaces[k][k2];
    if (address.family === 'IPv4' && !address.internal) {
      addresses.push(address.address);
    }
  }
}

var myIp = addresses[0];
var myPort = 8080;
var myEndpoint = 'http://' + myIp + ':' + myPort;
var myOrderEndpointPath = '/order';
var myOrderEndpoint = myEndpoint + myOrderEndpointPath;

console.log('My ip address is: ' + myIp + ', and my port: ' + myPort);

// Service and example options
// You must modify it under your requirements
var samplesOptions = {
  host : 'YOUR_SERVER_HOST',
  port : 'YOUR_SERVER_PORT',
  headers : {
    identity_key : 'YOUR_IDENTITY_KEY'
  },
  tokenId : 'YOUR_IDENTITY_KEY',
  providerTokenId : 'YOUR_PROVIDER_IDENTITY_KEY',
  provider : 'samples-provider',
  sensor : 'sample-sensor-nodejs',
  component : 'sample-component',
  componentType : 'generic',
  sensorDataType : 'TEXT',
  sensorType : 'status',
  sensorUnit : '',
  sensorLocation : 'YOUR_SENSOR_LOCATION'
};

// Starts a RESTful server to manage orders inputs via POST calls
var server = restify.createServer({
  name : 'SentiloClient for Nodejs Example Server',
  version : '1.0.0'
});
server.use(restify.acceptParser(server.acceptable));
server.use(restify.queryParser());
server.use(restify.bodyParser());

// We only need a POST endpoint service to receive ordercs callbacks
```

(continues on next page)

(continued from previous page)

```
// The path will be [POST] http://localhost:8080/order
server.post('/order', function(req, res, next) {
    res.send(req.params);

    console.info("[POST] Order received: " + JSON.stringify(req.params));

    // Execute the order in the actuator
    actuator.executeOrder(req.params);

    var value = 'Order received and executed: ' + JSON.stringify(req.params.message);
    sentilo.publishObservations(value, samplesOptions);

    return next();
});

// Starts the server and listen on port 8080
server.listen(myPort, function() {
    console.log('%s listening at %s', server.name, myEndpoint);
    console.log('The server is now ready to receive POST incoming calls');
});

// Init Sentilo services for this example
// Here you must pass as parameter the specific configuration
sentilo.init(samplesOptions);

// Test if is there the sensor configured in the catalog
var existsSensor = sentilo.existsSensorInCatalog(samplesOptions);
if (!existsSensor) {
    // If not, then create it
    sentilo.createSensor(samplesOptions);
}

// Now we can publish a first alarm that informs that the sensor is up
// First of all let create an external alert
console.log('Registering the System Status Alert...');
var alertsListInputMessage = {
    alerts : [ {
        id : 'SYSTEM_STATUS_ALERT',
        name : 'SYSTEM_STATUS_ALERT',
        description : 'Custom alert to inform the system status',
        type : 'EXTERNAL'
    } ]
};
sentilo.createAlerts(alertsListInputMessage);

// And then, we can publish an alarm to inform that the system is up now
var alarmInputMessage = {
    message : 'The system goes up on ' + new Date()
};
sentilo.publishAlarm('SYSTEM_STATUS_ALERT', alarmInputMessage);
console.log('Alarm published: ' + alarmInputMessage.message);

// Subscribe the sensor orders
// We'll manage it through our server on POST service
var subscriptionInputMessage = {
    endpoint : myOrderEndpoint
};
```

(continues on next page)

(continued from previous page)

```

sentilo.subscribeOrder(subscriptionInputMessage);
// sentilo.subscribeOrderToAll(subscriptionInputMessage);

// Now, we can publish observations every 60 seconds
// And still waiting for incoming orders
var systemObservationsTimeout = 60000;
console.log('The sensor is now up, and we\'ll be sending some observations every ' +
↪systemObservationsTimeout + ' ms');
setInterval(function() {
    // Send some System information
    var freeMemValue = "OS freemem: " + os.freemem();
    console.log('Retrieved system freemem value: [' + freeMemValue + '] and
↪publishing it as an observation...');
    sentilo.publishObservations(freeMemValue, samplesOptions);

    // Retrieve some sensor data and send it as observation...
    var sensorDataValue = "Sensor value: " + sensor.readSensorValue();
    console.log('Retrieved sensor value: [' + sensorDataValue + '] and publishing it
↪as an observation...');
    sentilo.publishObservations(sensorDataValue, samplesOptions);
}, systemObservationsTimeout);

```

First of all, we'll see the configuration options. They must be changed before run this example.

You must provide the correct values for these variables located into the **samplesOptions** object:

- **YOUR_SERVER_HOST**: provide the correct **ip address** or host of your Sentilo's instance rest server
- **YOUR_SERVER_PORT**: provide the correct **port** of your Sentilo's instance rest server
- **YOUR_IDENTITY_KEY**: you must provide your **private security key** (*tokenId*) that identifies your **application**. Remember that this application must have ADMIN permissions over all yours providers, components and sensors for this example
- **YOUR_PROVIDER_IDENTITY_KEY**: like above, you must provide your **provider's security token id**
- **YOUR_SENSOR_LOCATION**: this is optional, identifies the component location of the sample sensor

All the other configurations and variables are correctly coded and you don't need to change any more.

Now, what's happens in this code?

- First of all, we're start a **rest server** with the *restify* module, that allow to us to provide an endpoint for incoming order calls (POST method). After that, we create a subscription for our orders.
- When a POST call is received, the server will call the **actuator's executeOrder function**, so we can manage the order correctly (turn ON/OFF the LED, for example)
- Initialize the **sentilo's helper module** (as you can see above), implemented by the *sentilo.js* file
- We're passing to it our specific services configuration, like the sensor id, provider's token, etc. . .
- Request for the sensor in the Sentilo Catalog platform, and if it doesn't exists, create it
- Once we have created the sensor, we're creating an alert, named **SYSTEM_STATUS_ALERT**, and publishing an initial alarm that says **The system goes up on {date}**. Then, the sensor is up and we're informing it to the system
- After that, retrieve some system and sensor data values and publish them every 60000ms (1 minute) in a infinite loop

Executing the example

Now we can finally execute the example.

Simple type:

```
pi@raspberrypi ~/sentilo/sentilo-client-nodejs $ node server.js
Turning OFF the LED
My ip address is: 127.0.0.1, and my port: 8080
Registering the System Status Alert...
Alarm published: The system goes up on Thu May 07 2015 13:52:21 GMT+0000 (UTC)
The sensor is now up, and we'll be sending some observations every 60000 ms
SentiloClient for Nodejs Example Server listening at http://127.0.0.1:8080
The server is now ready to receive POST incoming calls
```

And now, the server is waiting for publish the observations every 60 seconds:

```
Retrieved system freemem value: [OS freemem: 846716928] and publishing it as an
↳ observation...
Retrieved sensor value: [Sensor value: 64] and publishing it as an observation...
```

Publishing and accepting orders

The server also is waiting for incoming POST calls that responses the ORDER requests. You can practice with orders, sending a PUT message to the Sentilo platform, some like this:

```
http://sentilo_platform_ip:8081/order/samples-provider/sample-sensor-nodejs
```

With these values:

```
HEADER > identity_key : 'YOUR_IDENTITY_KEY'
BODY   > {"order" : "ON"} > this turns ON the LED
BODY   > {"order" : "OFF"} > this turns OFF the LED
```

After that, you'll see in the console some log like this when you're turning the LED ON, sending **order = ON**:

```
[POST] Order received: {"message":"ON","timestamp":"07/05/2015T13:58:20","topic":"/
↳ order/samples-provider/sample-sensor-nodejs","type":"ORDER","sensor":"sample-sensor-
↳ nodejs","provider":"samples-provider","sender":"samples-provider","time
↳ ":1431007100595}
Executing order: ON
Turning ON the LED
```

Or turning it OFF, with **order = OFF**:

```
[POST] Order received: {"message":"OFF","timestamp":"07/05/2015T14:01:13","topic":"/
↳ order/samples-provider/sample-sensor-nodejs","type":"ORDER","sensor":"sample-sensor-
↳ nodejs","provider":"samples-provider","sender":"samples-provider","time
↳ ":1431007273310}
Executing order: OFF
Turning OFF the LED
```

Debugging the library

If you need to debug your execution, you can edit the file `/sentiloclient/utis/SentiloLogs.js` and edit the logs configuration properties, as you need:

```
var options = {
  className : 'Sentilo',
  enableLogs : true,
  enableDebug : false,
  enableInfo : true,
  enableWarn : true,
  enableError : true,
  enableFatal : true
};
```

For our purpose, we only have DEBUG, INFO and ERROR logs. Try to use **true** or **false** for each one.

8.3 Arduino Client



The **SentiloClient Library** for Arduino offers a basic C++ library implementation that allows to the developer a quick integration sketch with the Sentilo Platform through its API Rest Client.

For these examples we'll create a new sensor in the Sentilo Platform, only if it doesn't exists, and then we're going to publish some observations obtained from the local sensors.

8.3.1 Hardware

We'll need some hardware materials:

Material	Description
> Arduino board	We recommend that you use a Mega 2560 board, which brings to you a bit more program memory than Uno.
> Official Ethernet Shield for Arduino	The library is based on the communication layer that provides the Official Ethernet Shield. Basically, it is a http rest client module.
Some sensors	For these examples we'll use two types of sensors: a <i>LM35</i> temperature sensor, and a basic <i>LDR</i> brightness sensor (photocell)
Resistences	Two 1KOhm resistences 1/4W
Breadboard	A breadboard that allow to you the quick connection of the electronic components
Wires	Connection wires, like ' <i>dupont</i> ' male-to-male ones, that brings to you easily connection between the Arduino board and the breadboard and the sensors
Ethernet cable	A cable for connecting the Ethernet Shield to the Internet through your router o modem
B type USB cable	A cable for connecting your Arduino board to the PC

8.3.2 Software

We'll use the **Official Arduino IDE**, that you can download from [here](#). Of course, you'll need a PC, with Windows or Linux, or a Mac computer.

8.3.3 Setup the Arduino

The Arduino IDE

We assume that you have installed the **Arduino IDE** and you know how to use it :). If not, then you can have a look on this [link](#).

Download and install the library

Download our **SentiloClient** library or clone it from Git via this link: <https://github.com/sentilo/sentilo-client-arduino>, and then install it as a custom library into your Arduino IDE. If you don't know how to install custom libraries, you can have a look on this [link](#), see the **Importing a .zip Library** section.

8.3.4 The example

First example: publishing a basic observation

Once you have installed the library into the Arduino IDE, you can go to the menu option **File > Examples > Sentilo-Client > SentiloClient-Example-01** and open the sample code. In this example Arduino is going to connect to the network and publish a basic observation with these contents: **"This is a sample observation"**.

Sentilo configuration

You must have configured this information in the Sentilo catalog:

- A provider (in our case, named **samples-provider**) and its token
- A component (in our case, named **sample-component**)
- A sensor (in our case, named **sample-sensor-arduino-01** for the first example, and another one named **sample-sensor-arduino-02** for the second one), with this minimum configuration settings:


```

sensor = sample-sensor-arduino-01
type = status
dataType = TEXT
component = sample-component
componentType = generic

```

Then, you must replace the client connection data code (next section) with yours settings:

- Change the value “YOUR_API_KEY” with the api key of your provider (variable *apiKey*)
- Change the value “YOUR_IP_ADDRESS” with the ip address of your Sentilo instance (variable *ip*)
- Change the value “YOUR_PORT” with the port of your Sentilo server instance port (variable *port*)

The code

You’ll should see this code in the editor:

```

#include <Ethernet.h>
#include <SPI.h>

#include "SentiloClient.h"

/**** SENTILO ****/
char* apiKey = "YOUR_API_KEY";
char* ip = "YOUR_IP_ADDRESS";
int port = YOUR_PORT;
char* componentId = "sample-component";
char* providerId = "samples-provider";
char* sensorId = "sample-sensor-arduino-01";

// The Sentilo Client object
SentiloClient sentiloClient = SentiloClient(ip, port);

/**** NETWORK ****/
const int networkConnectionTimeout = 30;

/**** GLOBAL VARS ****/
const int generalCalibrationTimeout = 1000; // Wait after system setup is complete
String response = ""; // Rest call response (normaly as JSON message)
int statusCode = -1; // Rest call return code (the HTTP code)

void setup() {
    // Begin serial for debug purposes
    Serial.begin(9600);

    // Setup the Sentilo Client and the network connection
    setupSetiloClient();

    // Wait time for a general calibration
    delay(generalCalibrationTimeout);

```

(continues on next page)

(continued from previous page)

```

}

void loop() {
    // Create the Observation object
    SentiloClient::Observation observation;
    observation.value = "This is a sample observation";

    Serial.println("[loop] Publishing a sample observation...");

    // Publish the observation to Sentilo Platform
    statusCode = sentiloClient.publishObservation(providerId, sensorId,
↪observation, apiKey, response);

    // Read response status and show an error if it is necessary
    if (statusCode != 200) {
        Serial.print("[loop] [ERROR] Status code from server after publish_
↪the observations: ");
        Serial.println(statusCode);
        Serial.print("[loop] [ERROR] Response body from server after publish_
↪the observations: ");
        Serial.println(response);
    }

    Serial.println("[loop] Sample observation published!");
    Serial.println("[loop] Program ended");

    // The example has ended, so we are going to execute an infinite loop
    while (true) {}
}

/** Setup the Sentilo Client object, this process also configures the network_
↪connection */
void setupSentiloClient() {
    Serial.print("[setup] Connecting to network via DHCP ");
    sentiloClient.dhcp();
    for (int i = 0; i < networkConnectionTimeout; i++) {
        Serial.print(".");
        delay(100);
    }
    Serial.println(" done!");
    Serial.println("[setup] Connection is now established!");
}

```

What can we see in this example?

- We are setting up the Serial channel for debug output
- Setup the **SentiloClient** object (**sentiloClient**), which configures the client and connects to the network
- Once we're connected to the server, we publish a basic observation, with these contents: *"This is a sample observation"*
 - If the publish works properly, the system don't return any special data
 - Otherwise, it will show to you the system return code and message, if it is possible
- The test ends after publish only one observation

Second example: publishing sensors data as observations

In this case we'll retrieve data from sensors (LDR and LM35), and then we'll publish them as a observation, with a message in JSON format, like that:

```
{ "ldr": "{ldrValue}", "lm35": "{lm35Value}" }
```

Where the **ldrValue** contains the LDR photocell value, and the **lm35Value** contains the LM35 temperature value. Open the sample code in **File > Examples > SentiloClient > SentiloClient-Example-02**.

Connect the sensors and and other connections

Now, it is the time to connect the sensors and others elements.

See below:



In the upper image, you can see how the components has been located:

- Connect the *positive* pin from **Arduino (+5V)** to the upper channel of the breadboard (*red channel*)
- Connect the *negative* pin from **Arduino (GND)** to the second channel of the breadboard (*blue channel*)
- **LDR** photocell sensor connection:
 - Connect the LDR photocell between **GND signal and A0** (Analog IO 0 from Arduino) with a dupont wire, in this case, the orange color wire
 - Connect the LDR pin that holds the orange wire with a 1KOhm resitor, and the other resistor pin to **Arduino +5V** (red wire)
- **LM35** temperature sensor:
 - Connect the LM35 **positive pin** (left pin, front side) to **Arduino +5V**
 - Connect the LM35 **center pin** (signal) to the A5 (Analog IO 5 from Arduino) with a dupont wire, in this case, the orange color wire
 - Connect the LM35 **negative pin** (right pin, front side) to **Arduino GND**

The code

You should see this code in the editor:

```
#include <Ethernet.h>
#include <SPI.h>

#include "SentiloClient.h"

/***** SENSORS *****/
```

(continues on next page)

(continued from previous page)

```

int LDR = 0; // LDR input is A0
int LM35 = 5; // LM35 input is A5
const int ldrSetupTimeout = 10; // Time that LDR needs to be configured (dummy time)
const int lm35SetupTimeout = 10; // Time that LM35 needs to be configured (dummy time)

/***** SENTILO *****/
/***** NETWORK *****/
/***** GLOBAL VARS *****/
char* apiKey = "YOUR_API_KEY";
char* ip = "YOUR_IP_ADDRESS";
int port = YOUR_PORT;
char* componentId = "sample-component";
char* providerId = "samples-provider";
char* sensorId = "sample-sensor-arduino-02";

// The Sentilo Client object
SentiloClient sentiloClient = SentiloClient(ip, port);

/***** NETWORK *****/
const int networkConnectionTimeout = 30;

/***** GLOBAL VARS *****/
const int generalCalibrationTimeout = 1000; // Wait after system setup is complete
const int loopTimeout = 60000; // Loop timeout, time between observations (in ms)
String response = ""; // Rest call response (normally as JSON message)
int statusCode = -1; // Rest call return code (the HTTP code)

boolean existsSensor = false;

void setup() {
    // Begin serial for debug purposes
    Serial.begin(9600);

    // Setup the LDR sensor
    setupLDR();

    // Setup the LM35 sensor
    setupLM35();

    // Setup the Sentilo Client and network connection
    setupSentiloClient();

    // Wait time for a general calibration
    delay(generalCalibrationTimeout);
}

void loop() {
    // Get the LDR value
    int ldrValue = getLdrValue();

    // Get the LM35 value
    float lm35Value = getLM35Value();

```

(continues on next page)

(continued from previous page)

```

// Create the observation input message like this: {"ldr":"234","lm35":"24.5"}
String obsInputMsg =
    "{\\"ldr\\":\\" + String(ldrValue) +
    "\\","\\lm35\\":\\" + String(lm35Value) +
    "\\}\\\"";

int bufLength = obsInputMsg.length() + 1;
char obsMsgBuffer[bufLength];
obsInputMsg.toCharArray(obsMsgBuffer, bufLength);

// Create the Observation object
SentiloClient::Observation observation;
observation.value = obsMsgBuffer;

// Debug on Serial the observations value. Note that we must scape special
↳characters
Serial.print("[loop] Publishing actual sensors values as observations: ");
Serial.println(obsMsgBuffer);

// Publish the observation to Sentilo Platform
statusCode = sentiloClient.publishObservation(providerId, sensorId,
↳observation, apiKey, response);

// Read response status and show an error if it is necessary
if (statusCode != 200) {
    Serial.print("[loop] [ERROR] Status code from server after publish
↳the observations: ");
    Serial.println(statusCode);
    Serial.print("[loop] [ERROR] Response body from server after publish
↳the observations: ");
    Serial.println(response);
} else {
    Serial.println("[loop] Sensors observations published!");
}

delay(loopTimeout);
}

/** Emulate a possible LDR initialization process, if it is necessary */
void setupLDR() {
    Serial.print("[setup] Setting up the LDR brightness sensor ");
    for (int i = 0; i < ldrSetupTimeout; i++) {
        Serial.print(".");
        delay(100);
    }
    Serial.println(" done!");
    delay(50);
}

/** Get the brightness value from th LDR */
int getLdrValue() {
    return analogRead(LDR);
}

/** Emulate a possible LM35 initialization process, if it is necessary */
void setupLM35() {
    Serial.print("[setup] Setting up the LM35 temperature sensor ");
    for (int i = 0; i < lm35SetupTimeout; i++) {

```

(continues on next page)

(continued from previous page)

```

        Serial.print(".");
        delay(100);
    }
    Serial.println(" done!");
    delay(50);
}

/** Get the LM 35 temperature value in Celsius degrees */
float getLM35Value() {
    int val = analogRead(LM35);
    float mv = (val / 1024.0) * 5000;
    float cel = mv / 10;
    //float farh = (cel * 9) / 5 + 32;
    return cel;
}

/** Setup the Sentilo Client object. This process also configures the network
↳connection */
void setupSetiloClient() {
    // Connect via DHCP
    Serial.print("[setup] Connecting to network via DHCP ");
    sentiloClient.dhcp();
    for (int i = 0; i < networkConnectionTimeout; i++) {
        Serial.print(".");
        delay(100);
    }
    Serial.println(" done!");
    Serial.println("[setup] Connection is now established!");
}

```

What can we see in this example? There're some additions compared with the first example.

- Setup Arduino and the SentiloClient is the same of the first sample
- We're making a sensors setup, but in this case it isn't necessary, so it only informs us in debug mode what is happening in every moment...
- Into the loop
 - We're retrieving the LDR and LM35 values, and putting them into variables
 - Once we've retrieved the sensors data, we're mounting the new observation message, with value: `{"ldr": "{ldrValue}", "lm35": "{lm35Value}"}`
 - The SentiloClient library gets the value and transforms it on a complete **observation message** using the **publishObservation** method (see below)
- The sketch loops sleeps until **loopTimeout** millis has been reached, and then turns up and repeats the same process of data publication (in this example the sleep time is 60000ms, 1 minute per loop / publish)

This is the observation sent by to the Sentilo platform:

```

{"observations": [{
    "value": "{\"ldr\": \"{ldrValue}\", \"lm35\": \"{lm35Value}\"}"
  }]
}

```

If you want, you can include the **timestamp** variable in UTC format inside the observation object:

```

Observation observation;
observation.value = {"ldr":"382","lm35":"23.4"};
observation.timestamp = "05/05/2015T12:34:45";

```

And the message will be generated as:

```

{"observations": [{
  "value": {"ldr":"382","lm35":"23.4"},
  "timestamp": "05/05/2015T12:34:45"
}]
}

```

As you can see, the library object `Observation` (struct type) offers you an abstraction. In the next sample we will see them in working together with the `Sensor` object.

Third example: initialize sensor, create it in the catalog and publish observations continuously

In this third example we'll see that how the `SentiloClient` library can create a sensor “on-the-fly” and publish observations continuously. Next, we'll use the second example, plus a little bit of additional code that help us to check if the sensor exists in the catalog, and if not create it before publish observations. Open the sample code in **File > Examples > SentiloClient > SentiloClient-Example-03**.

The code

You should see this code in the editor:

```

#include <Ethernet.h>
#include <SPI.h>

#include "SentiloClient.h"

/*****/
/*****/ SENSORS *****/
/*****/
int LDR = 0; // LDR input is A0
int LM35 = 5; // LM35 input is A5
const int ldrSetupTimeout = 10; // Time that LDR needs to be configures (dummy time)
const int lm35SetupTimeout = 10; // Time that LM35 needs to be configures (dummy time)

/*****/
/*****/ SENTILO *****/
/*****/
char* apiKey = "YOUR_API_KEY";
char* ip = "YOUR_IP_ADDRESS";
int port = YOUR_PORT;
char* componentId = "sample-component";
char* providerId = "samples-provider";
char* sensorId = "sample-sensor-arduino-03";

// The Sentilo Client object
SentiloClient sentiloClient = SentiloClient(ip, port);

/*****/
/*****/ NETWORK *****/

```

(continues on next page)

(continued from previous page)

```

/*****
const int networkConnectionTimeout = 30;

/***** BGLOBA VARS *****/
/*****
const int generalCalibrationTimeout = 1000; // Wait after system setup is complete
const int loopTimeout = 60000; // Loop timeout, time between observations_
↳publications (in ms)
String response = ""; // Rest call response (normaly as JSON message)
int statusCode = -1; // Rest call return code (the HTTP code)

boolean existsSensor = false;

void setup() {
    // Begin serial for debug purposes
    Serial.begin(9600);

    // Setup the LDR sensor
    setupLDR();

    // Setup the LM35 sensor
    setupLM35();

    // Setup the Sentilo Client
    // and network connection
    setupSetiloClient();

    // Setup the Sentilo sensor
    // and create it if doesn't exists
    setupSentiloSensor();

    // Waiting for the next release of the observation
    delay(generalCalibrationTimeout);
}

void loop() {
    if (existsSensor) {
        // If the sensor exists,
        // we can start publishing observations

        // Get the LDR value
        int ldrValue = getLdrValue();

        // Get the LM35 value
        float lm35Value = getLM35Value();

        // Create the observation input message
        // like this: {"ldr":"234","lm35":"24.5"}
        String obsInputMsg =
            "{\\"ldr\\":\\" + String(ldrValue) +
            \\",\\"lm35\\":\\" + String(lm35Value) +
            \\"}";
        int bufLength = obsInputMsg.length() + 1;
        char obsMsgBuffer[bufLength];
        obsInputMsg.toCharArray(obsMsgBuffer, bufLength);

```

(continues on next page)

(continued from previous page)

```

        // Create the Observation object
        SentiloClient::Observation observation;
        observation.value = obsMsgBuffer;

        // Debug on Serial the observations value
        // Note that the message includes slashes (\) because we must scape
        ↪special characters as "
        Serial.print("[loop] Publishing actual sensors values as
        ↪observations: ");
        Serial.println(obsMsgBuffer);

        // Publish the observation to Sentilo Platform
        statusCode = sentiloClient.publishObservation(providerId, sensorId,
        ↪observation, apiKey, response);

        // Read response status and show an error if it is necessary
        if (statusCode != 200) {
            Serial.print("[loop] [ERROR] Status code from server after
        ↪publish the observations: ");
            Serial.println(statusCode);
            Serial.print("[loop] [ERROR] Response body from server after
        ↪publish the observations: ");
            Serial.println(response);
        } else {
            Serial.println("[loop] Sensors observations published!");
        }

        // Waiting for the next loop
        delay(loopTimeout);
    } else {
        // If the sensor does not exist and it could
        // not be created in the catalog, we must stop running
        Serial.println("[loop] [ERROR] Oops! The sensor doesn't exists, so I
        ↪can't publish data to it...");
        Serial.println("[loop] [ERROR] I'm sorry with you, but now I'm going
        ↪to halt...");
        Serial.println("[loop] [ERROR] Bye!");
        while (true) { }
    }
}

// Emulate a possible LDR initialization process, if it is necessary
void setupLDR() {
    Serial.print("[setup] Setting up the LDR brightness sensor ");
    for (int i = 0; i < ldrSetupTimeout; i++) {
        Serial.print(".");
        delay(100);
    }
    Serial.println(" done!");
    delay(50);
}

// Get the brightness value from th LDR
int getLdrValue() {
    return analogRead(LDR);
}

```

(continues on next page)

(continued from previous page)

```

// Emulate a possible LM35 initialization process, if it is necessary
void setupLM35() {
    Serial.print("[setup] Setting up the LM35 temperature sensor ");
    for (int i = 0; i < lm35SetupTimeout; i++) {
        Serial.print(".");
        delay(100);
    }
    Serial.println(" done!");
    delay(50);
}

// Get the LM 35 temperature value in Celsius degrees
float getLM35Value() {
    int val = analogRead(LM35);
    float mv = (val / 1024.0) * 5000;
    float cel = mv / 10;
    //float farh = (cel * 9) / 5 + 32;
    return cel;
}

// Setup the Sentilo Client object
// This process also configures the network connection
void setupSetiloClient() {
    // Connect via DHCP
    Serial.print("[setup] Connecting to network via DHCP ");
    sentiloClient.dhcp();
    for (int i = 0; i < networkConnectionTimeout; i++) {
        Serial.print(".");
        delay(100);
    }
    Serial.println(" done!");
    Serial.println("[setup] Connection is now established!");
}

// Setup the Sentilo Sensor (this Arduino)
// If the sensor doesn't exists in the catalog, create it
void setupSentiloSensor() {
    Serial.println("[setup] Retrieving catalog info from Sentilo and search for_
↪the sensor...");

    // Get catalog data for the provider with the supplied api key
    statusCode = sentiloClient.getCatalog(apiKey, response);

    // If the server status response is not ok, show the error
    if (statusCode !== 200) {
        Serial.print("[setup] [ERROR] Status code from server getting_
↪catalog: ");
        Serial.println(statusCode);
        Serial.print("[setup] [ERROR] Response body from server getting_
↪catalog: ");
        Serial.println(response);
    } else {
        // If we get a correct response, we must search the sensor
        if (find_text(sensorId, response) >## 0) {
            // The sensor is in the catalog
            Serial.println("[setup] The sensor is in the catalog");
            existsSensor = true;
        }
    }
}

```

(continues on next page)

(continued from previous page)

```

    } else {
        // The sensor isn't in the catalog, so we must create it
        Serial.println("[setup] The sensor isn't in the catalog, so
↪let register it now...");

        // Create the basic Sentilo Sensor Object
        SentiloClient::Sensor sensor;
        sensor.sensor = sensorId;
        sensor.type = "status";
        sensor.dataType = "TEXT";
        sensor.component = componentId;
        sensor.componentType = "generic";
        sensor.location = "sensorLat sensorLng";

        // Call the SentiloClient Register Sensor function
        statusCode = sentiloClient.registerSensor(sensor, providerId,
↪apiKey, response);

        // Read the server status response
        if (statusCode 200) {
            // If ok, the sensor has been yet created
            existsSensor = true;
        } else {
            // If nok, then we can't continue with the program
            existsSensor = false;
            Serial.print("[setup] [ERROR] Status code from server
↪getting catalog: ");

            Serial.println(statusCode);
            Serial.print("[setup] [ERROR] Response body from
↪server getting catalog: ");

            Serial.println(response);
        }
    }
}

// Auxiliary function for search text in a String
int find_text(String needle, String haystack) {
    int foundpos = -1;
    for (int i = 0; (i < haystack.length() - needle.length()); i++) {
        if (haystack.substring(i, needle.length() + i) == needle) {
            foundpos = i;
        }
    }
    return foundpos;
}

```

And finally, in the last example, we can see:

- Initialization is the same that in the other examples
- Before ending the initialization process, we search for the sensor in the catalog:
 - Into the *setupSentiloSensor()* method, the **sentiloClient.getCatalog** retrieves all the catalog data related to the provider, so we can now search for the value of our sensor, in this case, **sample-sensor-arduino-03**, and we see that it doesn't exists in the catalog (you must not create it manually!)
 - Then, create it with **sentiloClient.registerSensor**, including a Sensor object (see values below), if you

want to publish its location **don't forget** to initialize the *sensorLat* and *sensorLng* values!

- Once the sensor is created, we end the setup process and starts the loop
- If there is any error registering the sensor, the serial prints the error message and the server status code in the console
- In the loop, like in Example 2, retrieve sensors data (LDR and LM35), and publish them as new sensor observation

Next, there is an example of Sensor object message with the example values:

```
SentiloClient::Sensor sensor;  
sensor.sensor = "sample-sensor-arduino-03";  
sensor.type = "status";  
sensor.dataType = "TEXT";  
sensor.component = "sample-component";  
sensor.componentType = "generic";  
sensor.location = "41,385063 2,1734034";
```

And before invoking the Sentilo API Rest platform, the SentiloClient library transforms this object in a JSON message like this:

```
{ "sensors": [{  
    "sensor": "sample-sensor-arduino-03",  
    "description": "",  
    "type": "status",  
    "dataType": "TEXT",  
    "unit": "",  
    "component": "sample-component",  
    "componentType": "generic",  
    "componentDesc": "",  
    "location": "41,385063 2,1734034",  
    "timeZone": "CET"  
}]  
}
```

As you can see, the type is generic and the data type is text, because this is the best way to publish any data without any format problem.

On this page you will find several tutorials about how to connect to Sentilo using various existing platforms and languages.



Java Client

Java library that allows access to Sentilo Platform through its REST API Client



RaspberryPi Client

Independent platform library created with NodeJS that allows embedded architectures, such as Raspberry Pi, to communicate with Sentilo Platform through its REST API Client



Arduino Client

A simple Arduino library that allows connect with the Official Ethernet Shield to the Sentilo Platform

9.1 In which platforms has been Sentilo tested ?

The first deployment for the Barcelona City Council has been tested in the following infrastructure:

- Four virtual machines, two for the front-ends and another two for the back-end
- All of them use as operating system Ubuntu server LTS 12.04
- The real time database server(Redis) works with 16 GB of memory and 36 GB of hard disk
- The other three servers works with 4 GB of memory and 16 GB of hard disk

Another deployment configuration should work properly, always keeping in mind the expected load by the system. There is also a [virtual machine](#) ready for use that can be used for testing purposes.

9.2 I successfully published an observation, but I cannot see the data in catalog.

Check that the Catalog and Sentilo API Server are in the same timezone, for example in UTC. Make sure the sentilo-server script has the following VM option:

```
-Duser.timezone=UTC
```

Also, make sure that the Tomcat that hosts the Catalog application has the same option, for example en \$JAVA_OPTS variable.

9.3 Maps is not showing up in Catalog application

Recently Google changed its policy regarding Maps key. Please go to <https://developers.google.com/maps/documentation/javascript/get-api-key> and create one.

If you are using the last release of Sentilo(1.6) you can define the API key inside the catalog-config.properties configuration file:

```
# Google API key to use Google Maps
google.api.key=<your key>
```

9.4 I created a provider and immediately after that, an observation using the new provider's token is rejected with 401 "Invalid credential"

The providers are activated in a background job that runs every 5 minutes. Please wait a moment :-)

9.5 The command `mvn package appassembler:assemble` fails.

You have to execute the command in the directory of the component you want to install.

9.6 I think I installed Sentilo. How can I confirm all is up & running?.

You can use this script:

```
./scripts/testServerStatus.sh
```

You also might want to check [Platform Testing](#)

If you installed everything on your local machine, you can access the catalog at <http://localhost:8080/sentilo-catalog-web> and the REST API at <http://localhost:8081>

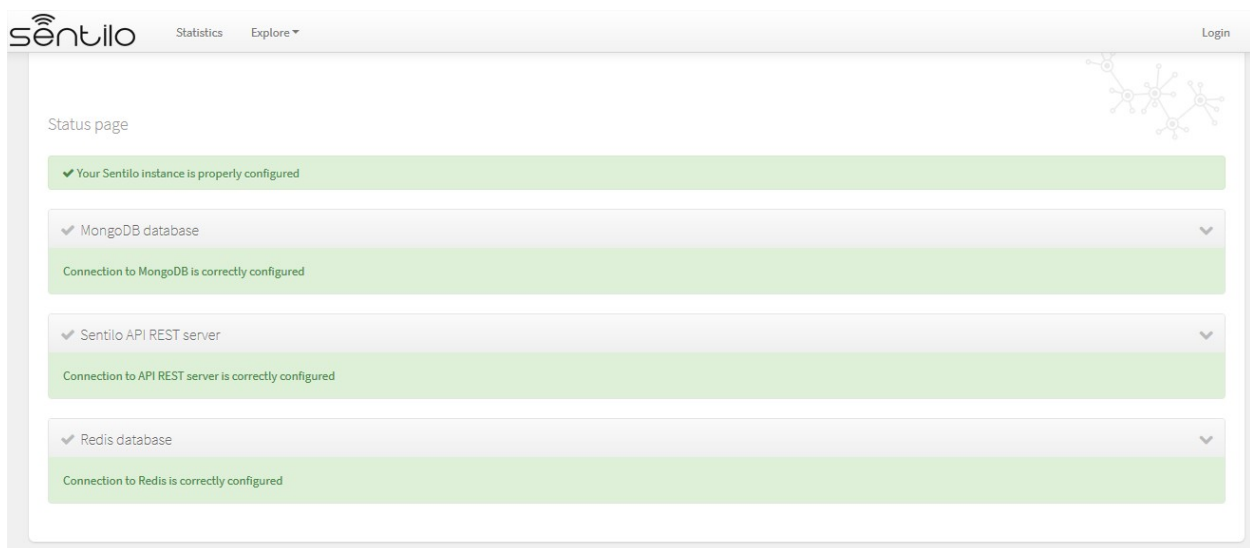
To check everything is properly configured and running, you can run the following set of tests.

10.1 Infrastructure servers test

10.1.1 Status page

To validate that all services are up and running (Redis, MongoDB and PubSub), you can access to the following catalog page:

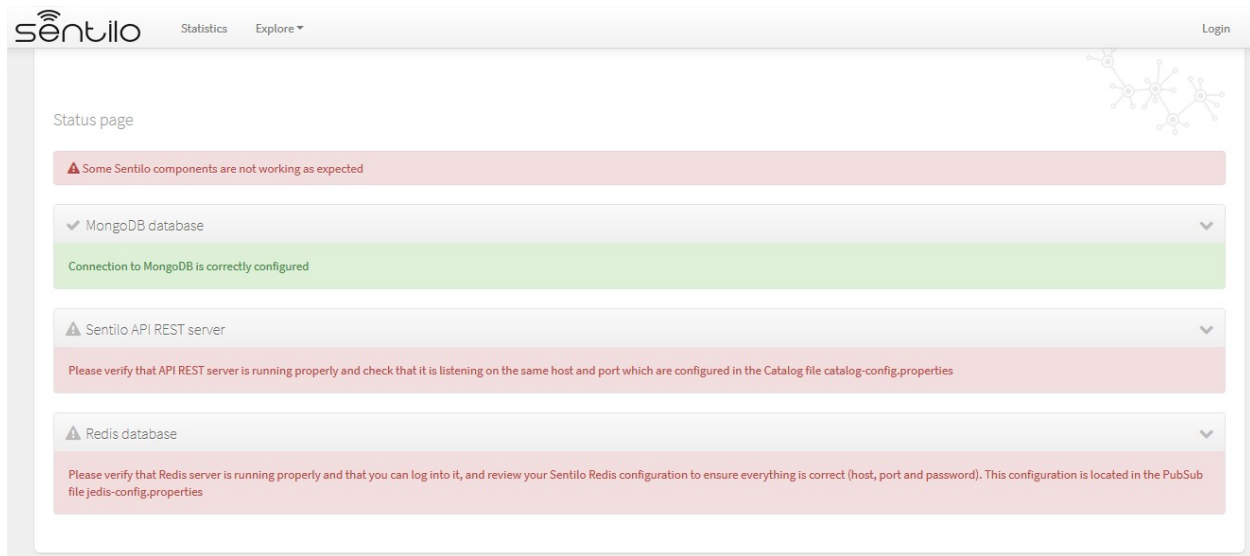
```
http://ip:port/sentilo-catalog-web/status/
```



In this screen you can check the status independently for each Sentilo main service. In each case it will be indicated, through a green status message, the correct operation of the same. In the event either it is not possible to connect to

the service or there is an error, an error message will be displayed .

Next screenshot shows to you an error connecting to the API:



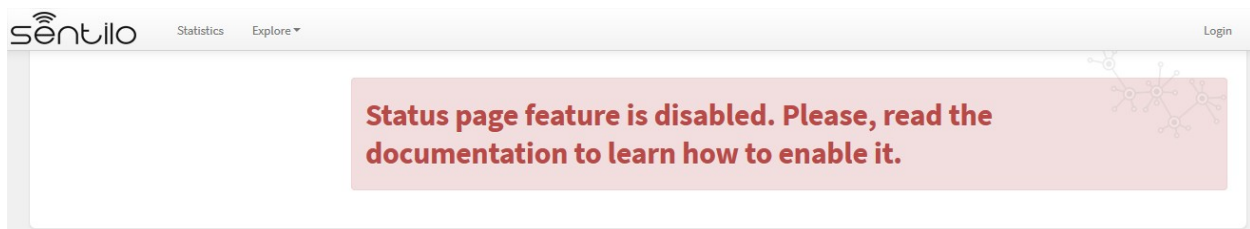
Deactivating the status page

By default, the status page is enabled in your Sentilo instance.

To disable it, you must provide a JVM Tomcat parameter:

```
-Dsentilo.state_page.enabled=false
```

Then, the status page will be inaccessible:



10.2 Postman tests

To test the API REST services individually, you can also test end-end functionality with [Postman](#), or if you prefer CLI, via [Newsman](#):

```
newman run postman-script.json -e postman-script-env.json --delay-request 2000 --
  ↳reporters cli,json --reporter-json-export outputfile.json
```

where files *postman-script.json* and *postman-script-env.json* are located in subdirectory *scripts/test* from your local copy of Sentilo.

This script provides tests all Sentilo REST API resources and can serve you also as example of the API usage.

Use a Virtual Machine

A Sentilo sample instance is available for testing purposes distributed as a Open Virtual Appliance file ([OVA](#)).

The appliance contains the **1.7.0 Sentilo release**.

Components installed:

- Sentilo Catalog (web application)
- Sentilo Platform Server (REST API)
- Sentilo Relational Agent (saves the data to mySQL)
- Sentilo Alert Agent
- Sentilo Location Updater Agent

Two different distribution files are available:

- One designed for **Virtual Box**, available [here](#). It has been tested using version **5.0.24**.
- The second one, built for **ESXI** systems, available [here](#). It has been tested using **VMPlayer 12.5.5** and **ESXI 6.0**.

Please, keep in mind some important facts:

- The virtual machine credentials are **sentilo/sentilo**.
- You should config the network type as **“Bridged Adapter”**.
- When stopping the virtual machine, it should be done in a organized way, in a Virtual Box environment you have to do this using the option **“Shutdown ACPI”**. You could also do this from the command line executing **“sudo shutdown -h now”**

After the virtual machine is started, all the sentilo services are launched automatically. The IP of the virtual machine is assigned automatically, to know which one is, enter into virtual machine and execute the **“ifconfig”** command. In some settings you might need to port forward guest ports (essentially 8080 and 8081) and access them from your host machine.

First steps:

- Review the README file located in `/home/sentilo`.

- The Catalog Console webapp will be ready to access in: http://your_ip:8080/sentilo-catalog-web/ with a access credentials: admin/1234
- The API Rest endpoint will be listening for requests in: http://your_ip:8081