istSOS²

- WORKSHOP -

Exploring the Sensor Observation Service Standard Enhanced by istSOS Special Features

Held in:

http://2014.ogrs-community.org/

http://foss4g-e.org/

Table of Contents

Workshop intro

istSOS (https://geoservice.ist.supsi.ch/projects/istsos) is an OGC SOS server implementation of the Sensor Observation Service Standard, entirely written in Python, istSOS allows managing and dispatching observations from monitoring sensors.

The workshop will give the audience all the necessary skills to deploy the istSOS server for managing and dispatching sensor data. Moreover attendees will be introduced with all the extra features that istSOS offers.

The workshop will cover the following sections:

Introduction to the standard and to istSOS	30 min. (PT30M)
Installation of the software	1 h 15 min. (PT1H15M)
Set-up of the service	
Registering new sensors	
Inserting observations	
Manipulating observations (load, edit, delete)	
Validation of observations (quality index)	
Create virtual sensors (virtual procedures)	
Dynamic aggregation	
istSOS RESTful API overview	
PAUSE	15 min. (PT15M)
Mapping sensors and data with OpenLayers 3	1 h (PT1H)
Plotting observations on the Web	
Connect Open Hardware & Sensors with istSOS	1 h (PT1H)
Setting up the istSOS web notification service	

SOS

Sensor Observation Service

Sensor Observation Service (SOS) is an Open Geospatial Consortium (OGC) approved standard (in version 1 and 2) that is part of the Sensor Web Enablement (SWE) initiative.

Sensor Web Enablement

The OGC Sensor Web Enablement (SWE) working group was established to provide a first definition of the system that is able to enable the development of the Sensor Web idea by identifying the technology, the language syntax and the architecture to be used.
In summary, the SWE aim is to define a unique environment where specialists and users can search, access, and process data observed by a multitude of heterogeneous sensor networks.
With this propose the SWE working group has defined a series of OpenGIS® standards:

1. SWE Common: XML model for sensors data
2. SensorML (Sensor model Language): XML model for processes and components
3. O&M (Observation and Measurements): XML model for the observation and measurements
4. SOS (Sensor Observation Service): interface for data access and distribution
5. SPS (Sensor Planning Service): interface for the sensor operation activation

And a series of candidate specification or discussion paper like the:

1. SAS (Sensor Alert Service): interface for dispatching alerts using the XMPP (extensible messaging and presence protocol)
2. WNS (Web Notification Service): interface for the notification of information with different protocols

Sensor Observation Service

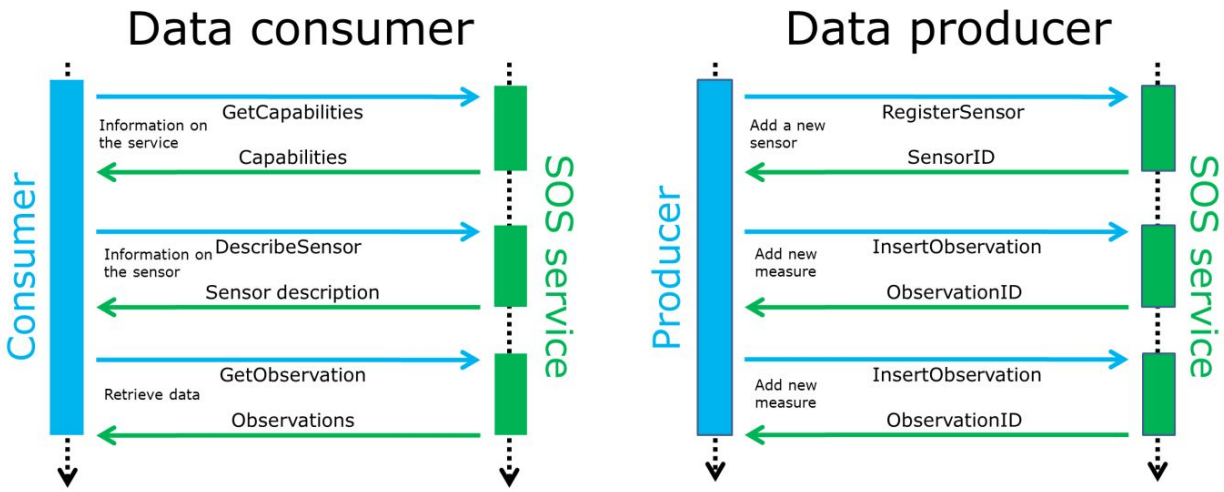
As part of the SWE, the Sensor Observation Service standard defines the interface to interact with sensor observations: from sensors exploration to measures retrieval and data management throughout transactional operations. It worth noting that this description refers to SOS in version 1; the version 2 of the standard introduces some minor changes.

Like most of the OGC standards, the SOS service is based on the exchange of standard messages (requests and responses) between the service and the consumer by using the HTTP protocol. The requests are sent to the service through an HTTP POST (in this case an XML file formatted according to the specification is submitted) or an HTTP GET method (in this case a KVP, key-value-pairs, is submitted) specifying the request type and the relative permitted parameters. The service responses are always XML file compliant with the specifications. According to the OGC specification a SOS version 1.0 service must implement at minimum the three mandatory requests of the SOS core profile, while other operations of the transactional profile and of the enhanced profile are optional (see next Table).

SOS request	Profile	Man-datory	Short description
GetCapabilities	Core	Yes	Allow to describe the service providing information on administrator, offered capabilities, observed property and features, etc..
DescribeSensor	Core	Yes	It provides a potentially detailed description of a given registered component, system or process in SensorML format
GetObservation	Core	Yes	It provides observations based on the setting of filters that includes timing, processes, phenomena, feature of interest, and other parameters in O&M model
RegisterSensor	Transactional	No	It provides capability to automatically register a new sensor to the existing service
InsertObservation	Transactional	No	It provides capability to dynamically insert new observation(s) related to a registered sensor
GetFeatureOfInterest	Enhanced	No	It provides requested feature of interest in GML format
GetResult	Enhanced	No	It provides a light way to request observation without provide full request every time
GetObservationByID	Enhanced	No	It provides a quick access to observation by identification number
GetFeatureOfInterestTime	Enhanced	No	It provides the time interval when a given feature of interest has been observed

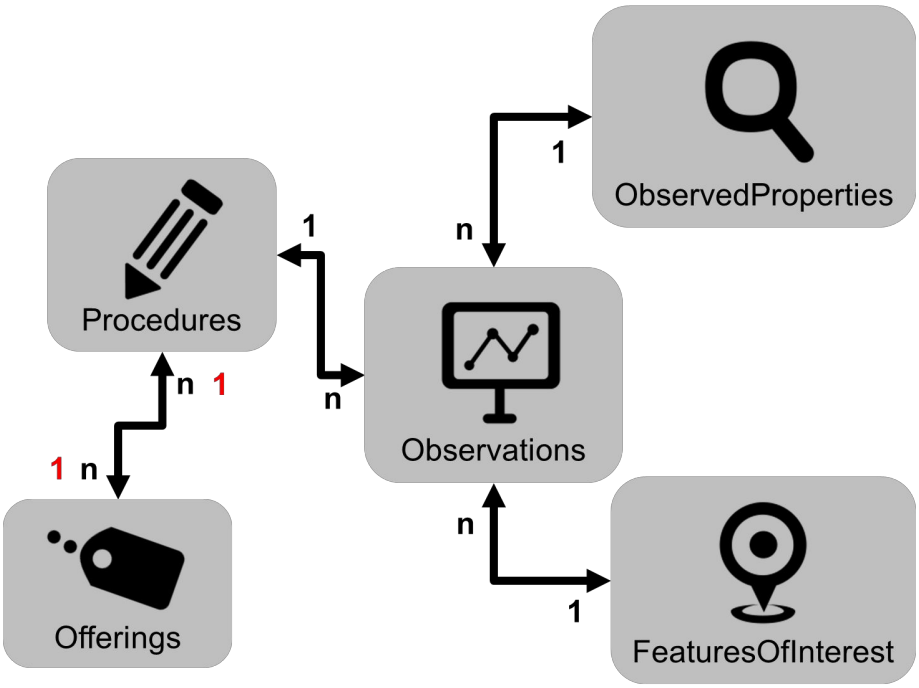
DescribeFeatureType	Enhanced	No	It provides the schema used to represent the features of interest
DescribeObservationType	Enhanced	No	It provides the schema used to represent the Observations
DescribeResultModel	Enhanced	No	It provides the schema used to represent the result object within the sml:observation

Two typical SOS UML sequence diagrams respectively from a data consumer and data producer perspective are presented in Figure:



The SOS is based on five key objects:

1. Observations: they are the center of the standard and represent the values measured at given time instants (e.g.: value: 0.2, datetime: 08-11-2015 12:12) and represented according to the O&M standard data model.
2. Procedure: indicates who provide the observations, this is generally the sensor but it may also be a generic process that leads to some observations (e.g.: procedure: TREVANO) and is represented as SensorML standard data model.
3. ObservedProperties: they represent the phenomena that are observed (e.g.: phenomenon: air-temperature) and is represented with a URI (uniform resource identifier) composed by colon separated text according to the om:observedProperty of the O&M standard.
4. FeaturesOfInterest: it is the feature that relates to the observations, so for an in-place instrument is the sensor location, while for remote device it the target location (e.g.: location: Trevano, coordinates: 718345,99224,389, reference system: CH1903/LV03) represented according to the om:featureOfInterest element of the O&M standard.
5. Offering: it is a collection of sensor used to conveniently group them up (e.g.: offering: weather-sensor-SUPSI) and is represented as sos:ObservationOffering element of the SOS standard.



SOS elements: in red relationships changes introduced with SOS version 2.0

istSOS

Istituto Scienze della Terra
Sensor Observation Service

Scuola universitaria professionale della Svizzera italiana
Dipartimento ambiente costruzioni e design
Istituto scienze della Terra

SUPSI

Introduction

istSOS (Istituto Scienze della Terra Sensor Observation Service) is the implementation of the Sensor Observation Service standard from Open Geospatial Consortium by the Institute of Earth Sciences (IST, Istituto Scienze della Terra). The development of istSOS started in 2009 in order to provide a simple implementation of the SOS standard for the management, provision and integration of hydro-meteorological data collected in Canton Ticino (Switzerland).

The Institute of Earth Sciences (IST-DACD-SUPSI)

The Institute of Earth Sciences (IST) is active in disciplines aimed to protect the environment and resources of the territory. It is composed of specialists in the management of surface- and groundwater, in modeling related to natural hazards, the study of regional geology, in environmental databases and geographic information systems (GIS).
The Institute collaborates with national and international universities, public administrations and private companies on the realization of applied research projects related to the territory, financed by national funds from the public and private sector and sometimes from the European Union. The partnership with the cantonal administration regards several issues: the maintenance of the monitoring network of water resources, flood forecasts, the development of web services and the design and management of environment databases.
The main tasks consist in conducting applied research and development, providing services and contributing to teach in basic and post formation. The IST carries out its activities in disciplines, such as geology and geotechnics, hydrology, hydrogeology and geomatics, all intended to manage and protect the environment and its resources. Among the topics covered are therefore the management of surface water and groundwater, monitoring, modeling related to hazards, environmental databases and geographic information systems (GIS).

Technologies

istSOS is a FOSS4G software, entirely written in Python and is based on:

- PostgreSQL / PostGIS
- Apache / mod_wsgi (mod_python for version <= 1.0)
- GDAL (for versions < 2.0)

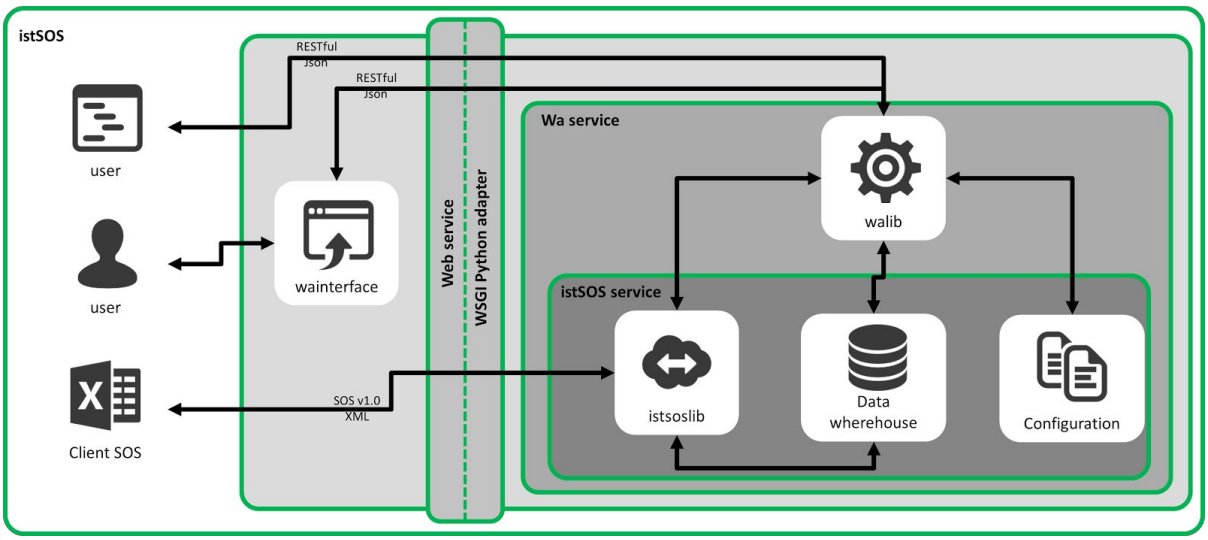
And it takes advantage of other python modules:

- isodate
- python-gdal (for version <=2.0)
- psycopg2
- pytz

Software architecture

istSOS evolved in time from being a SOS service provider to complete data management system.
Initially it was an implementation supporting the standard as defined by the OGC through the istSOSlib, the core of the system. Thanks to the continue interaction with hydrologists and data users, We soon realized that the standard didn't account for a number of requirements, like a missing supporting for irregular time series. So, We started developing extending features and enriched the **istSOSlib**. Afterward, We understood that without a comfortable Graphical User Interface (GUI) the time used to manage the system and the data (e.g.: update metadata, add new sensors, verify, correct and validate data) were too overwhelming. This lead to the idea of developing a Web Administration interface (**wainterface**) based on a RESTful service: **walib**.

As a result today istSOS is like a peach, were the **wainterface** is the peel, **walib** is the pulp and **istsoslib** is the kernel. These three components interact each others and offer different interfaces for different uses types: humans, processes or standard clients.

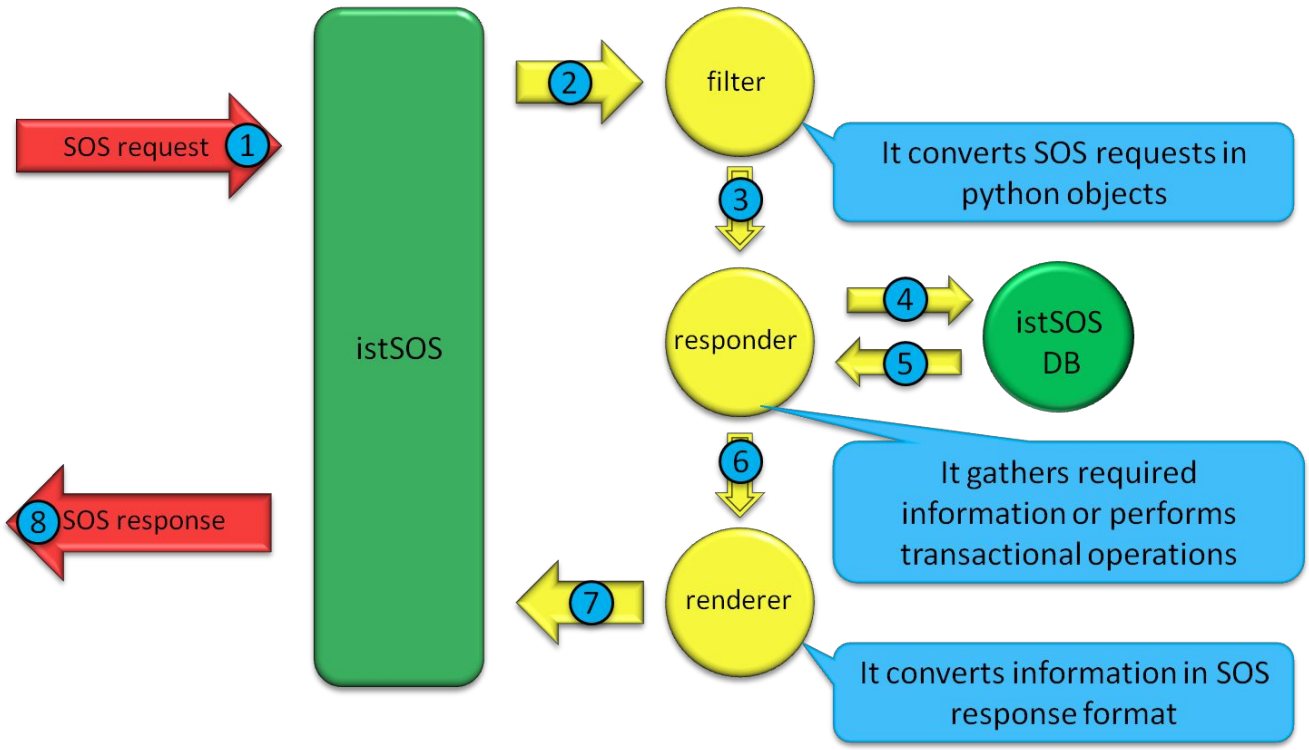


Basic software structure

istSOS has been developed with a **factory approach** using three modules:

- filters: provide an interface for converting http requests (GET or POST) submitted according the SOS standard in python objects that contains the submitted parameters and values.
- responders: resolve the specific request, interacting with the istSOS database and gathering the required informations.
- renderers: responsible for converting the informations stored in the responders into SOS response format as defined by SOS standard.

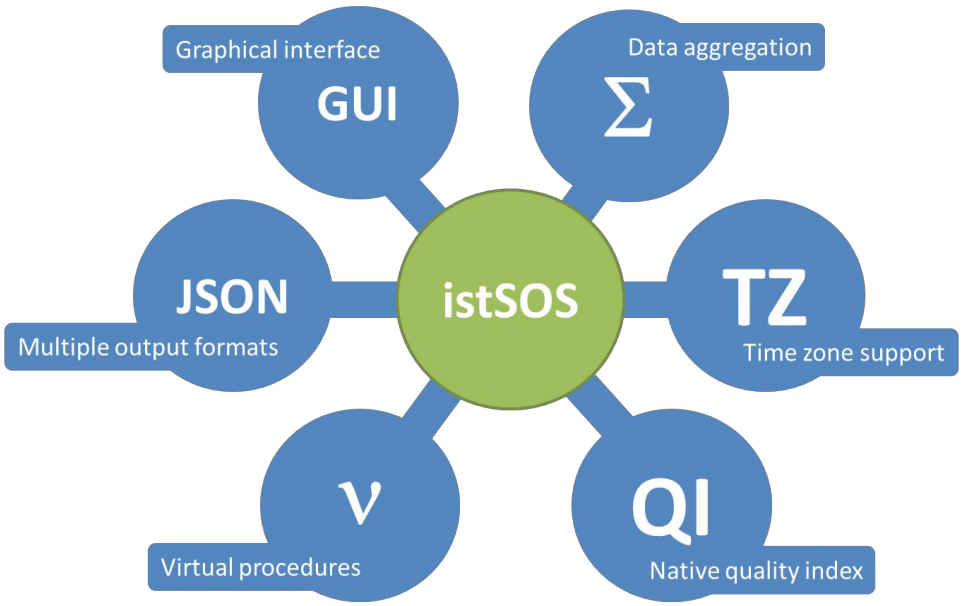
These istSOS service handle the different SOS requests by instantiating sequentially a factory_filter, a factory_responder and a factory_renderer which are capable to call the specific classes for elaborating the correct response.
As illustrated below, at first the factory_filter instantiate the appropriate filter class (function of the request type), then the factory_responder automatically instantiate the relative responder class and finally the correct renderer is called by the factory_renderer to produce an SOS response.



A configuration file store important setting parameters like service metadata (e.g.: name, version, owner), database connection parameters, dictionaries definition, handled requests, etc.

istSOS extending features

Apart from standard features like filter capabilities of the observations by means of time periods, observed properties, geospatial relationships and sensor names istSOS implements several extending features that, according to the developers and decades experienced collaborators in sensor network management and data management were found particularly helpful.

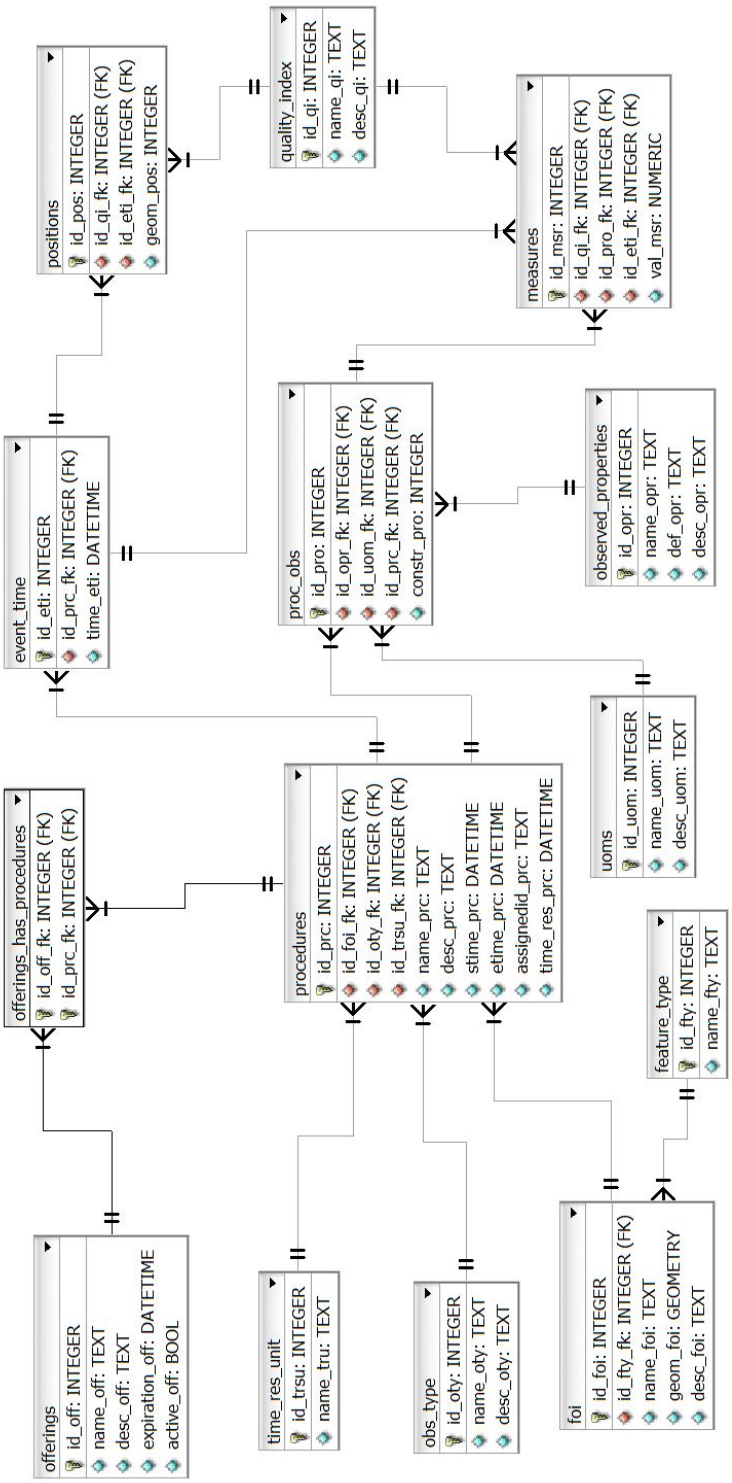


Some of the extending capabilities are:

- Handle of irregular time series
- On-the-fly aggregation of observed measures with no-data management.
- Support for different output formats like application/json, text/csv, and text/xml:subtype=sensorML.
- Possibility of inserting many observations with a single insertObservation requests.
- Supporting of a override parameter that allows to overwrite already registered observations with new ones.
- Capability to filter observations based on partial observed property names (LIKE filtering support).
- Native support for data quality index associated with each observation.
- Setting of maximum period for data retrievement requests to avoid server overloads.
- Availability of a virtual procedure mechanism, that expose new sensors and observed properties as on-the-fly elaboration of regular observation.
- On the fly data aggregation.
- Real time data validation and quality index

IstSOS database schema

each istSOS instance is based on a PostGIS schema as represented below.



Project links

Mailing list: https://groups.google.com/forum/#!forum/istsos
Web: https://istsos.org
Source code: http://sourceforge.net/projects/istsos

Licensing

Acknowledgements

The authors thank all the partners of the Verbano Lake working group for flood risk reduction, and specially the Canton Ticino, (Ufficio dei corsi d'acqua), for supporting this project. Special thanks to people contributing to the software development and Google Summer of Code and OSGeo for sponsoring students works on istSOS.

istSOS tutorial

Step by step guide for istSOS usage: from installation to Web mapping.

This part of the tutorial provides instruction on installing istSOS on a Linux Operating System based on Debian distribution and Windows. Nevertheless, even though extensive testing has been conducted in this environment only, istSOS is developed in Python which is known for its cross platform support and existing installation are recorded in Windows ® systems and OS X Apple's operating system should be supported too. Interested users may look at the istSOS Website or ask support on the mailing list.

Installation on Linux from Debian package

The easiest way to install istSOS on a Debian distribution is to use the istSOS deb packages.

⚒ Download the debian file from the repository

```
wget  http://downloads.sourceforge.net/project/istsos/python-istsos_2.2.1-1_all.deb
```

⚒ Install the debian file

```
sudo dpkg -i python-istsos_2.2.1-1_all.deb;sudo apt-get -f -y install
```

This command will install all the required dependencies, with the exception of PostgreSQL and PostGIS as the database is not mandatory. In fact it could reside on other servers.

To install and configure the database, please go to the "**Database configuration**" paragraph.

ⓘ	If everything has gone well, you should see the administration page at this address: http://localhost/istsos/admin/

Installation on Linux from source

The dependencies need to be installed manually or using apt-get command. please refer also to specific software installation procedures.

⚒ Install Apache2 and mod_wsgi

```
sudo apt-get install apache2 libapache2-mod-wsgi
```

⚒ Install psycopg2

```
sudo apt-get install python-psycopg2
```

⚒ Download istSOS and unpack it

Go to https://sourceforge.net/projects/istsos/files/latest/download?source=files, the download of the latest version of istSOS will start in 5 seconds, save the file in the Downloads folder in your home directory, then unpack executing these commands:

```
cd ~/Downloads
sudo tar -zxvf istSOS-2.1.1.tar.gz -C /usr/local/
```

⚒ Set executing permission and owner for the services and logs folders

```
sudo chmod 755 -R /usr/local/istsos
sudo chown -R www-data:www-data /usr/local/istsos/services
sudo chown -R www-data:www-data /usr/local/istsos/logs
```

⚒ Configure Apache and WSGI

Open /etc/apache2/sites-enabled/000-default:

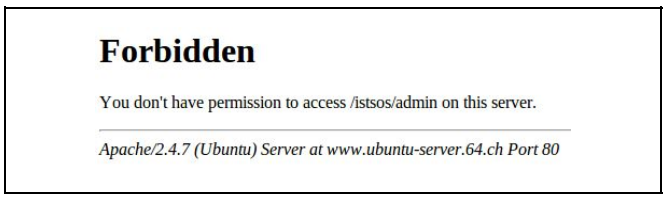
```
sudo gedit /etc/apache2/sites-enabled/000-default.conf
```

And add the following lines just before the last VirtualHost tag:

```
<VirtualHost *:80>
        # The ServerName directive sets the request scheme, hostname and port that
        # the server uses to identify itself. This is used when creating
        # redirection URLs. In the context of virtual hosts, the ServerName
        # specifies what hostname must appear in the request's Host: header to
        # match this virtual host. For the default virtual host (this file) this
        # value is not decisive as it is used as a last resort host regardless.
        # However, you must set it for any further virtual host explicitly.
        #ServerName www.example.com

        ServerAdmin webmaster@localhost
        DocumentRoot /var/www/html

        # Available loglevels: trace8, ..., trace1, debug, info, notice, warn,
        # error, crit, alert, emerg.
        # It is also possible to configure the loglevel for particular
        # modules, e.g.
        #LogLevel info ssl:warn
```

```
        ErrorLog ${APACHE_LOG_DIR}/error.log
        CustomLog ${APACHE_LOG_DIR}/access.log combined

        # For most configuration files from conf-available/, which are
        # enabled or disabled at a global level, it is possible to
        # include a line for only one particular virtual host. For example the
        # following line enables the CGI configuration for this host only
        # after it has been globally disabled with "a2disconf".
        #Include conf-available/serve-cgi-bin.conf

        WSGIScriptAlias /istsos /usr/local/istsos/application.py
        Alias /istsos/admin /usr/local/istsos/interface/admin
        Alias /istsos/modules /usr/local/istsos/interface/modules
</VirtualHost>
```

If you are using Apache/2.4.6 or above (like in Ubuntu 13.10 or above) you could meet the "403 Forbidden" message.



In that case additional setup shall be made. In the "**000-default.conf**" file add also this configurations:

```
[...]
        WSGIScriptAlias /istsos /usr/local/istsos/application.py
        Alias /istsos/admin /usr/local/istsos/interface/admin
        Alias /istsos/modules /usr/local/istsos/interface/modules
        <LocationMatch /istsos>
            Options +Indexes +FollowSymLinks +MultiViews
            AllowOverride all
            Require all granted
        </LocationMatch>
</VirtualHost>
```

Restart the Apache web server

```
sudo service apache2 restart
```

If everything has gone well, you should see the administration page at this address:
http://localhost/istsos/admin/

Database configuration

⚒ Install PostgreSQL and PostGIS

```
sudo apt-get install postgresql postgresql-9.3-postgis-2.1 pgadmin3
```

⚒ Change the postgreSQL password

```
sudo -u postgres psql
alter user postgres password 'postgres';
```

ⓘ	Ctrl-D to exit from psql console

⚒ Create your PostGIS database

For Postgresql 9.1 and later versions:

```
sudo -u postgres createdb -E UTF8 istsos
sudo -u postgres psql -d istsos -c 'CREATE EXTENSION postgis'
```

⚠	For older versions of postgresql:
	`sudo -u postgres createdb -E UTF8 istsos`
	`sudo -u postgres psql -d istsos \`
	`  -f /usr/share/postgresql/9.1/contrib/postgis-1.5/postgis.sql`
	`sudo -u postgres psql -d istsos \`
	`  -f /usr/share/postgresql/9.1/contrib/postgis-1.5/spatial_ref_sys.sql`

Installation on windows 7 and 8

⚒ install python

download python 2.7 from https://www.python.org/downloads/%20 and install it. Check if the python path is in the Environment variables:

```
Computer > properties > advanced system settings > Environment Variables.
Check if the python27 exists in the variable PAth, if not add ‘;C:\Python27\’
```

⚒ install postgreSQL with PostGIS

get PostgreSQL from
http://www.enterprisedb.com/products-services-training/pgdownload#windows and install it.

```
During the installation configure the password to be ‘postgres’.
Install postGIS 2.1 using the application Stack Builder at the end of the installation of
postgreSQL. Check the option to create a new database and call it ‘istsos’
```

⚒ install apache 2.2

download Apache 2.2 (http://mirror.switch.ch/mirror/apache/dist//httpd/binaries/win32/) and install it using the .msi file. If an error signals a missing dll, download and install Microsoft Visual C++, then try again to install Apache. If the error persists, download the missing dll from http://www.dll-files.com/dllindex/index-m.shtml and copy into the /windows/system32 folder and reboot the system.

⚒ install mod_wsgi

get the apache module mod_wsgi (http://www.lfd.uci.edu/~gohlke/pythonlibs/#mod_wsgi) for apache 2.2 and python 2.7 and copy it in the folder / modules of the Apache installation folder.

⚒ install extra modules

Download this extra modules and install them:
psycopg2: http://www.stickpeople.com/projects/python/win-psycopg/
python-dateutil: http://www.lfd.uci.edu/~gohlke/pythonlibs/#python-dateutil
six: http://www.lfd.uci.edu/~gohlke/pythonlibs/#six

```
During the installation configure the password to be ‘postgres’.
Install postGIS 2.1 using the application Stack Builder at the end of the installation of
postgreSQL. Check the option to create and call it ‘istsos’
```

⚒ install istSOS

Download istSOS (http://sourceforge.net/projects/istsos/files/) and unpack it in e.g. C: so that will be a folder C:\istsos

⚒ Configure apache2.2

Go to the folder where Apache is installed, modify the permissions of conf/httpd.conf and conf/extra/httpd-vhosts.conf that they are writable from Everyone.
Open conf/httpd.conf with a text editor and add this line:

```
LoadModule wsgi_module modules/mod_wsgi.so #where the other LoadModule line are
Remove the # from the line Include conf/extra/httpd-vhosts.conf
```

Open conf/extra/httpd-vhosts.conf, delete the two examples of <VirtualHost> and paste the following code. Modify the paths so they correspond to the Apache and istSOS folders.

```
<VirtualHost *:80>
        ServerAdmin webmaster@localhost
        DocumentRoot "C:/Apache2/htdocs"
        <Directory />
                Options FollowSymLinks
                AllowOverride None
        </Directory>

        <Directory C:/Apache2/htdocs/>
                Options Indexes FollowSymLinks MultiViews
                AllowOverride None
                Order allow,deny
                allow from all
        </Directory>

        ScriptAlias /cgi-bin/ "c:/Apache2/cgi-bin/"
        <Directory "c:/Apache2/cgi-bin">
                AllowOverride None
                Options +ExecCGI -MultiViews +SymLinksIfOwnerMatch
                Order allow,deny
                Allow from all
        </Directory>

        ErrorLog "c:/Apache2/logs/error.log"
        LogLevel warn
        CustomLog "c:/Apache2/logs/access.log" combined
        Alias /doc/ "c:/Apache2/manual/"

        <Directory "c:/Apache2/manual/">
                Options Indexes MultiViews FollowSymLinks
                AllowOverride None
                Order deny,allow
                Deny from all
                Allow from 127.0.0.1
        </Directory>

        WSGIScriptAlias /istsos "c:/istsos/application.py"
        <Location "/istsos">
                Options Indexes MultiViews FollowSymLinks
                AllowOverride None
                Order deny,allow
                Deny from all
                Allow from 127.0.0.1
        </Location>
        Alias /istsos/admin "c:/istsos/interface/admin"
        Alias /istsos/modules "c:/istsos/interface/modules"
</VirtualHost>
```

⚒ Restart apache 2.2

Restart apache 2.2 using the icon in the system try or:

```
control panel > system and security > administrative tools > services
click on Apache 2.2 and then on restart.
```

Check the installation

Now istSOS is up and running. Open a web browser and go to http://localhost/istsos/admin. You should see the istSOS Web Admin page. If an error occurs, take a look at the Apache error log with this command to understand what's going wrong:

In Linux:

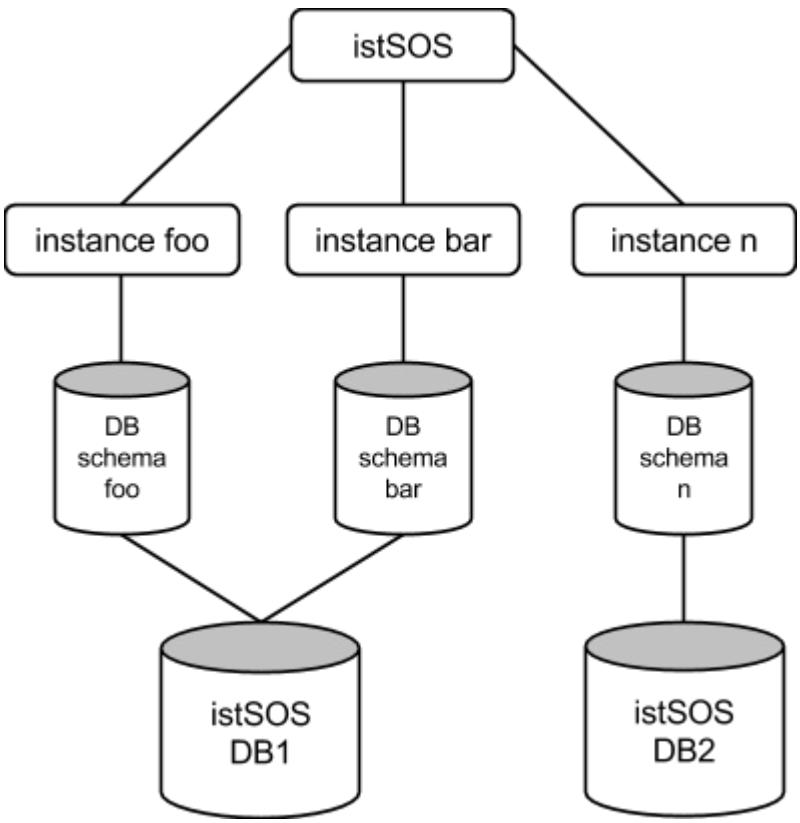
```
tail -f /var/log/apache2/error.log
```

In Windows:

```
Open the file <Apache2.2 folder>\logs\error.log
```

Setting up an istSOS service instance

With istSOS you can organize your sensor data in different instances. Every instance has its own database schema independent from other instances. You can even deploy other databases over your network according to your needs.



The first steps into istSOS setup is to configure the *default configuration* options. These options will then be automatically used for your convenience by every new istSOS instance created.

⚒ Now open the Web Admin interface:

http://localhost/istsos/admin

⚒ Configure your default database connection

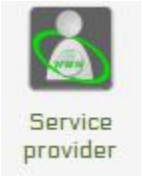
From the toolbar buttons menu press the database button and fill in the database configuration options:

user: postgres
password: postgres
host: localhost
port: 5432
DB name: istsos

Press the “test connection” button to check if parameters are correct.

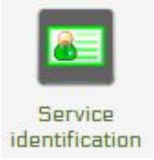
⚒ Configure the default Service provider information

Use your institutional information.



⚒ Configure the default Service identification information

Fill up with custom metadata that describe the service usage.



⚒ Configure your service coordinates system

Default EPSG code: this will be the native Coordinate Reference Systems of your geometries in the database.

Permitted EPSG: here you can configure which other Coordinate Reference Systems your system will support and reproject if requested.

⚒ Configure your getobservation request options

Set maximum time interval per GetObservation requests (zero for no limits) and aggregation no-data value.

⚒ Configure your service proxy address

The Proxy URL field is the base URL seen beyond a reverse proxy.

Creating a new service instance

Now that you have configured istSOS, it's time to create a new service instance.

⚒ Create a new service

Choose the name of your instance. In this workshop use the name **demo**. Here you could also change the default EPSG and database configuration for this istSOS instance if you prefer to not use the defaults ones.

⚒ Press "next".

As you pressed the "next" button, the server status panel is displayed.
If something's gone wrong during the creation you will see here an alert.

⚒ Check the new service exists

Now the istSOS **"demo"** instance is up and running.
Let's try to execute a getCapabilities request.
http://localhost/istsos/demo?request=getCapabilities&service=SOS

ⓘ	The istSOS "demo" instance has inherited all the configuration options from the *default configuration*. If, for any reason you decide to modify them, the changes will affect only this instance.

Registering new sensors

From the “services” drop down button choose the “demo” instance.

Some advices:

⚠	Once a procedure is created the outputs (observed properties) cannot be changed.

⚠	Before registering new sensors it’s advised to initialize missing observed properties and unit of measures.

⚠	**Procedures with multiple observed properties** Is it possible to add new procedures observing more than one properties, but in istSOS this means that for each observation instant we have a list all the observed properties values. e.g.: if a procedure that observes temperature (T), humidity (H) and rain (R) is created the time series will always have a list of 3 values [T,H,R] for each instant. If you try to insert an observation with two only values it will raise an exception.

⚒ Add procedures GRABOW with multiple observed properties

Let’s use the following settings for the station in GRABOW

Name: GRABOW
Description: Enorasis Meteo Station in Grabow, Poland.
Keywords: weather, meteorological,IUNG-PIB

System type: insitu-fixed-point
Sensor type: Meteo Station

FOI name: GRABOW
EPSG: 4326
Coordinates: x: 22.67 y: 51.25 z: 177

Outputs:

Observed property:
urn:ogc:def:parameter:x-istsos:1.0:meteo:air:humidity:relative
Unit of measure: %
Description: -

Quality index check: Between / from 0 to 100

Observed property:
urn:ogc:def:parameter:x-istsos:1.0:meteo:air:rainfall
Unit of measure: mm
Description: -
quality index check: Between / from 0 to +500

Observed property:
urn:ogc:def:parameter:x-istsos:1.0:meteo:air:temperature
Unit of measure: °C
Description: conversion from resistance to temperature
Quality index check: Between / from -40 to +60

Observed property:
urn:ogc:def:parameter:x-istsos:1.0:meteo:air:wind:velocity
Unit of measure: m/s
Description: -
Quality index check: Between / from 0 to 200

Observed property:
urn:ogc:def:parameter:x-istsos:1.0:meteo:solar:radiation
Unit of measure: W/m2
Description: -
Quality index check: from 0 to +500

Optional parameters:
fill at your own need and willing

Register the new sensor (procedure) pressing the "**submit**" button.

⚒ Add other procedures

In order to speed up the *"boring"* process of inserting all the other procedures we have prepared a script. Open a terminal and run:

```
cd ~/Desktop/Tutorial
python fill/execute.py
```

Verify the inserted procedures using the administration interface

Check your procedures by accessing the "**Procedures**" panel.
You will see a table showing an abstract of all the inserted procedures. By clicking on the name you will be able to enter the details metadata that you configured during the procedure registration.

	The "**Procedures**" panel not only allows for procedures and metadata exploration but also allows details modification. The only exception are the outputs parameters.

Verify the inserted procedures using the Sensor Observation Service requests

Let's try to execute a getCapabilities request to verify if procedures are now available.
We can use the "Requests" test page where some samples are already present.

http://localhost/istsos/modules/requests

Choose the the demo service and then choose "GET > GetCapabilities" option and then modify the section parameter to contain just the "contents" option the request to be like this:

http://localhost/istsos/demo?
request=getCapabilities&
section=contents&
service=SOS

Let's execute a describeSensor request to verify that the procedure description is available:

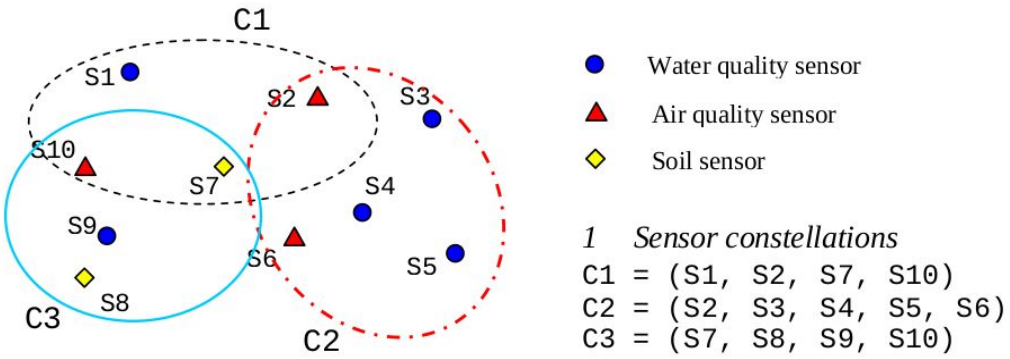
http://localhost/istsos/demo?
request=DescribeSensor&
procedure=**T_LUGANO&**
outputFormat=text/xml;subtype="sensorML/1.0.1"&
service=SOS&
version=1.0.0

repeat for procedures: P_LUGANO, LOCARNO, BELLINZONA, GRABOW, RH_GNOSCA

	Procedures are stored in the SOS with a uniqueID which is urn:ogc:def:procedure:x-istsos:1.0:XXXX for procedure named XXXX. istSOS, as we can see later for other parameters also, is not strict and allows to specify just the procedure name in the request.

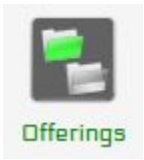
Observation Offerings

In the Sensor Observation Service 1.0.0 the concept of an Observation Offering is equivalent to that of a sensor constellation. An Observation Offering is analogous to a “layer” in Web Map Service because each offering is typically a non-overlapping group of related observations.



Create a new offering

Press the “**new**” button and fill the form as following:



Name: workshop
Description: demo dataset for FOSS4G meeting
Expiration (optional): 2020-01-01T00:00:00+09:00
Validity: Enabled

Associate procedures with offering

Activate the tab panel pressing “**Offering-procedure memberships**”.
In the dropdown list select the newly created offering “workshop”.
On the left side you will see all the procedure that will be assigned to that offering. On the right there are all the procedures not assigned to that offering.
Use drag and drop functionality to move procedures from right to left.

Verify that procedures are associated with offering as desired

Then check the getCapabilities request to see what happened.

http://localhost/istsos/demo?
request=getCapabilities&
section=contents&
service=SOS&
version=1.0.0

The “**temporary**” offering is system wide offering that is used to associate every registered procedure. Every new procedure is automatically assigned to this offering.

Insert observations

For this part of the tutorial you should use ASCII files with sensor data formatted according to text/csv, subtype=istSOS. This format is a CSV represented by a header as the first line containing the URI names of the observed properties, the following lines contains the data.

Example: PROCEDURENAME_YYYYMMDDhhmmssfff.dat

```
urn:ogc:def:parameter:x-istsos:1.0:time:iso8601,urn:ogc:def:parameter:x-istsos:1.0:meteo:air:temperature
2015-01-01T00:10:00.000000+0100,0.446000
2015-01-01T00:20:00.000000+0100,0.862000
2015-01-01T00:30:00.000000+0100,0.932000
2015-01-01T00:40:00.000000+0100,0.384000
```

Pay attention to the file name: there is a timestamp (YYYYMMDDhhmmssfff in GMT+0:00). This parameter is used as the *endPosition* in the sampling time of a procedure. This is particularly important when the procedure is an irregular time series.

Think about tipping bucket rain gauge, when there isn't rain no data are sent. But updating the endPosition we will be able to know that the sensor is working and that there is no rain, instead of thinking that the sensor is not transmitting or that it is broken.

⚒ Uploading CSV files

In the data directory of this workshop there is folder named "**dataset**". There are some examples of CSV datafiles in the "text/csv, subtype=istSOS" format:

- BELLINZONA_20150603125000.dat
- GRABOW_201505272100000.dat
- LOCARNO_201506031200000.dat
- P_LUGANO_20150603142000000.dat
- RH_GNOSCA_20150603142000000.dat
- T_LUGANO_20150603142000000.dat

⚒ Loading CSV data

Open a Shell and execute the followings commands:

If installed from source

```
cd /usr/local/istsos
```

If installed from debian package

```
cd /usr/share/istsos/
```

Then launch the import script:

```
python scripts/csv2istsos.py -p BELLINZONA LOCARNO P_LUGANO T_LUGANO GRABOW RH_GNOSCA \
  -u http://localhost/istsos -s demo \
  -w ~/Desktop/Tutorial/dataset
```

The "**csv2istsos.py**" file, is a python script that makes use of the WA-REST features of istSOS to insert observations.

```
python scripts/csv2istsos.py --help

usage: csv2istsos.py [-h] [-v] [-t] -p procedures [procedures ...]
                     [-q quality index] [-u url] -s service
                     -w working directory [-e file extension]
                     [-usr user name] [-pwd password]

Import data from a csv file.

optional arguments:
  -h, --help            Show this help message and exit
  -v, --verbose         Activate verbose debug
  -t, --test            Use to test the command, deactivating the insert
                        observation operations.
  -p procedures [procedures ...]
                        List of procedures to be aggregated.
  -q quality index      The quality index to set for all the measures of
                        the CSV file, if not set into the CSV.
                        (default: 100).
  -u url                IstSOS Server address IP (or domain name) used for
                        all request. (default: http://localhost:80/istsos).
  -s service            The name of the service instance.
  -w working directory  Working directory where the csv files are located.
  -e file extension     Extension of the CSV file. (default: .dat)
  -usr user name
  -pwd password
```

Loading data with OGC-SOS InsertObservation request

Even if we have used the *csv2istsos* script to facilitate the data loading, users may also use the SOS *insertObservation* request directly. For example, a valid request for loading a single observation to the service is:

```
<?xml version="1.0" encoding="UTF-8"?>
<sos:InsertObservation xmlns:gml="http://www.opengis.net/gml"
xmlns:om="http://www.opengis.net/om/1.0" xmlns:sos="http://www.opengis.net/sos/1.0"
xmlns:swe="http://www.opengis.net/swe" xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" service="SOS" version="1.0.0">
<sos:AssignedSensorId>f10b70b2561111e5a35e0800278295cb</sos:AssignedSensorId>
<sos:ForceInsert>true</sos:ForceInsert>
<om:Observation>
    <om:procedure xlink:href="urn:ogc:def:procedure:x-istsos:1.0:LOCARNO"/>
    <om:samplingTime>
        <gml:TimePeriod>
            <gml:beginPosition>2015-06-03T14:10:00+02</gml:beginPosition>
            <gml:endPosition>2015-06-03T14:50:00+02</gml:endPosition>
        </gml:TimePeriod>
    </om:samplingTime>
    <om:observedProperty>
        <swe:CompositePhenomenon dimension="5">
            <swe:component xlink:href="urn:ogc:def:parameter:x-istsos:1.0:time:iso8601"/>
            <swe:component xlink:href="urn:ogc:def:parameter:x-istsos:1.0:meteo:air:rainfall"/>
            <swe:component
xlink:href="urn:ogc:def:parameter:x-istsos:1.0:meteo:air:rainfall:qualityIndex"/>
            <swe:component xlink:href="urn:ogc:def:parameter:x-istsos:1.0:meteo:air:temperature"/>
            <swe:component
xlink:href="urn:ogc:def:parameter:x-istsos:1.0:meteo:air:temperature:qualityIndex"/>
        </swe:CompositePhenomenon>
    </om:observedProperty>
    <om:featureOfInterest xlink:href="urn:ogc:def:feature:x-istsos:1.0:Point:LOCARNO"/>
    <om:result>
        <swe:DataArray>
            <swe:elementCount>
                <swe:value>5</swe:value>
            </swe:elementCount>
            <swe:elementType name="SimpleDataArray">
                <swe:DataRecord definition="urn:ogc:def:dataType:x-istsos:1.0:timeSeries">
                    <swe:field name="Time">
                        <swe:Time definition="urn:ogc:def:parameter:x-istsos:1.0:time:iso8601"/>
                    </swe:field>
                    <swe:field name="air-rainfall">
                        <swe:Quantity
definition="urn:ogc:def:parameter:x-istsos:1.0:meteo:air:rainfall">
                            <swe:uom code="mm"/>
                        </swe:Quantity>
                    </swe:field>
                    <swe:field name="air-rainfall:qualityIndex">
                        <swe:Quantity
definition="urn:ogc:def:parameter:x-istsos:1.0:meteo:air:rainfall:qualityIndex">
                            <swe:uom code="-"/>
                        </swe:Quantity>
                    </swe:field>
                    <swe:field name="air-temperature">
                        <swe:Quantity
definition="urn:ogc:def:parameter:x-istsos:1.0:meteo:air:temperature">
                            <swe:uom code="°C"/>
                        </swe:Quantity>
                    </swe:field>
                    <swe:field name="air-temperature:qualityIndex">
                        <swe:Quantity
definition="urn:ogc:def:parameter:x-istsos:1.0:meteo:air:temperature:qualityIndex">
                            <swe:uom code="-"/>
                        </swe:Quantity>
                    </swe:field>
```

```
                </swe:DataRecord>
            </swe:elementType>
            <swe:encoding>
                <swe:TextBlock blockSeparator="@" decimalSeparator="." tokenSeparator=","/>
            </swe:encoding>
<swe:values>2015-06-03T14:10:00+02,0,200,20.4,200@2015-06-03T14:20:00+02,0.1,200,19.5,200@2015-06-03
T14:30:00+02,0.1,200,19.1,200@2015-06-03T14:40:00+02,0,200,19.5,200@2015-06-03T14:50:00+02,0,200,20.
6,200</swe:values>
        </swe:DataArray>
    </om:result>
</om:Observation>
</sos:InsertObservation>
```

Let’s insert observations using the XML format:

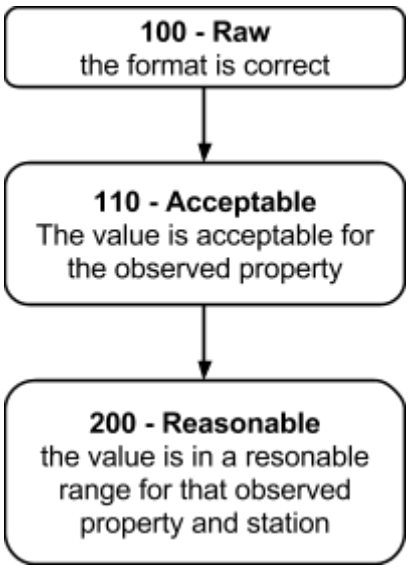
1. Open the requests test page: http://localhost/istsos/modules/requests/
2. Select the “demo” service instance
3. Choose the “POST” option
4. Paste into the field the InsertObservation xml

⚠	Pay attention to the **AssignedSensorId** parameter: this according to the standard is returned by the system only when the sensor is registered. To access it, you can use administration interface, looking at the procedure metadata details.

5. Press “Send”

Data validation with quality index

istSOS is bundled with an automatic data validation. In the next paragraphs we will see 3 levels of validation.



The data quality index configuration panel

Here you can change the value and the meaning of istSOS quality indexes

Code	Name	Description
-100	aggregation no data	no values are present for this aggregation interval
0	outboud	gross error
100	raw	the format is correct
110	acceptable	the value is acceptable for the observed property
200	reasonable	the value is in a resonable range for that observed property and station
300	timely coherent	the value is coherent with time-series
400	spatilly coherent	the value is coherent with close by observations
500	manually adjusted	the value has been manually corrected
600	correct	the value has not been modified and is correct

Raw data quality index
For every new inserted observed property the raw data quality index is assigned (by default QI 100 Raw data). This quality index suggests that the observation data type is correct, which means that istSOS checks if the measure inserted is in a numeric type.

Correct quality index
In the **observed properties** panel**,** for each observed property, you can define specific constraint based on logical operators (greater than, Lower than, between and value list). This is the place where you can set general quality index check for each Observed Property. For instance a percentage (%) observed property can use a constraint of type “Between”, because the values can be between 0% and 100%.

Statistical quality index
The statistical QI is more granular. This is set when you create a new procedure and it will be specific only to the new procedure created. For instance in the case of temperature measurements, we know that in our region temperature never goes under -20°C and over 40°C, so we can put as correct QI the "**between**" constraint. But a new sensor deployed on top of a mountain the limits are different and the QI constraint can be more specific for this station (between -20° and +20°C).

⚒ Testing quality index check

Lets try to load some data that will better explain the "quality index check" functionality:

Sample data present under qi folder in your dataset:

```
urn:ogc:def:parameter:x-istsos:1.0:time:iso8601,urn:ogc:def:parameter:x-istsos:1.0:meteo:air:temperature
2015-06-03T15:30:00+01:00,T_LUGANO,150
2015-06-03T15:40:00+01:00,T_LUGANO,62
2015-06-03T15:50:00+01:00,T_LUGANO,25
```

Open a terminal and…

If installed from source

```
cd /usr/local/istsos
```

If installed from debian package

```
cd /usr/share/istsos/
```

Then import data with errors..

```
python scripts/csv2istsos.py -p T_LUGANO \
  -u http://localhost/istsos -s demo \
  -w ~/Desktop/Tutorial/qi
```

Now check what happens executing a getObservation request:

http://localhost/istsos/demo?
service=SOS&
version=1.0.0&
request=GetObservation&
offering=temporary&
procedure=T_LUGANO&
eventTime=2015-06-03T15:20:00+01:00/2015-06-03T15:50:00+01:00&
observedProperty=temperature&
responseFormat=text/plain&
qualityIndex=True

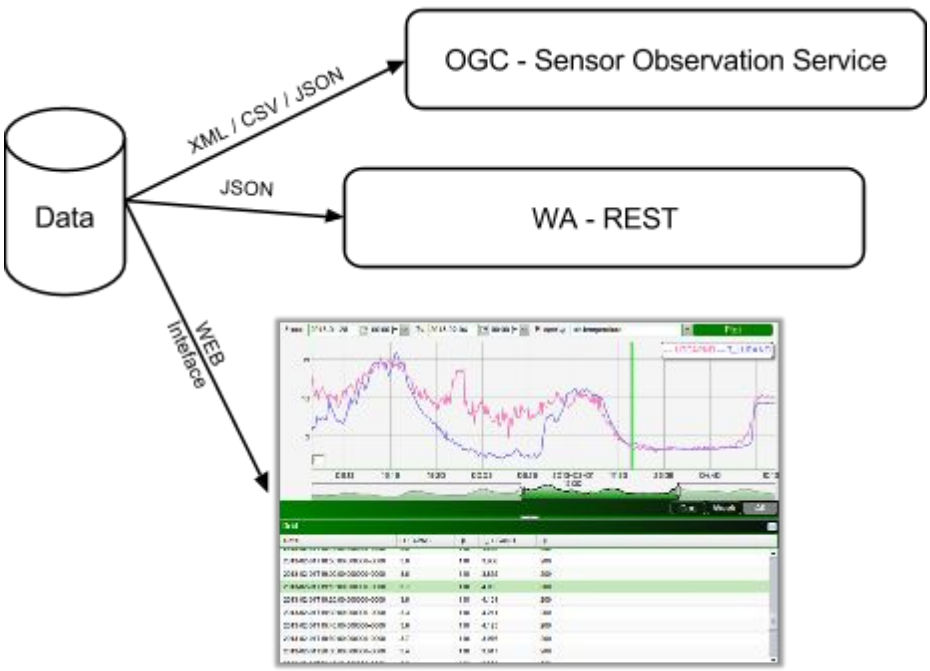
Looking at the result you can note the different quality indexes associated with the measures:

```
urn:ogc:def:parameter:x-istsos:1.0:time:iso8601,urn:ogc:def:procedure,urn:ogc:def:parameter:x-istsos:1.0:meteo:air:tempera
ture,urn:ogc:def:parameter:x-istsos:1.0:meteo:air:temperature:qualityIndex
2015-06-03T15:30:00+01:00,T_LUGANO,150.000000,100
2015-06-03T15:40:00+01:00,T_LUGANO,62.000000,110
2015-06-03T15:50:00+01:00,T_LUGANO,25.000000,200
```

- The first measure (150) didn't pass the "acceptable" quality check and didn't get a 110 index
- The second (62) pass the "acceptable" quality check but didn't pass the "reasonable" quality check and didn't get a 200 index
- The third measure (25) passed both the "acceptable" and "reasonable" quality check so it get a 200 index

Accessing data

Accessing the data is possible using the SOS Standard requests, WA REST requests or directly using the Data Viewer or Data Editor.



Let's try to request data using the different interfaces…..

Getting observations in the SOS 1.0.0 way

Quick view on the GetObservation request using the GET method:

Parameter	Description	Definition	Multiplicity and Use
request	getObservation is designed to request detailed sensor metadata	getObservation	One (mandatory)
service	Service type identifier	SOS	One (mandatory)
version	Specification version for operation	1.0.0	One (mandatory)
srsName	Defines the spatial reference system that should be used for any geometries that are returned in the response.	This must be one of the advertised values in the offering specified in gml:srsName elements.	One (Optional)
offering	Specifies the offering URI advertised in the GetCapabilities document. All of the following parameters are dependent on the selected offering.	This must match the gml:name of the offering or be constructed as a URL with a fragment identifier resolving to the offering gml:id.	One (mandatory)
eventTime	Specifies the time period(s) for which observations are requested. This allows a client to request observations from a specific instant, multiple instances or periods of time in the past, present and future. The supported range is listed in the selected offering capabilities	ISO 8601 Examples: Time instant: 2015-06-11T17:30:00+0200 Time period 2015-06-11T14:30:00+0200/ 2015-06-11T17:30:00+0200	Zero or many (Optional)
procedure	The procedure parameter specifies the sensor system(s) for which observations are requested.It defines a filter for the procedure property of the observations.	comma separated valid sensors from the GetCapabilities	Zero or many (Optional)
observedProperty			One or many (mandatory)
featureOfInterest	Specifies the feature for which observations are requested. This can either be represented by a reference to a feature ID advertised in the capabilities document or can be a spatial constraint.		Zero or many (Optional)

responseFormat	Specifies the desired resultFormat MIME content type for transport of the results (e.g. TML, O&M native format, or MPEG stream out-of-band). The supported output formats are listed in the selected offering capabilities. Desired output format of the getObservation operation. This can be a MimeType or QName for example.	text/xml;subtype="sensorML/1.0.0"	One (mandatory)

Get data with GetObservation (procedure-time-property filters)
Requesting rainfall observations from sensor P_LUGANO between 2015-01-01T00:00:00+01 and 2015-02-4T17:00:00+01:

http://localhost/istsos/demo?
service=SOS&
request=GetObservation&
offering=temporary&
procedure=**urn:ogc:def:procedure:x-istsos:1.0:P_LUGANO&**
eventTime=**2015-06-01T00:00:00+0200/2015-06-03T00:00:00+0200&**
observedProperty=**urn:ogc:def:parameter:x-istsos:1.0:meteo:air:rainfall&**
responseFormat=text/xml;subtype="om/1.0.0"&
service=SOS&
version=1.0.0

Get data with GetObservation (time-property filters)
Requesting rainfall observations from all the stations between 2014-06-01T00:00:00+02 and 2014-06-03T00:00:00+02:

http://localhost/istsos/demo?
service=SOS&
request=GetObservation&
offering=temporary&
eventTime=2015-06-01T00:00:00+0200/2015-06-03T00:00:00+0200&
observedProperty=**urn:ogc:def:parameter:x-istsos:1.0:meteo:air:rainfall&**
responseFormat=text/xml;subtype="sensorML/1.0.0"&
service=SOS&
version=1.0.0

Get data with GetObservation (time-property-area filters)
Requesting the last meteo observations applying a spatial intersection with a BBOX:

http://localhost/istsos/demo?service=SOS&
request=GetObservation&
offering=temporary&
observedProperty=meteo&
responseFormat=text/xml;subtype=”sensorML/1.0.1”&
service=SOS&
version=1.0.0&
featureOfInterest=<ogc:BBOX><ogc:PropertyName>the_geom</ogc:PropertyName><gml:Box srsName='EPSG:4326'><gml:coordinates>18,45 30,55</gml:coordinates></gml:Box></ogc:BBOX>

Get data with GetObservation (time-property-distance filters)
Requesting the last meteo observations applying a distance filter:

http://localhost/istsos/demo?
service=SOS&
request=GetObservation&
offering=temporary&
observedProperty=temperature&
responseFormat=text/xml;subtype=”sensorML/1.0.1”&
service=SOS&
version=1.0.0&
featureOfInterest=<ogc:DWithin><ogc:PropertyName>SHAPE</ogc:PropertyName><gml:Point srsName="EPSG:4326"><gml:coordinates decimal="." cs="," ts=" ">8.961,46.027</gml:coordinates></gml:Point><ogc:Distance>10</ogc:Distance></ogc:DWithin>

Now try to change the distance from 10 to 100.

Get data with GetObservation (time-property filters and result values)
Requesting rainfall observations from all the stations between 2014-06-01T00:00:00+02 and 2014-06-03T00:00:00+02 and where the rain value is greater than 0:

http://localhost/istsos/demo?service=SOS&request=GetObservation&offering=temporary&eventTime=2015-06-01T00:00:00+0200/2015-06-03T00:00:00+0200&observedProperty=urn:ogc:def:parameter:x-istsos:1.0:meteo:air:rainfall&responseFormat=text/plain&service=SOS&version=1.0.0&result=%22%3Cogc:PropertyIsGreaterThan%3E%3Cogc:PropertyName%3Erainfall%3C/ogc:PropertyName%3E%3Cogc:Literal%3E0%3C/ogc:Literal%3E%3C/ogc:PropertyIsGreaterThan%3E%22

Getting observations using the istSOS extending features

In this part of the tutorial we will explore the extending features of istSOS, developed for making user life easier and to fulfill data experts requirements.

⚒ GetObservation with simple names

According to the standard observedProperties and procedures are accessible using a Unique Resource Identifier (URI). This is those used when the observed property and the procedure was created.
In this tutorial we used for example:

- urn:ogc:def:parameter:x-istsos:1.0:meteo:air:rainfall
- urn:ogc:def:procedure:x-istsos:1.0:P_LUGANO

istSOS is not strict and allows to specify in a GetObservation request just the or procedure name and/or observed property in the request: as a result for the desired procedures all the observed properties with that words will be selected (LIKE '%XXX%' SQL query).

In a suggested hierarchical usage of URIs, this allows to quickly access to all the subdomain properties, so for example using ***observedProperty=urn:ogc:def:parameter:x-istsos:1.0:meteo*** in a getObservation request in istSOS will return all the available observations which measure meteo parameters (rainfall, windspeed, humidity, etc..).

Let's try:

http://localhost/istsos/demo?
service=SOS&
request=GetObservation&
offering=temporary&
procedure=**P_LUGANO&**
eventTime=2015-06-01T00:00:00+0200/2015-06-03T00:00:00+0200&
observedProperty=**rainfall&**
responseFormat=text/xml;subtype="sensorML/1.0.1"&
service=SOS&
version=1.0.0

Requesting all the **rainfall observations** between 2015-06-01T00:00:00+02 and 2015-06-03T00:00:00+02:

http://localhost/istsos/demo?
service=SOS&
request=GetObservation&
offering=temporary&
eventTime=2015-06-01T00:00:00+0200/2015-06-03T00:00:00+0200&
observedProperty=rainfall&
responseFormat=text/xml;subtype="sensorML/1.0.0"&
service=SOS&
version=1.0.0

⚒ GetObservation with specific time zone

istSOS support time zones. Whenever in getObservation request the eventTime is specified with a time zone (e.g.: +0700) the response will be returned with the same time zone.

http://localhost/istsos/demo?
service=SOS&
request=GetObservation&
offering=temporary&
procedure=**P_LUGANO&**
eventTime=2015-06-01T00:00:00+0500/2015-06-03T00:00:00+0500&
observedProperty=**rainfall&**
responseFormat=text/xml;subtype=”sensorML/1.0.1”&
service=SOS&
version=1.0.0

⚒ GetObservation in CSV or JSON

in addition to the mandatory text/xml;subtype=”sensorML/1.0.0” istSOS support also application/json and text/csv (for simplification also text or json)

Data in CSV:
http://localhost/istsos/demo?
service=SOS&
request=GetObservation&
offering=temporary&
eventTime=2015-06-01T00:00:00+0200/2015-06-03T00:00:00+0200&
observedProperty=rainfall&
responseFormat=text/plain&
service=SOS&
version=1.0.0

Data in JSON:
http://localhost/istsos/demo?
service=SOS&
request=GetObservation&
offering=temporary&
eventTime=2015-06-01T00:00:00+0200/2015-06-03T00:00:00+0200&
observedProperty=rainfall&
responseFormat=application/json&
service=SOS&
version=1.0.0

⚒ GetObservation with data aggregation on the fly

When executing a getObservation request istSOS offer an extra feature. Adding vendor specific parameters **aggregateInterval**, **aggregateFunction**, **aggregatenodata** and **aggregatenodataqi** you can request data already aggregated by istSOS.

- **aggregateInterval**: ISO 8601 Durations (P1DT = 1 Day, PT12H = 12 hours)
- **aggregateFunction:** AVG, SUM, MAX, MIN

The next two parameters are optional and the default values can be configured using the Web Admin interface:

- **aggregatenodata**: numeric value to use in the case of irregular time series.
- **aggregatenodataqi:** the quality index to be assigned to no data.

For example we can Request maximal daily temperature observation:

http://localhost/istsos/demo?
service=SOS&
request=GetObservation&
offering=temporary&
procedure=T_LUGANO&
eventTime=2015-05-04T00:00:00+01/2015-05-14T00:00:00+01&
observedProperty=temperature&
aggregateInterval=PT24H&
aggregateFunction=MAX&
responseFormat=text/plain&
service=SOS&
version=1.0.0

⚒ **GetObservation in CSV with qualityIndex**
in addition to the mandatory text/xml;subtype=”sensorML/1.0.0” istSOS support also application/json and text/csv (for simplification also text or json)

Data in CSV with qualityIndex:
http://localhost/istsos/demo?
service=SOS&
request=GetObservation&
offering=temporary&
eventTime=2015-06-01T00:00:00+0200/2015-06-03T00:00:00+0200&
observedProperty=rainfall&
responseFormat=text/plain&
service=SOS&
version=1.0.0&
qualityIndex=True

Data in CSV with qualityFilter:

Note that when applying a qualityFilter istSOS returns only the observations that has all the observedProperties values with associated qualityIndex satisfying the criteria.

e.g.:
Suppose you have a procedure observing rainfall, temperature and humidity.

Your single observation will have the triplet of values and quality index:
[time1, rain value, rain qi, temperature val, emperature qi, humidity val, humidity qi]
[time2, rain value, rain qi, temperature val, emperature qi, humidity val, humidity qi]
[time3, rain value, rain qi, temperature val, emperature qi, humidity val, humidity qi]
[time4, rain value, rain qi, temperature val, emperature qi, humidity val, humidity qi]
….

Operator	istSOS behaviour
>	only records where all qi are > of the filter value are returned
>=	only records where all qi are >= of the filter value are returned
<	only records where all qi are < of the filter value are returned
<=	only records where all qi are <= of the filter value are returned
=	all the records that has at least one qi equal to the filter value

http://localhost/istsos/demo?
service=SOS&
request=GetObservation&
offering=temporary&
eventTime=2015-06-01T00:00:00+0200/2015-06-03T00:00:00+0200&
observedProperty=rainfall&
responseFormat=text/plain&
service=SOS&
version=1.0.0&
qualityIndex=True&
qualityfilter=>110

Getting observations using the WA REST

Composing a WA REST request is all about building the correct path url.

⚒ GetObservation with WA REST

To get the observations execute this request:

http://localhost/istsos/wa/istsos/services/demo/operations/getobservation/offerings/temporary/procedures/T_LUGANO/observedproperties/temperature/eventtime/2015-05-21T00:00:00+02:00/2015-05-28T00:00:00+02:00

ⓘ	Executing a service request, you will receive a list of istSOS service instances: http://localhost/istsos/wa/istsos/services Executing a procedures get list operation, you will receive a list of procedures belonging to a specific service: http://localhost/istsos/wa/istsos/services/demo/procedures/operations/getlist

Getting observations with the Data Viewer

The istSOS web administration pages interact with the service making use of WA REST.
The Data Viewer panel is implemented as an example of data visualization.

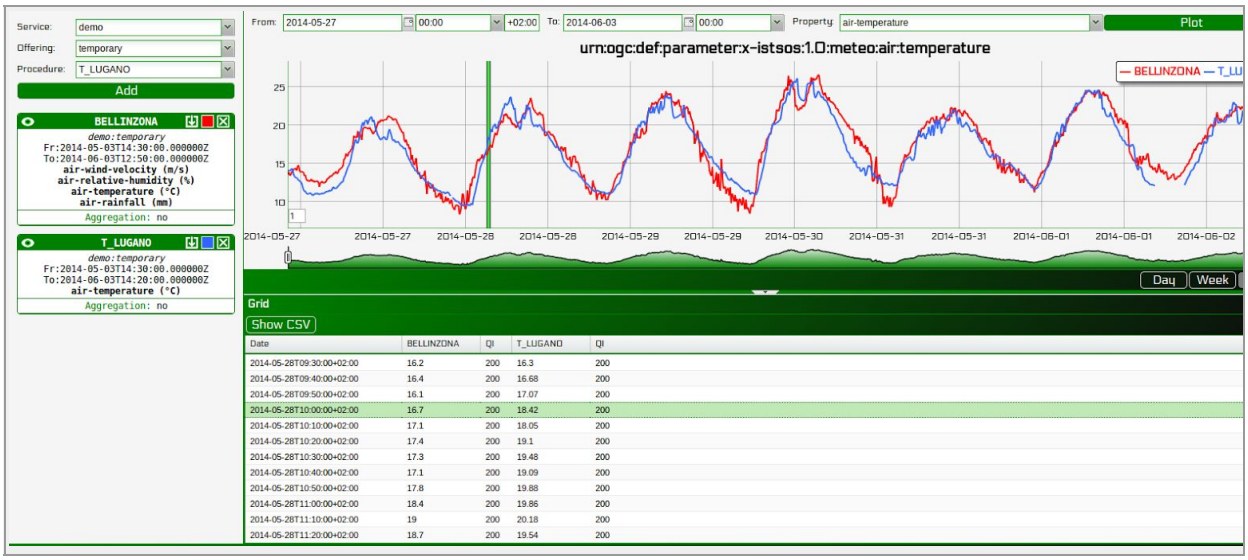
View observation

To open your Web Viewer follow this link:

http://localhost/istsos/admin/viewer.html

ⓘ	Up to now the viewer permit to display data of a single observationProperties only, you can select and display multiple procedures but with the same observed property.

Go ahead and take some confidence with the Data Viewer..



Editing observations with the Data Editor

The procedure **T_LUGANO** has some problems..
From 2015-06-02T02:40:00 to 2015-06-02T07:20:00 there are no data values:

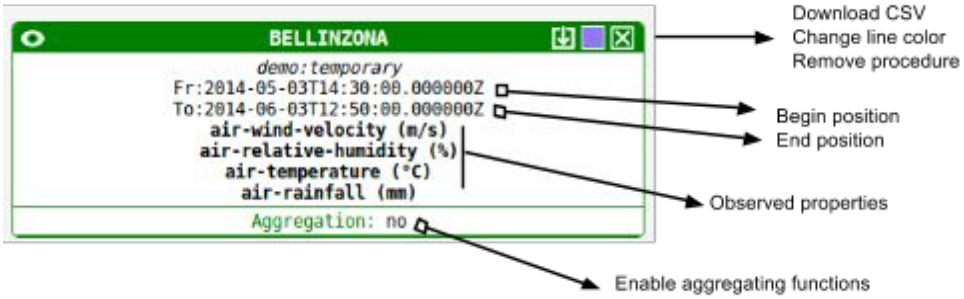
```
urn:ogc:def:parameter:x-istsos:1.0:time:iso8601,urn:ogc:def:parameter:x-istsos:1.0:meteo:air:tempera
ture
[...]
2015-06-02T02:30:00.000000+0200,12.150000
2015-06-02T02:40:00.000000+0200,-999.9
2015-06-02T02:50:00.000000+0200,-999.9
2015-06-02T03:00:00.000000+0200,-999.9
[...]
```

We can correct them using the Data Editor!

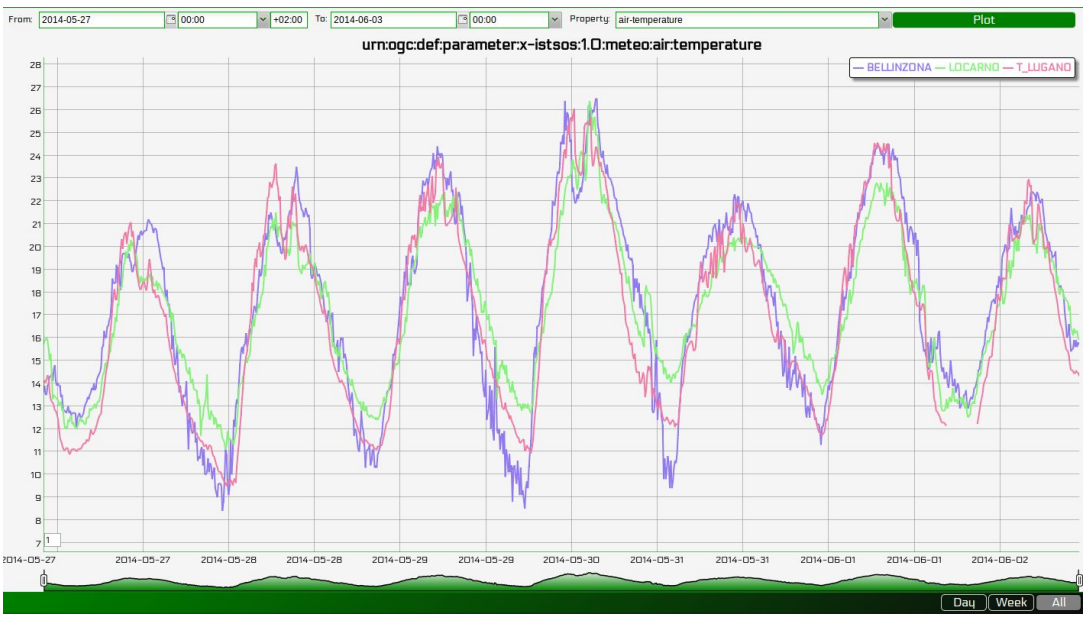
Load the data

From the Web Admin:

- Go to Data Viewer
- Press the Data Editor button
- Like in the Data Viewer sequentially choose
  - the service **demo**,
  - the offering **temporary**
  - and then "*Add*" **BELLINZONA**, **LOCARNO** and **T_LUGANO**



- On the right panel choose the Property: **air-temperature**
- Press "**Plot**", the last week of measurements is loaded and displayed



Editing with the "Calculator"

On the left panel there is the "**Editor**" tab:

- Select **T_LUGANO** from the combo list
- The press “**Start editing**”, the grid is now displayed
- At the bottom-right corner of the chart there are 3 buttons “Day”, “Week” and “All”
    - Click on “**Day**”, the chart is zoomed to contain only one day of data
    - Drag the timeline bar on the right where you will see that T_LUGANO has no data

- Click on the chart to select the last observation before the “**nodata**” hole, a green line is displayed and in the **Editing Grid** the corresponding row is selected.
- Now go to the **Editing Grid** panel
    - Click the first row where data are **NaN**,
    - Scroll to the last **NaN** record and holding the **SHIFT** Key click on it
    - The press the “**Calculator**” button

WIth the **Calculator** we are able to correct an interval of data in a single action. It is possible to set a numeric value or also use a function using data from the other loaded procedures.

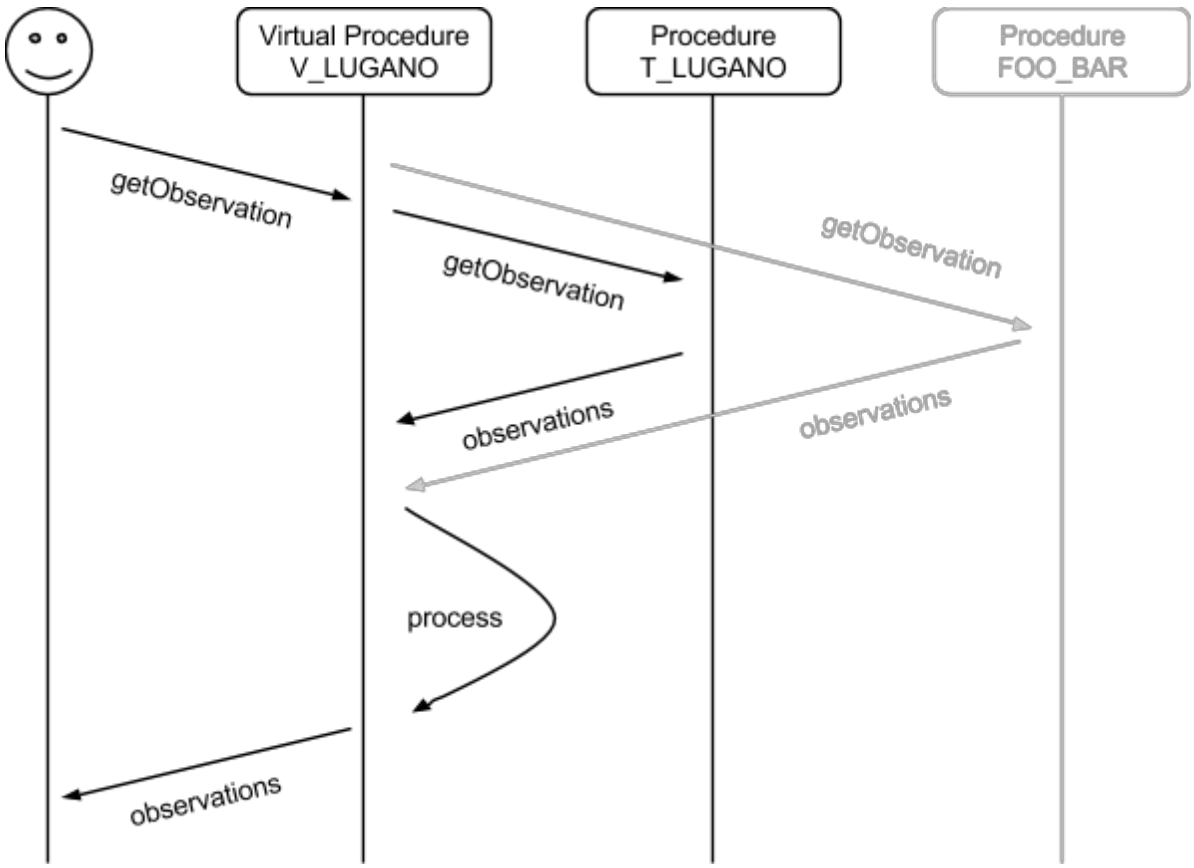
Let's build a function that make the average of the data from BELLINZONA and LOCARNO and then removes to units:

((BELLINZONA+LOCARNO)/2)-2

Select the quality index. In this case we can choose a QI 500 (manually adjusted).

Creating Virtual Procedures

With Virtual Procedures you are able to use other procedures data (real or virtual) and manipulate data to get a different result.



When working with VP it's easy to make some mistakes while coding.. So while testing your VP look at the apache error log to read about errors:

```
tail -f /var/log/apache2/error.log
```

If something goes wrong or you do some modification on your code, you should also restart the apache server.

```
sudo service apache2 restart
```

When you have filled up istSOS automatically you have create a couple of virtual procedures:

V_LUGANO: it gets data from a sensor measuring temperature in Celsius degree (T_LUGANO) and transforms it to Fahrenheit degree, here is the code:

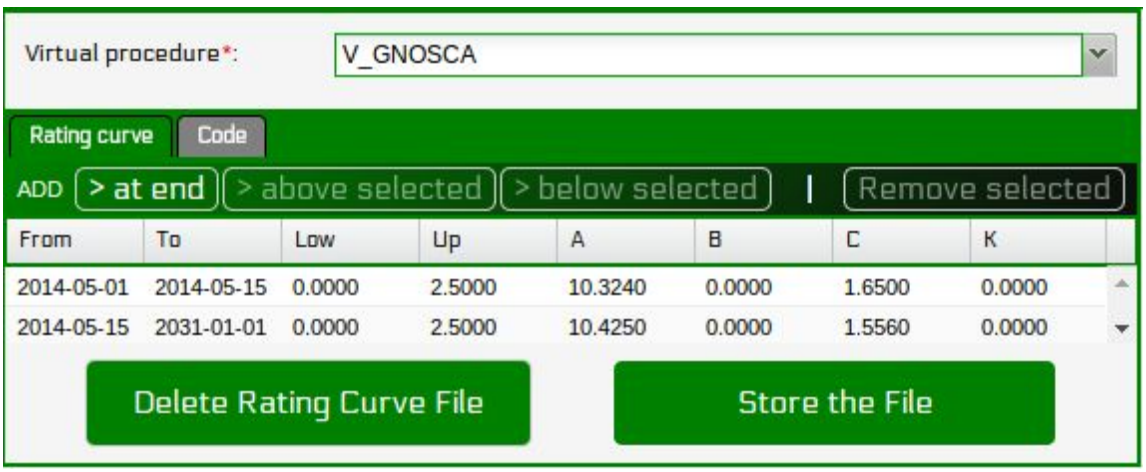
```
from istsoslib.responders.GOresponse import VirtualProcess
class istvp(VirtualProcess):
    procedures = {
        "T_LUGANO": "urn:ogc:def:parameter:x-istsos:1.0:meteo:air:temperature"
    }
    def execute(self):
        data = self.getData("T_LUGANO")
        out=[]
        for idx in range(len(data)):
            rec = data[idx]
            if self.filter.qualityIndex == True:
                out.append([rec[0], self.convert(rec[1]), rec[2]])
            else:
                out.append([rec[0], self.convert(rec[1])])
        return out
    def convert(self, celsius):
        if celsius is None:
            return -999.9
        return (float(celsius) *1.8 + 32)
```

Test the virtual procedure executing a getObservation request here

V_GNOSCA: transforms river water levels in river discharge appling a rating curve defined with different parameters for different periods.
Here is the code, which instantiate a special istSOS virtual procedure class (VirtualProcessHQ):

```
from istsoslib.responders.GOresponse import VirtualProcessHQ

class istvp(VirtualProcessHQ):
    procedures = {
        "RH_GNOSCA": "urn:ogc:def:parameter:x-istsos:1.0:river:water:height"
    }
```

Select the “Rating curve” tap panel to see the rating curves values:



Test the virtual procedure executing a getObservation request here

Creating a reference evapotranspiration virtual procedure manually

Evapotranspiration is an important parameter that takes part in the water balance assessment, and therefore in any attempt to calculate and predict plant water needs and optimized irrigation. ENORASIS use evapotranspiration as one of the component to feed the innovative advanced algorithms and optimization processes leading to irrigation optimized scheduling.

A possible approach it to assess water needs by using a daily water balance calculation considering rainfall and irrigations as the water inputs and crop evapotranspiration (ETc) as the loss. The water balance for a particular irrigation block, in millimetres, can then be calculated using the formula:

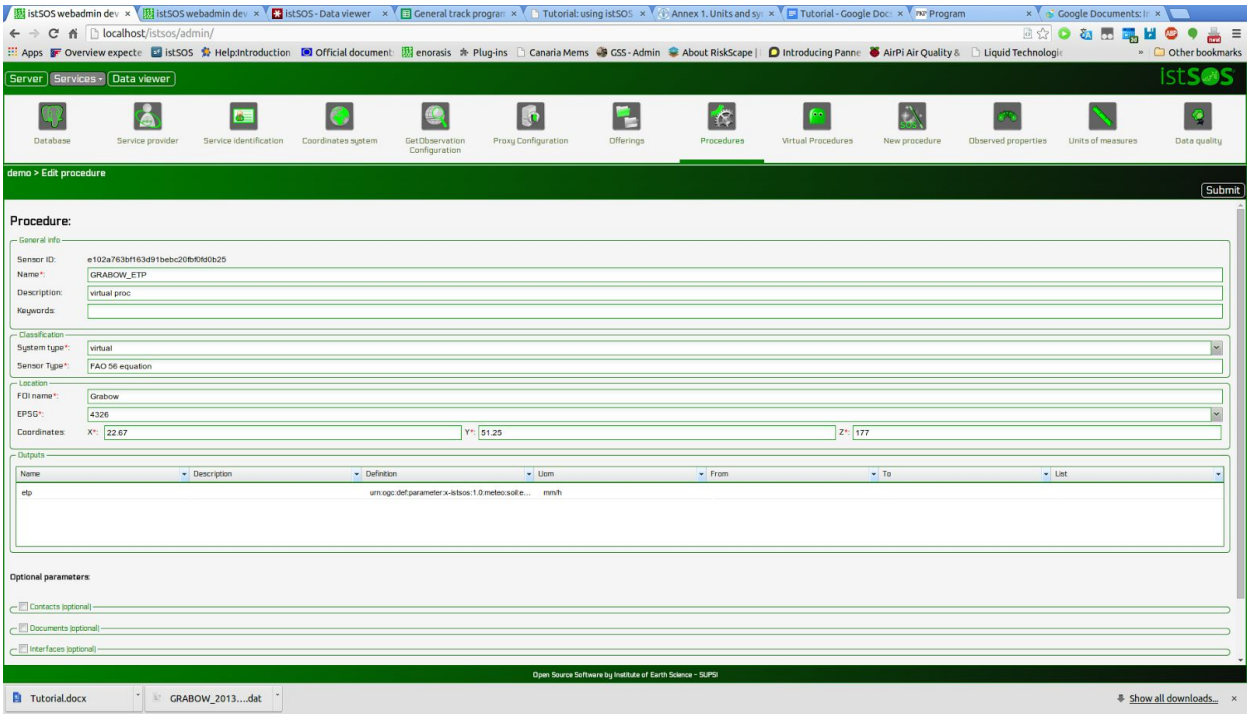
$$WB = R + I - ETc$$

Where:

- WB = water balance (mm)
- R = rainfall depth (mm)
- I = irrigation volumes (mm/area)
- ET c = crop evapotranspiration (mm/area)

In this approach, the water balance does not include water stored in the soil profile. This is a simplification but it avoids the difficulty of trying to build a complex system reliant on accurate soil moisture accounting, that in many circumstances adds little benefit as the other agronomic or cultural preferences are unknown. In contrast to ENORASIS, a system operating in this mode does therefore not tell irrigators when and how much to irrigate, but provides an indication on how much water the crop has used since last irrigation. The decision when to irrigate is up to the grower based upon all the agronomic, economic and social/cultural considerations.

Create a virtual procedure named ETP_GRABOW observing Evapotranspiration


istSOS webadmin dev
localhost/istsos/admin/
Server
Services
Data viewer
istSOS
Database
Service provider
Service identification
Coordinates system
GetObservation Configuration
Proxy Configuration
Offerings
Procedures
Virtual Procedures
New procedure
Observed properties
Units of measures
Data quality
demo > Edit procedure
Submit
Procedure:
General info
Sensor ID: e102a763bf163d91bebc20fbf0fd0b25
Name*: GRABOW_ETP
Description: virtual proc
Keywords:
Classification
System type*: virtual
Sensor Type*: FAO 56 equation
Location
FOI name*: Grabow
EPSG*: 4326
Coordinates: X*: 22.67 Y*: 51.25 Z*: 177
Outputs
Name
Description
Definition
Uom
From
To
List
etp
urn:ogc:def:parameter:x-istsos:1.0:meteo:soil:e...
mm/h
Optional parameters:
Contacts [optional]
Documents [optional]
Interfaces [optional]
Open Source Software by Institute of Earth Science - SUPSI
Tutorial.docx
GRABOW_2013....dat
Show all downloads...

⚒ Copy the FAO56 python function in the virtual procedure folder

If installed from source

```
sudo cp ~/Desktop/Tutorial/vp/FAO56.py /usr/local/istsos/services/demo/virtual/ETP_GRABOW
```

If installed from debian package

```
sudo cp ~/Desktop/Tutorial/vp/FAO56.py /usr/share/istsos/services/demo/virtual/ETP_GRABOW
```

Open the file and note the function ET0 which calculate evapotranspiration from a number of inputs:

```
'''
    =============================================================================
    Potential evaporation functions using Penman-Montheit with hourly data
    =============================================================================
'''

def ET0(isodate,T,RH,u2,Rs,lat,lon,z,P=None,verbose=False):

    """
    Input:
        isodate: (str) iso datetime in UTC
        T: (float) hourly air temperature at 2m [Celsius]
        RH: (float) hourly relative air humidity [Pa]
        u2: (float) hourly wind speed at 2 m [m/s]
        Rs: (float) hourly incoming solar radiation [J/m2/hour]
        lat: (float) latitude of the measurement point [decimal degree]
        lon: (float) longitude of the measurement point [decimal degree]
        z: (float) altitude above sea level of the measurement point [m]
        P: (float) hourly air pressure [Pa] (Opzional)

    Output:
        - ET0: (float) hourly reference evapotranspiration [mm/h]

    Examples::
        >>> import FAO56
        >>> FAO56.ET0(isodate="2015-10-01T02:00Z",T=28,RH=90,u2=1.9,
                      Rs=0,lat=16.21,lon=-16.26,z=8)
        >>>
        >>> FAO56.ET0(isodate="2015-10-01T14:00Z",T=38,RH=52,u2=3.3,
                      Rs=2.450,lat=16.21,lon=-16.26,z=8)
        >>> 0.626874880652

    References:
        http://www.fao.org/docrep/X0490E/x0490e00.htm#Contents

    """
```

⚒ Code the evapotranspiration virtual procedure

```
# -*- coding: utf-8 -*-
# import
from istsoslib.responders.GOresponse import VirtualProcess

import FAO56

class istvp(VirtualProcess):

    procedures = {
        "GRABOW": [
            "urn:ogc:def:parameter:x-istsos:1.0:meteo:air:temperature",
            "urn:ogc:def:parameter:x-istsos:1.0:meteo:air:humidity:relative",
            "urn:ogc:def:parameter:x-istsos:1.0:meteo:air:wind:velocity",
            "urn:ogc:def:parameter:x-istsos:1.0:meteo:solar:radiation"
        ]
    }

    def execute(self):

        data = self.getData("GRABOW")
        data_out = []
        for rec in data:
            if self.filter.qualityIndex == True:
                # rec is a list:
                # [0]=time, [1]=T,[2]=Tqi, [3]=RH,[4]=RHqi,
                # [5]=u2,[6]=u2qi, [7]=Rs,[8]=Rsqi
                etp = FAO56.ET0(isodate = str(rec[0]),
                      T=float(rec[1]),
                      RH=float(rec[3]),
                      u2=float(rec[5]),
                      Rs=float(rec[7]*0.0036), # W/m2 to MJ/(m2*h)
                      lat=22.67,
                      lon=51.25,
                      z=177)
                data_out.append([rec[0], etp, min([rec[2],rec[4],rec[6],rec[8]])])
            else:
                # rec is a list: [0]=time,[1]=T,[2]=RH,[3]=u2,[4]=Rs
                etp = FAO56.ET0(isodate = str(rec[0]),
                      T=float(rec[1]),
                      RH=float(rec[2]),
                      u2=float(rec[3]),
                      Rs=float(rec[4]*0.0036),
                      lat=22.67,
                      lon=51.25,
                      z=177)
                data_out.append([rec[0], etp])
        return data_out
```

Test the virtual procedure executing a getObservation request here

Mapping sensors and data with OpenLayers 3

Intro

In the *istsos/intereface/modules/ol-example*s folder you can find some examples using OpenLayers 3 (http://openlayers.org/).

- **Example 1**
  A simple OpenLayer 3 example with an OpenStreetMap basemap and a GeoJSON Vector overlay showing sensor's position.
  http://localhost/istsos/modules/ol-examples/example-1.html

- **Example 2**
  Example 1 + Styling
  http://localhost/istsos/modules/ol-examples/example-2.html

- **Example 3**
  Example 2 + Mouse click interaction displaying sensor's metadata
  http://localhost/istsos/modules/ol-examples/example-3.html

- **Example 4**
  Example 3 + Chart plot of last week of data
  http://localhost/istsos/modules/ol-examples/example-4.html

Practice with OpenLayers 3

If you want make some practice, we have prepared a file from which to begin to do some practice, let's go to:
http://localhost/istsos/modules/ol-examples/practice.html

Open the file practice.html so we can start..

If installed from source

```
sudo gedit /usr/local/istsos/interface/modules/ol-examples/practice.html
```

If installed from debian package

```
sudo gedit /usr/share/istsos/interface/modules/ol-examples/practice.html
```

Here an overview of the file that we have prepared. A simple OpenLayer 3 example with a OpenStreetMap basemap.

```
<!DOCTYPE html>
<html>
  <head>
    <link rel="stylesheet" href="ol/ol.css" type="text/css">
    <script src="ol/ol-debug.js"></script>
    <style>
      #map {
        width: 600px;
        height: 400px;
      }
    </style>
  </head>
  <body>
    <div id="map" class="map"></div>
    <script>
      var map = new ol.Map({
        layers: [
          new ol.layer.Tile({
            source: new ol.source.OSM()
          })
        ],
        target: 'map',
        view: new ol.View({
          center: [1614350, 6168773],
          zoom: 5
        })
      });
    </script>
  </body>
</html>
```

Loading istSOS sensors on the map

The istSOS WA REST exposes a request to retrieve a GeoJSON file including all the sensors offered by a istSOS instance.

Try to load the GeoJSON:
http://localhost/istsos/wa/istsos/services/demo/procedures/operations/geojson

You can also execute a reprojection by adding the **epsg** parameter:
http://localhost/istsos/wa/istsos/services/demo/procedures/operations/geojson?epsg=3857

To add the istSOS layer is quite simple. You have to create a Vector Layer with a Vector Source. And the source have to be configured defining the format (as GeoJSON) and the url (from where to download the GeoJSON data)

```
...
<script>
  var map = new ol.Map({
    layers: [
      new ol.layer.Tile({
        source: new ol.source.OSM()
      }),
      new ol.layer.Vector({
        source: new ol.source.Vector({
          format: new ol.format.GeoJSON(),
          url: '../../wa/istsos/services/demo/procedures/operations/geojson?epsg=4326'
        })
      })
    ],
    target: 'map',
    view: new ol.View({
      center: [1614350, 6168773],
      zoom: 5
    })
  });
</script>
...
```

Go to the practice.html page in your browser and press F5 to reload the map on the browser (now you should also see some circles representing the sensor position).

Changing the istSOS vector layer style

Modify code as shown in the next box, defining a custom style:

```
var map = new ol.Map({
  layers: [
    new ol.layer.Tile({
      source: new ol.source.OSM()
    }),
    new ol.layer.Vector({
      source: new ol.source.Vector({
        format: new ol.format.GeoJSON(),
        url: '../../wa/istsos/services/demo/procedures/operations/geojson?epsg=4326'
      }),
      style: [new ol.style.Style({
        image: new ol.style.Circle({
          radius: 5,
          fill: new ol.style.Fill({color: 'green'}),
          stroke: new ol.style.Stroke({color: 'red', width: 1})
        })
      })]
    })
  ],
  target: 'map',
  view: new ol.View({
    center: [1614350, 6168773],
    zoom: 5
  })
});
```

Reload (F5) the web page on the browser

Adding interaction to the map to display sensor metadata

Append after the map initialization this code to enable the “ol.interaction.Select” feature:

```
var map = new ol.Map({
  …
});
...
// select interaction working on "singleclick"
var select = new ol.interaction.Select({multi: true});
// Add the interaction to the map
map.addInteraction(select);
// Listen for select event
select.on('select', function(e) {
  var selected = e.selected,
      html = '';
  for (var c = 0, l = selected.length;c<l;c++){
    var feature = selected[c];
    html += feature.getProperties().name + "<br/><br/>" +
      "Begin: " + feature.getProperties().samplingTime.beginposition + "<br/>" +
      "End: " + feature.getProperties().samplingTime.endposition + "<br/><br/>" +
      "Observed properties:<br/>" ;
    html += "<ol>";
    var op = feature.getProperties().observedproperties;
    for (var cnt = 0; cnt < op.length; cnt++){
        html += "<li>" + op[cnt].name + "</li>";
    }
    html += "</ol><hr/>";
  }
  document.getElementById('details').innerHTML = html;
});
...
```

Reload (F5) the web page on the browser and click on a point displayed on the map. Sensor details will be displayed in the details div.

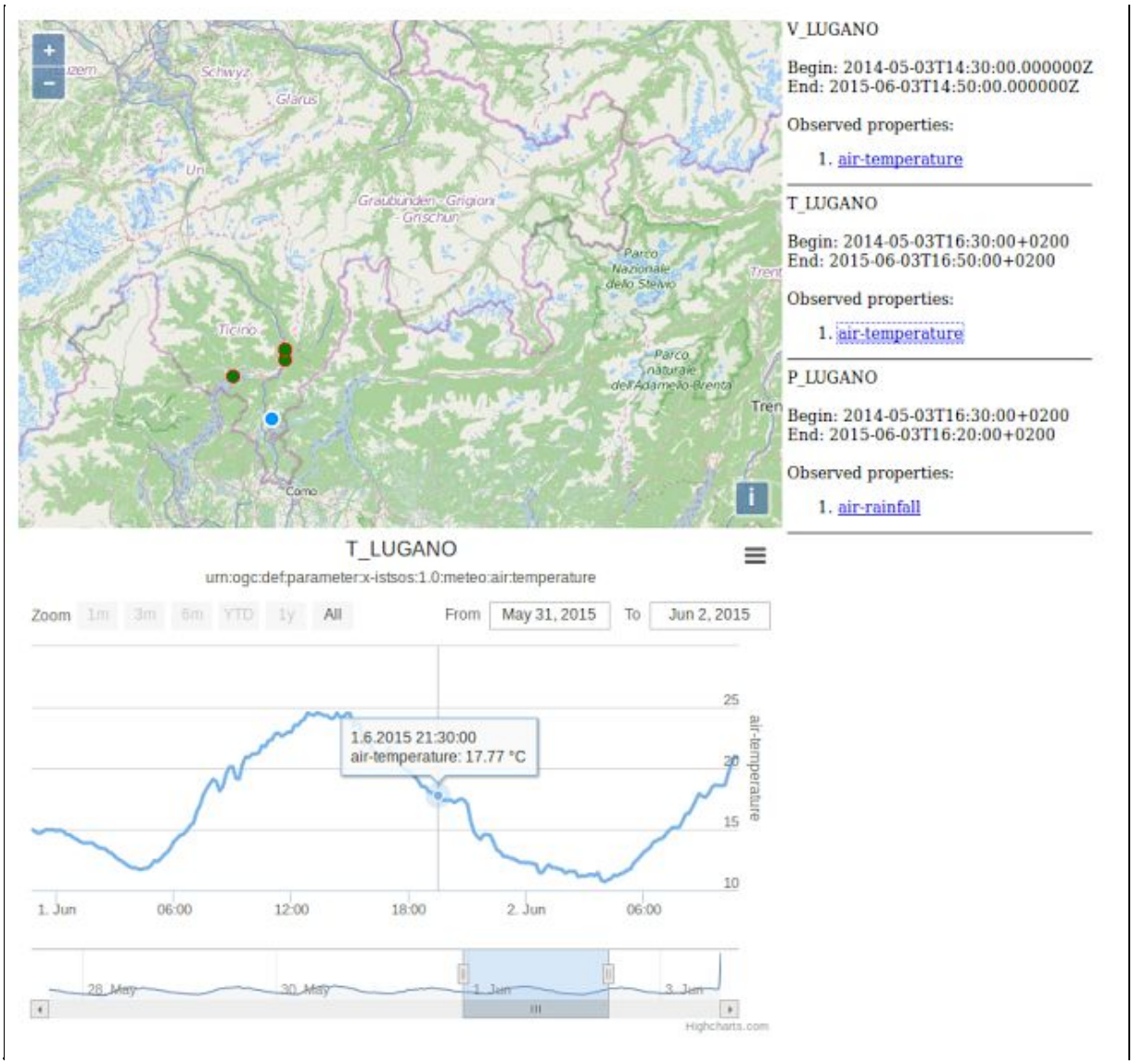
If you want to display other properties using the `feature.getProperties()` function, this are the attributes that can be accessed:


```
{
    "samplingTime": {
        "beginposition": "2007-01-01T00:00:00+0100",
        "endposition": "2011-12-31T23:50:00+0100"
    },
    "sensortype": "insitu-fixed-point",
    "observedproperties": [
        {
            "name": "water-height",
            "uom": "m"
        }
    ],
    "description": "",
    "name": "A_AETCAN_AIR",
    "assignedid": "8c4b9c18d464493568cfb18d015bbed5",
    "offerings": [
        "temporary"
    ],
    "id": 51
}
```

Plotting measures in a chart

The example-4.html (http://localhost/istsos/modules/ol-examples/example-4.html) is a little bit more for advanced user. Take a look a the code, here you can see how to plot observation measurements with the support of some well known JavaScript library: JQuery with MIT license (https://jquery.com) and Highcharts with a non-commercial license (http://www.highcharts.com).

⚠	**Highcharts** is one of the prettiest interactive charts on the web, unfortunately the licence is not from the FOSS family. But you can use the software for free under the non-commercial license. More info on non-commercial licensing can be found here: http://shop.highsoft.com/faq/non-commercial#what-is-non-commercial



```
// select interaction working on "singleclick"
var select = new ol.interaction.Select({multi: true});
// Add the interaction to the map
map.addInteraction(select);
// Listen for select event
select.on('select', function(e) {
  var selected = e.selected,
      html = '';
  for (var c = 0, l = selected.length;c<l;c++){
    var feature = selected[c];
    var name = feature.getProperties().name;
    var begin = feature.getProperties().samplingTime.beginposition;
    var end = feature.getProperties().samplingTime.endposition;
    html += name + "<br/><br/>" +
      "Begin: " + begin + "<br/>" +
      "End: " + end + "<br/><br/>" +
      "Observed properties:<br/>" ;
    html += "<ol>";
    var from = new Date(end);
    begin = new Date(end);
    begin.setDate(from.getDate() - 7);
    var op = feature.getProperties().observedproperties;
    for (var cnt = 0; cnt < op.length; cnt++){
        html += "<li><a
href='javascript:chart(\""+name+"\",\""+begin.toISOString()+"\",\""+end+"\",\""+op[cnt].def
+"\");'>" + op[cnt].name + "</a></li>";
    }
    html += "</ol><hr/>";
  }
  document.getElementById('details').innerHTML = html;
});

function chart(name, begin, end, observedProperty){

$.getJSON('../../wa/istsos/services/demo/operations/getobservation/offerings/temporary/proc
edures/'+name+'/observedproperties/'+observedProperty+'/eventtime/'+begin+'/'+end,
function(json) {

    //console.log(json);

    var sosData = json.data[0].result.DataArray.values;
    var field = json.data[0].result.DataArray.field[1];
    var data = [];

    for(var c = 0, l = sosData.length; c<l; c++){
      var date = new Date(sosData[c][0]);
      data.push([+date,parseFloat(sosData[c][1])]);
    }
    console.log(data);

    $('#chart').empty();

    $('#chart').highcharts('StockChart',{
      rangeSelector : {
          selected : 1
      },
      title : {
          text : name
      },
      subtitle: {
          text: field.definition
      },
      yAxis: {
          title: {
```

```
                text: field.name
            }
        },
        tooltip: {
            valueDecimals: 2,
            formatter: function() {
              var date = new Date(this.x);
              return date.toLocaleString() + "<br/>" +
                     field.name + ": " +
                     this.y + ' ' + field.uom;
            }
        },
        series: [{
          name: field.name,
          data: data,
          labels: {
              formatter: function () {
                  return this.value + " - stic";
              }
          }
        }]
      });
    });
}
```

Arduino with istSOS

Connecting Arduino with DHT11 sensor and upload data to istSOS

Intro

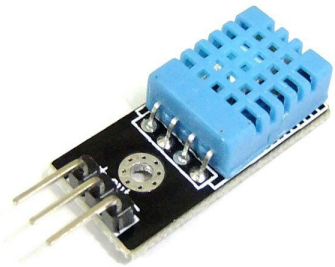
Arduino UNO Rev. 3

Arduino is an open-source prototyping platform based on easy-to-use hardware and software. Arduino boards are able to read inputs - light on a sensor, a finger on a button, or a Twitter message - and turn it into an output - activating a motor, turning on an LED, publishing something online. All this is defined by a set of instructions programmed through the Arduino Software (IDE).

For this workshop we use a Arduino Uno board.

DHT11 sensor

The DHT11 is a basic, ultra low-cost digital temperature and humidity sensor. It uses a capacitive humidity sensor and a thermistor to measure the surrounding air, and spits out a digital signal on the data pin (no analog input pins needed). It's fairly simple to use, but requires careful timing to grab data. The only real downside of this sensor is you can only get new data from it once every 2 seconds, so when using our library, sensor readings can be up to 2 seconds old.



Details:

- Good for 20-80 % humidity reading with 5% accuracy
- Good for 0-50 °C temperature reading +/- 2°C accuracy
- Low cost

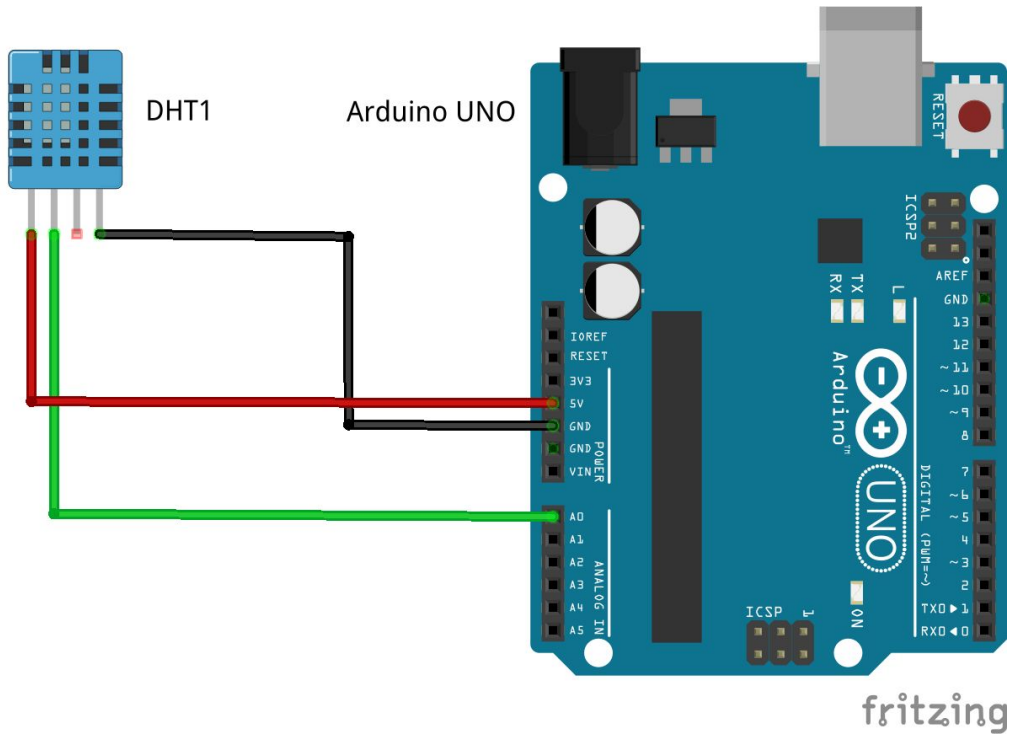
Mounting instruction

⚒ Connect the Sensor to Arduino board

Connect the DHT11 sensor to Arduino board

DHT11 pin	Arduino Pin
+	5V
out	A0
-	GND

Schema

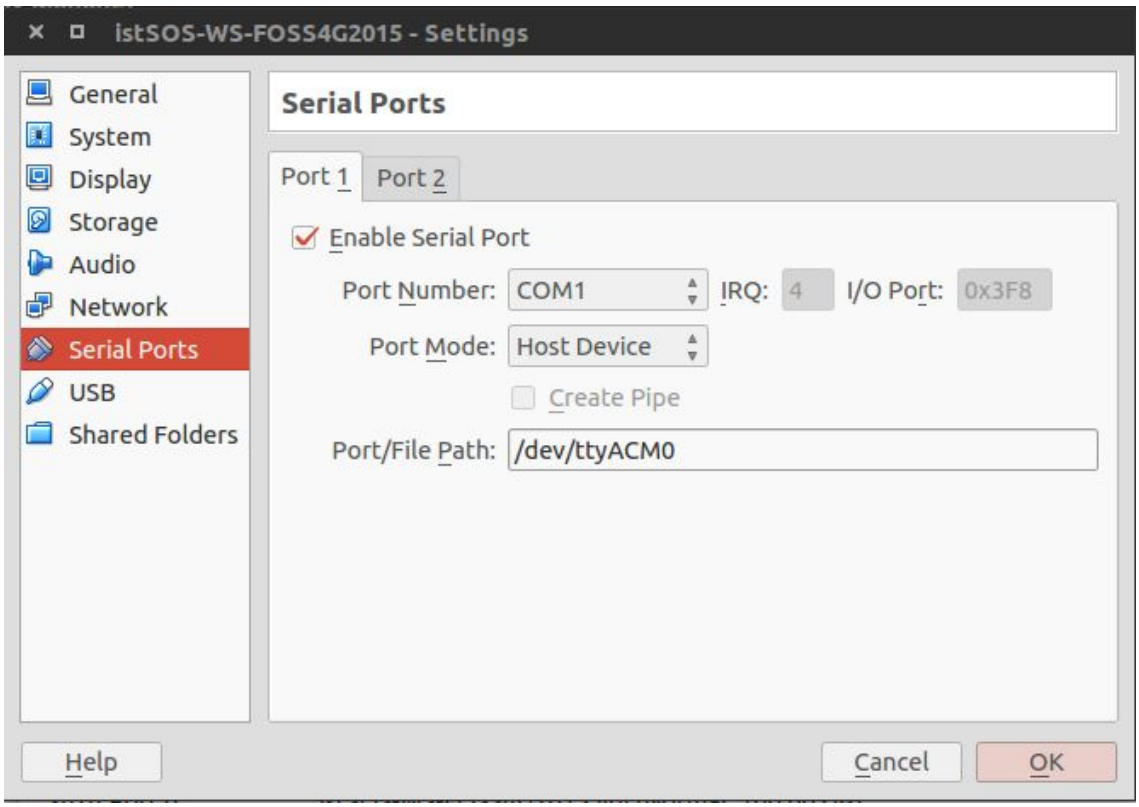


Setup

⚒ Connect Arduino with the VirtualMachine

Arduino can be connected to your PC through the USB port. The mainboard use the serial port to communicate. To enable a serial connection with the a VirtualBox VirtualMachine you have to:

- Shut down the VM
- Verify that your Arduino is connected with your PC
- Activate the Setting windows from your VirtualBox Manager
- Set configuration parameters as in the following image



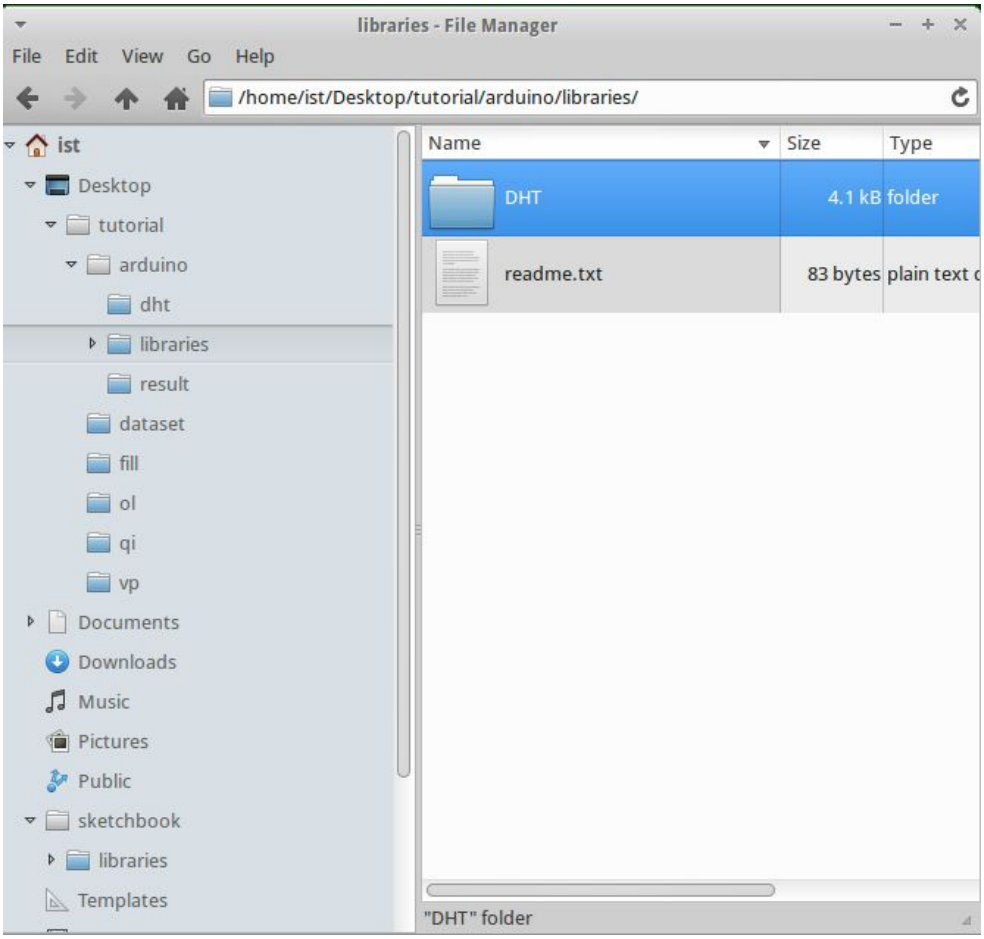
- Restart your VM !

⚒ Set your Arduino Software environment

Arduino boards are able to read inputs and turn it into an output. All this is defined by a set of instructions programmed through the Arduino Software (IDE) and uploaded to the microcontroller.

- Open the arduino IDE and add the user to the other group at the prompt if asked.
- Log out from your from your session and log-in again

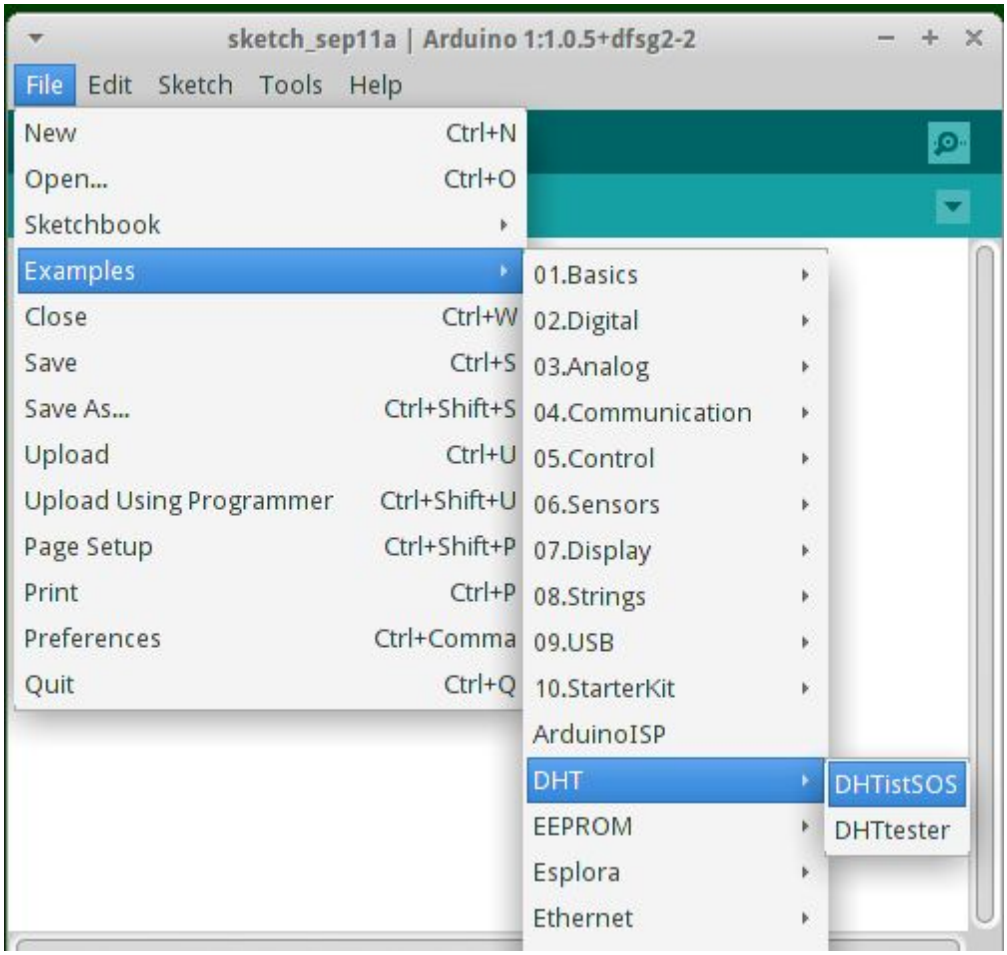
- copy the DHT libraries from */home/ist/Desktop/Tutorial/arduino/libraries/* to the */home/ist/sketchbook/libraries* folder



⚒ Upload to Arduino the script to read data

- Open your Arduino IDE program

- Select from the example the DHTistSOS (File->Examples->DHT->DHTistSOS)

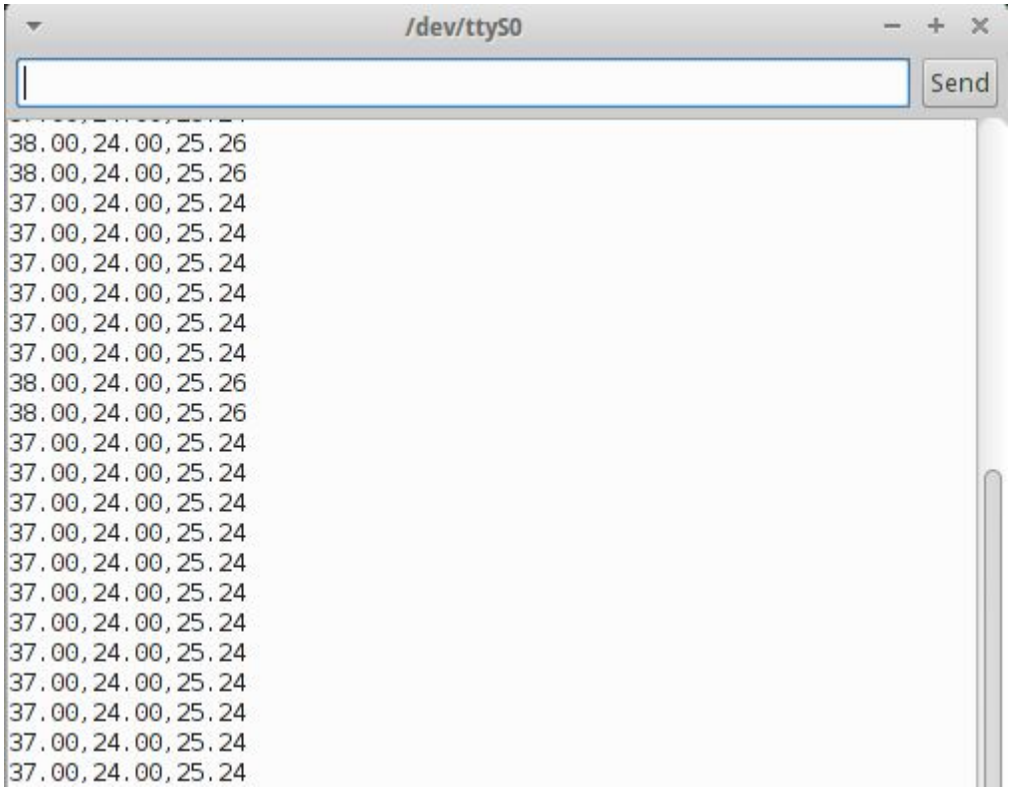


⚠	Before uploading the program to the board verify the code to avoid error like missing library. 

- If no error upload the code to the Arduino board.  .

⚒ Visualize your measurements

- To check if the the program is correctly running use the serial monitor.



Data acquisition

Now the arduino script read the data from the sensor and send the result through the serial port. The data are sent as a CSV triplet of:

- air relative humidity
- air temperature,
- air heat index:

Store data into csv file (subtype istSOS)

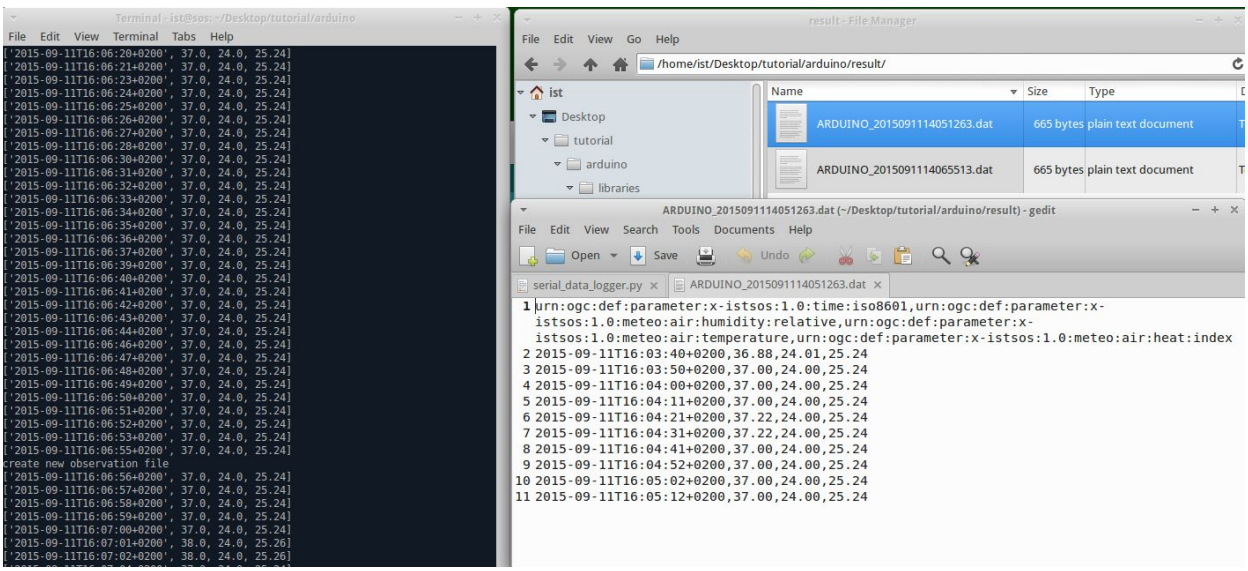
To store the data read from Arduino we have prepared a utility script: **serial_data_logger.py**

This script (found in Tutorial/arduino) reads the data from a serial port and write a series of files in *text/csv;subtype=istSOS* under the folder ***Tutorial/arduino/result***

⚠	To avoid bad read please close the serial monitor

Run the script with the following parametres:

```
python serial_data_logger.py -c arduino_config.json -s /dev/ttyS0
```



⚠	To avoid bad read please close the serial monitor

ⓘ	The "**serial_data_logger.py**" file, is a python script that makes use of pyserial and tzlocal to create data log files in a folder.
	
	`python serial_data_logger.py --help`
	`usage: serial_data_logger.py [-h] -s S [-b B] -c C`

```
Load data read from a serial port inside csv file.

optional arguments:
  -h, --help          show this help message and exit
  -s S, --serial S    Serial port to listen
  -b B, --baudrate B  serial port baudrate
  -c C, --config C    path to configuration file
```

Il file di configurazione è come segue

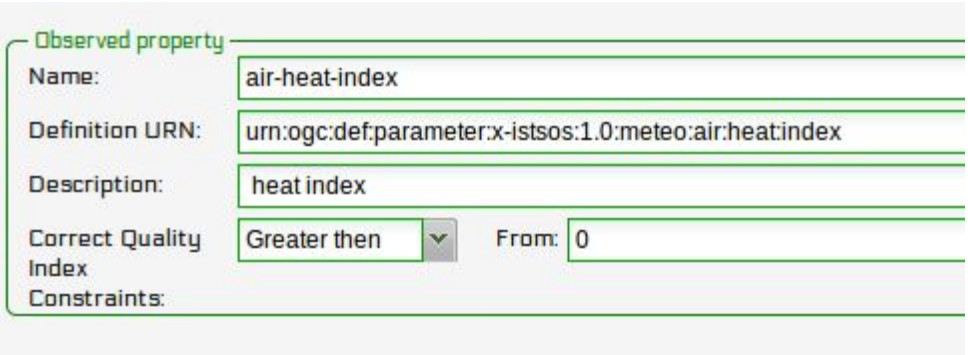
```
{
   "propertiesAggregation":{
        "urn:ogc:def:parameter:x-istsos:1.0:meteo:air:humidity:relative" :
"AVG",
        "urn:ogc:def:parameter:x-istsos:1.0:meteo:air:temperature": "AVG",
        "urn:ogc:def:parameter:x-istsos:1.0:meteo:air:heat:index" : "AVG"
   },
   "procedure": "ARDUINO",
   "dataResolution": 10,
   "measuresPerFile": 10
}
```

where the fields are:

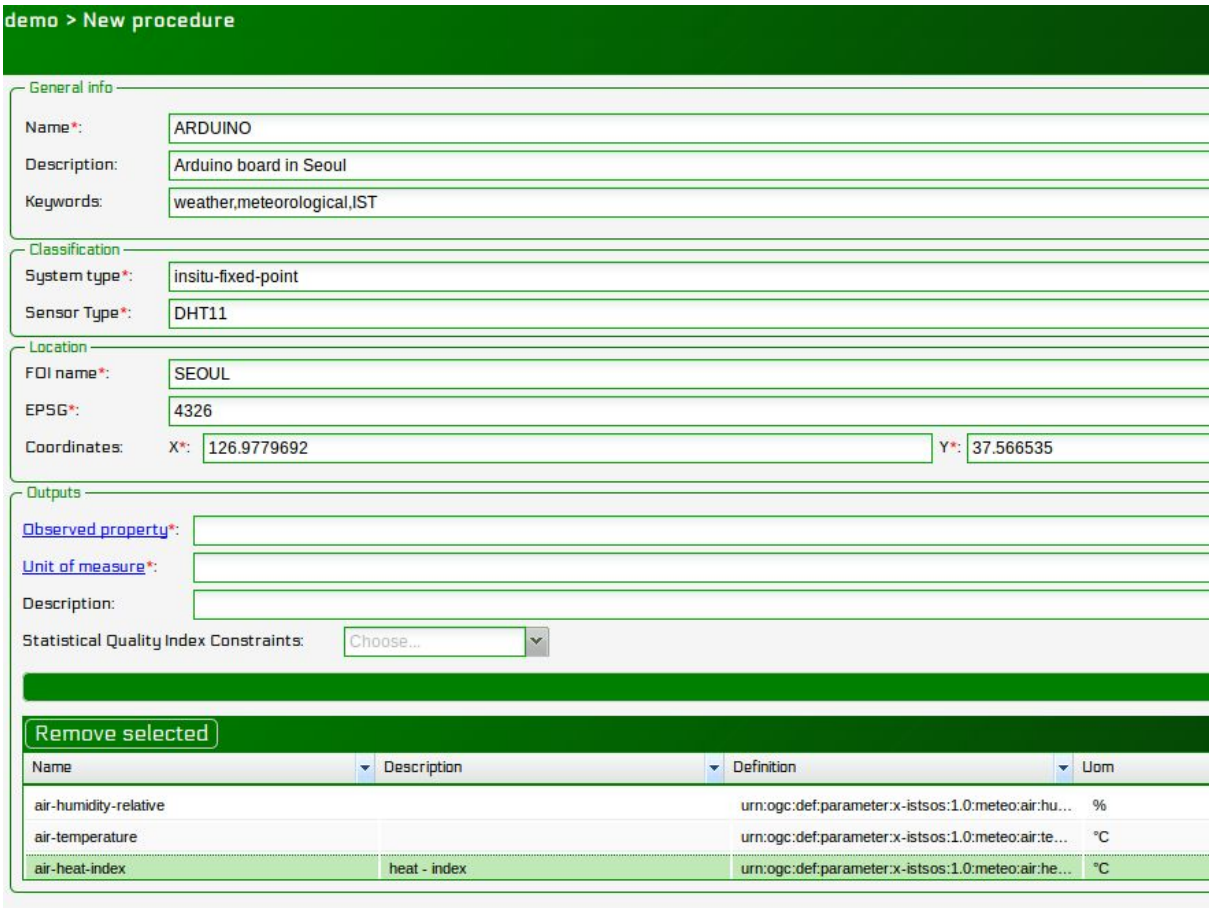
- *propertiesAggregation*: observed property and aggregate function (AVG, MIN, MAX) - !!! Please pay attention to the order. Observed properties must be in the same order in which data is received from Arduino. !!! -
- *procedure*: procedure name
- *dataResolution*: expressed in second, the system add a new measure every x second
- *measurePerFile*: how many measure to create a new file

⚒ Create your istSOS procedure

Set your Observed properties if required (humidity temperature, heat-index):



and your ARDUINO procedure:



⚒ Upload your log files to istSOS

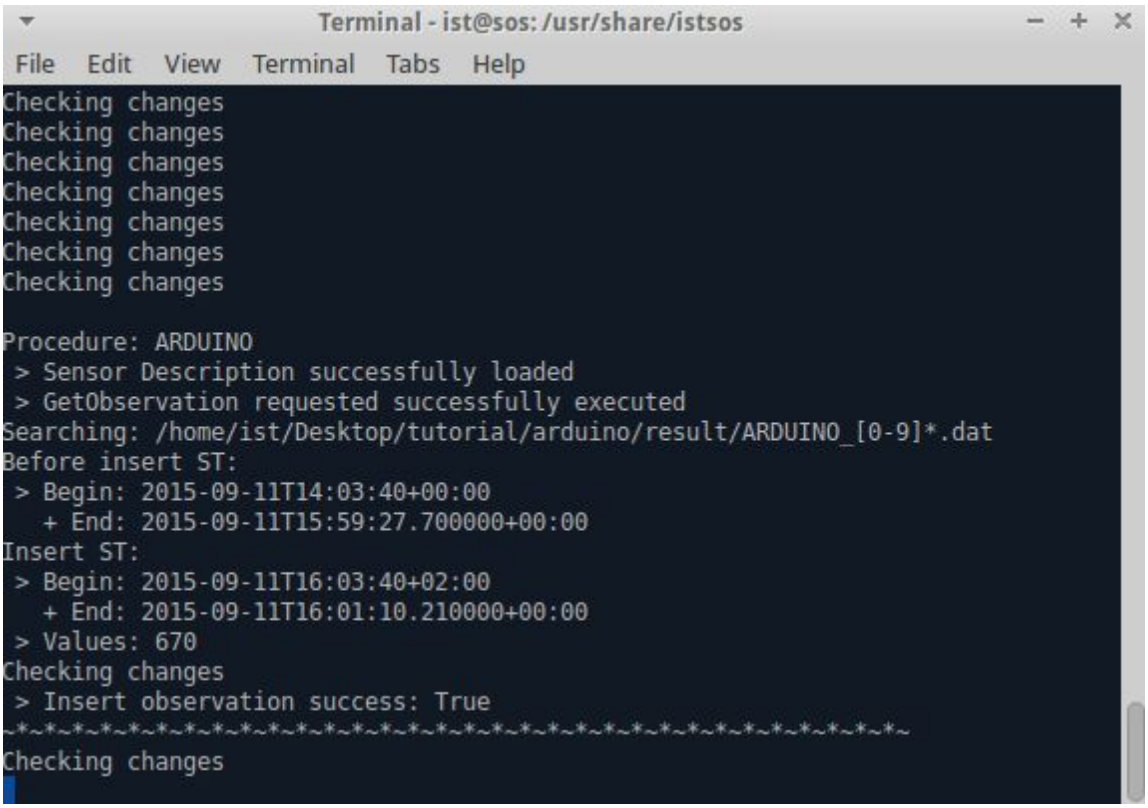
Open a new terminal and copy the file **~/Desktop/tutotial/arduino/demo.aps** to the **/usr/share/istsos/services/demo**

```
sudo cp ~/Desktop/tutotial/arduino/demo.aps /usr/share/istsos/services/demo
```

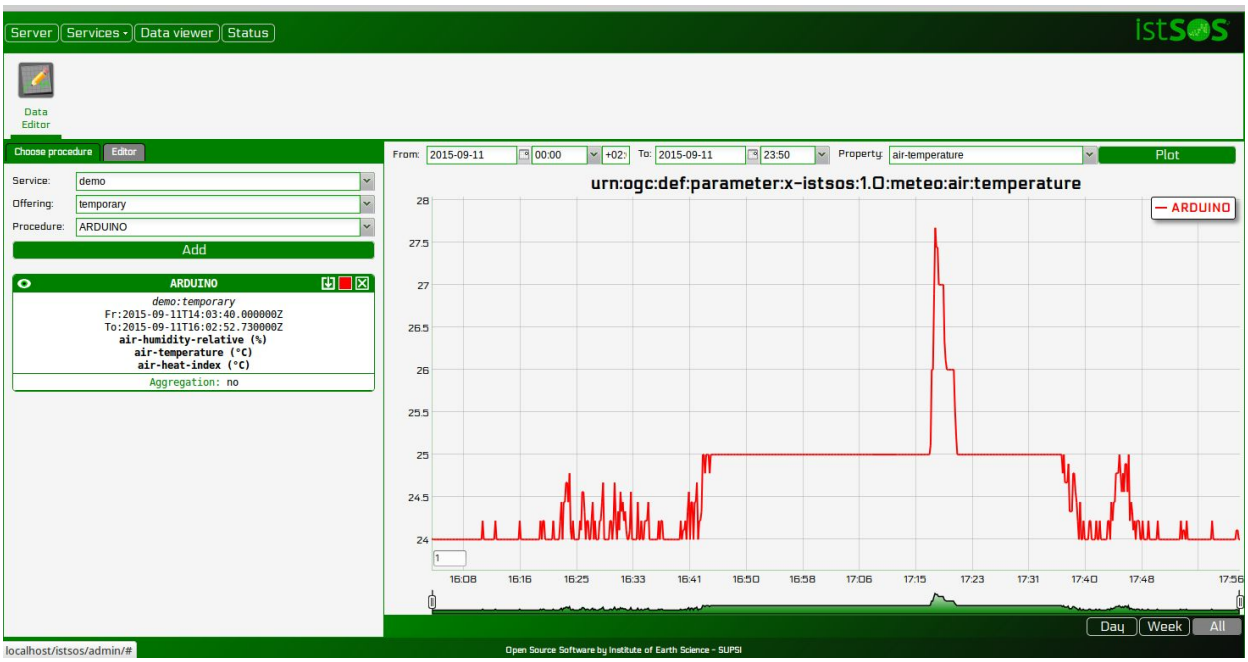
Now start the scheduler:

```
cd /usr/share/istsos/
python scheduler.py
```

You should see data uploading:



Once the first slot of data have been uploaded you can check your data in the



istWNS

istSOS Web Notification Service

Introduction

the istWNS is a service gathering data from an istSOS database and sending a notification to the users after testing the retrieved data to meet some conditions. The systems is divided in three parts: a database for storing the information about the notifications, the users and the registrations of the users to the notifications; a database of the istSOS service storing the actual data received from external sensors; and a scheduler that periodically runs the functions to retrieve the data, test it and send the notifications to the registered users. The system could also do a notification on a twitter account.

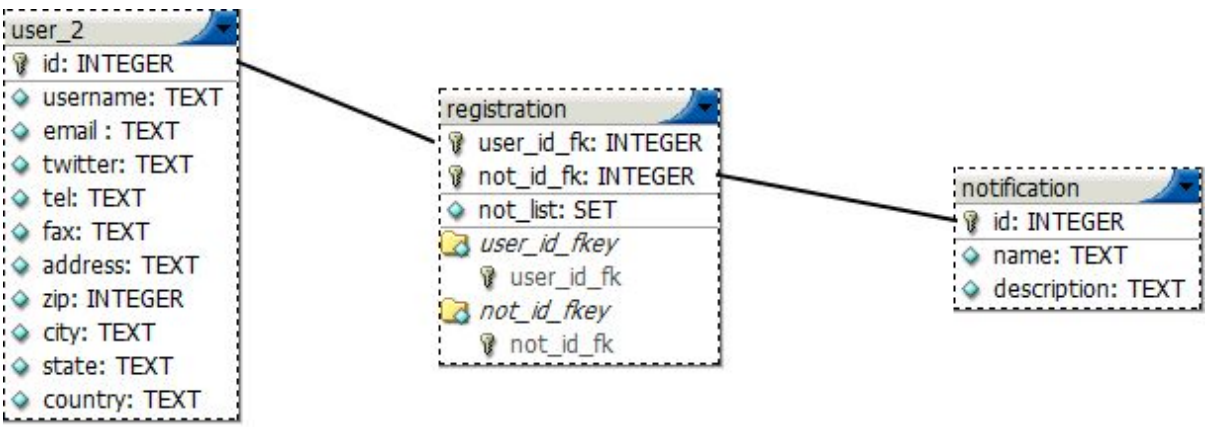
The following features are supported:

- creation/deletion of notifications
- creation/deletion of users
- subscription/unsubscription of users to notifications

	**THIS FEATURE IS EXPERIMENTAL - WE ARE WORKING HARD TO RELEASE SOON A STABLE VERSION** A web interface is still missing, so we need to handle modifying some files and using the poster Firefox extension (chrome://poster) that allows to send requests to a server ;-)

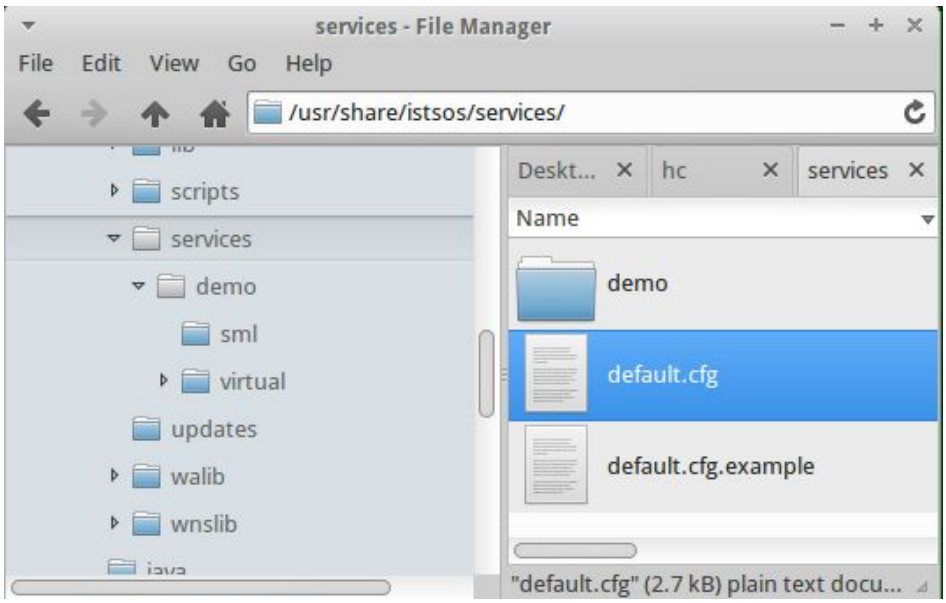
istWNS database schema

The wns schema is represented below.



Activation of istWNS

In istSOS each service configuration and related files are stored in the folder service (/usr/share/istsos/services/). In this folder there is a *default.cfg* file which store the service general configuration.



Configure the istWNS

By default the istWNS uses the database connection of the intended service but you could specify a different db setting specific connection parameters. What you **MUST** set up are at least one of the credentials for using mail or twitter for notifications or alert. The other parameter reported below are used to send notification via Gmail or twitter: **we use alert for this tutorial.**

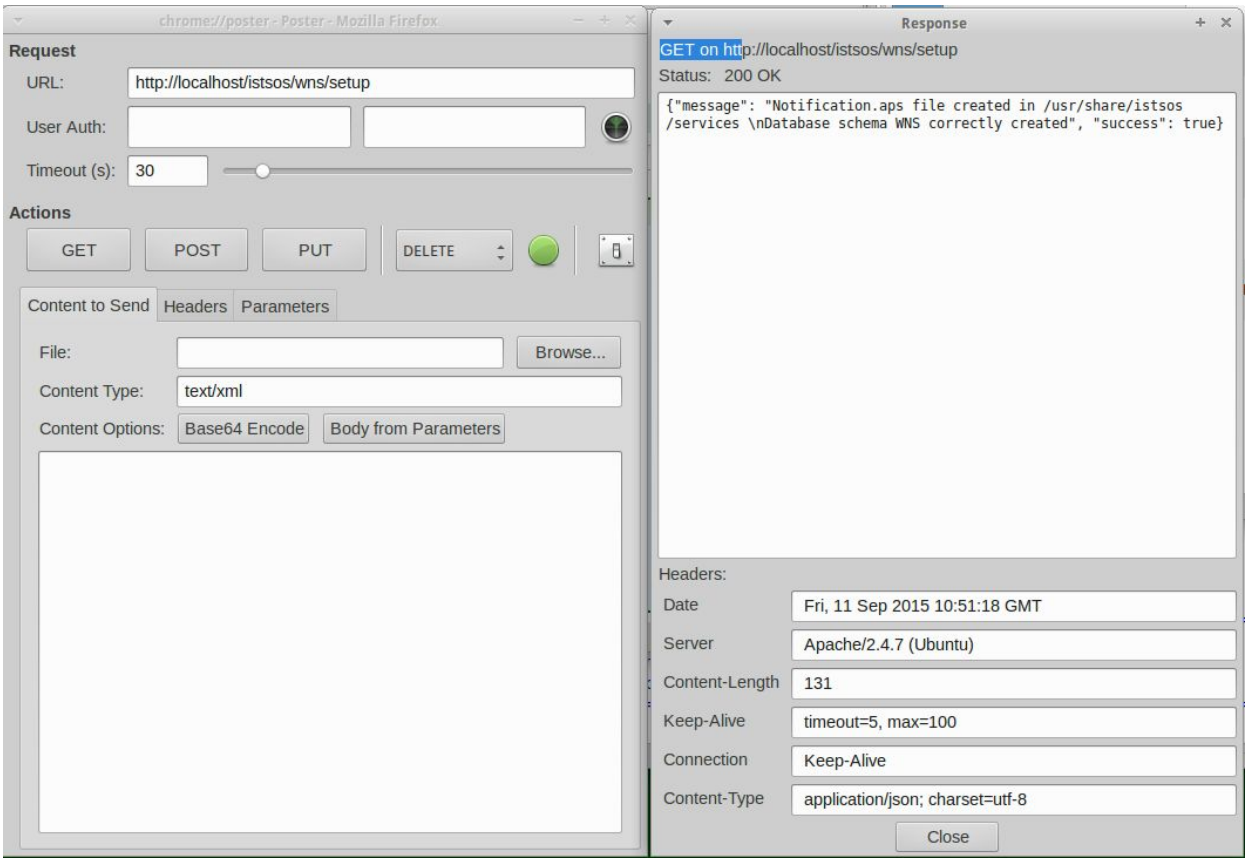
```
[connectionWns]
dbname = istsos
host = localhost
user = postgres
password = postgres
port = 5432

[alert]
active = true

[mail]
usermail = ""
password = ""

[twitter]
oauth_token = ""
oauth_secret = ""
consumer_key = ""
consumer_secret = ""
```

When a notification is raised, the system send a message to the user.

⚠	Actually the system update the status on twitter and can send notification via email or popup an alert box.

Set-up the istWNS

To setup the istSOS Web Notification Service open the poster (CTRL+ALT+P) add-on in your browser and execute this GET request: http://localhost/istsos/wns/setup.



This will create a notification.asp file under the service/ folder (where the notification function are stored) and create a schema in the istWNS database.

Create notifications

It's possible to create two types of notification:

- **simple notification**, that execute a getObservation and compare the results with a condition
- **complex observation**, the user write a specific requests and validation functions in Python.

Create a simple notification

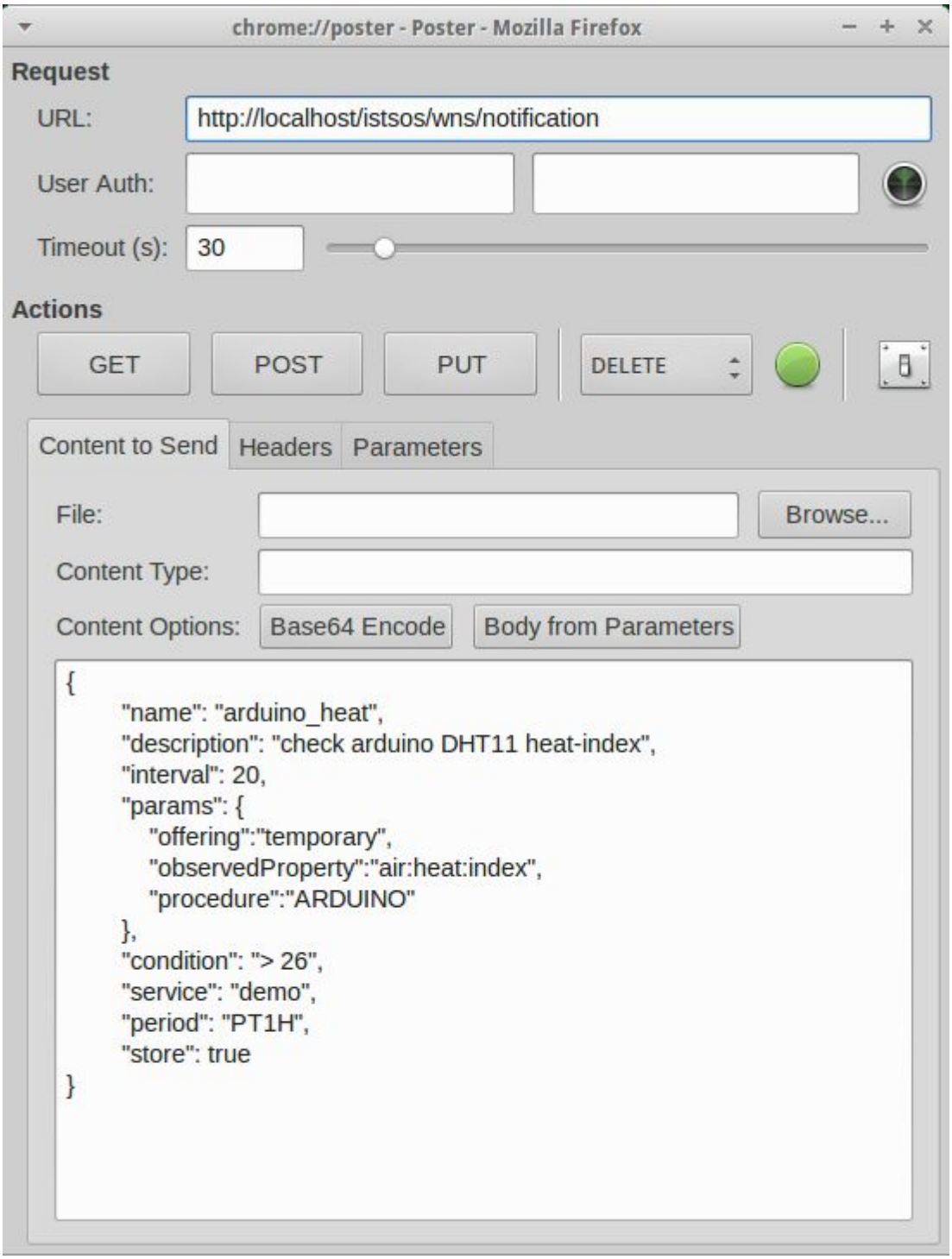
⚒ Create a json body with the following parameter

- *name*: the function name [mandatory]
- *description*: the description indicates what the function does [mandatory]
- *period*: expressed in hours, is the interval over which the getObservation will be performed, starting from now i.e. the last 2 hours. [optional]
- *interval*: expressed in minutes [mandatory]
- *service*: the service name [mandatory]
- *condition*: the condition describes in which case the notification will be performed. Every element retrieved with the getObservation is tested again this condition, and as soon one element satisfies it the notification is triggered.
- *params*: this is used to build a getObservation request. The *offering*, *observedProperty* and *procedure* are mandatory.
- *store*: flag, if true the system store the response

```
{
	"name": "arduino_heat",
	"description": "check arduino DHT11 heat-index",
	"interval": 20,
	"params": {
	   "offering":"temporary",
	   "observedProperty":"air:heat:index",
	   "procedure":"ARDUINO"
	},
	"condition": "> 26",
	"service": "demo",
	"period": "PT1H",
	"store": true
}
```

⚒ Upload and activate the new notification

execute a POST request to the following url: http://localhost/istsos/wns/notification



Register a user

To subscribe a notification and receive updates you must be a registered user.
To create a user you need to provide some personal data.

⚒ Prepare a JSON with the required data

The JSON shall contain the following fields:

- *username*: the user name [mandatory]
- *email*: a user mail contact [mandatory]
- *name*: user first name [mandatory]
- *surname*: user last name [mandatory]
- *twitter*: twitter account name, [required to receive notification via twitter private message]
- *tel*: mobile phone number, [required to receive nofification via SMS]
- *fax, address, zip, city, state, country*: additional info about the user

```
{
  "username": "pippo",
  "email": "pippo@gmail.com",
  "name": "Pinco",
  "surname": "Pallino",
  "twitter": "pluto",
  "tel": "+41123456789",
  "fax": "+41123456080",
  "address": "via Trevano",
  "zip": "12",
  "city": "Lugano",
  "state": "Ticino",
  "country": "Switzerland"
}
```

⚒ Activate the user

Send an HTTP POST request with the previously prepared JSON as request body at the following url http://localhost/istsos/wns/user

Subscribe to a notification

To receive notification you must subscribe to an existing notification.
To subscribe a notification you have to know the *user_id* and the *notification_id*.

⚒ Prepare a JSON with the required data

The only field is *data*w hich is mandatory and shall be of type array.
Current supported elements values are *mail* and *twitter*.

```
{
    "data": [ "alert"]
}
```

⚒ Subscribe the user to a notification

Send an HTTP POST request with the previously prepared JSON as request body at the following url: http://localhost/istsos/wns/user/<user_id>/notification/<notification_id> POST

Unsubscribe a notification

You can delete a user with this simple HTTP DELETE request:

- http://localhost/istsos/wns/user/<user_id>/notification/<notification_id>

Activate the scheduler

To activate the scheduler move to the istsos installation folder and run the scheduler script.

```
cd path_to_istsos
python scheduler.py
```

now….

Warm your sensor and look for a pop-up notification!

Create complex notification

⚒ Create a notification script

Open a new file in a text editor and write the Python function that will

1. access the data
2. raise a notification if conditions are meet

The following rules must be respected in writing the function:

1. The name of the function is the name of the notification
2. The name of the function must be unique
3. The function must call the *notify* method of the wnslib to send notification

An example of complex notification function that evaluates and sends notifications if the average value of the last 5 hours of temperature registered at the sensor named T_TRAVANO is greater than 30 Celsius degrees with different messages depending on the hazard level:

```
def temp5h():
    import datetime
    import time
    from pytz import timezone
    now = datetime.datetime.now().replace(tzinfo=timezone(time.tzname[0]))
    endDate = now.strftime('%Y-%m-%dT%H:%M:%S%z')
    eventTime = now - datetime.timedelta(hours=5)
    startDate = eventTime.strftime('%Y-%m-%dT%H:%M:%S%z')

    rparams = {"service": "SOS", …,"procedure": "T_TREVANO"}
    rparams['eventTime'] = str(startDate) + "/" +str(endDate)

    import lib.requests as requests
    res = requests.get('http://localhost/istsos/demo', params=rparams )

    result = res.json()
    values = result['ObservationCollection']['member'][0]['result']['DataArray']['values']

    mean = 0
    count = 0

    for el in values:
        if float(el[1]) != -999.9:
            sum += float(el[1])
            count += 1

    if len(result) == 0:
        message = "Cannot make mean with no data"
    else:
        mean = sum / count
        message = "The ... 5h in " + rparams['procedure'] +" was: " + str(mean)

    notify = {
        "twitter": {
            "public": "public message, update status",
            "private": "private message to someone"
        },
        "mail":{
            "subject": "notification from temp5h procedure",
```

```
            "message": message
        }
    }

    if mean > 40:
        import wnslib.notificationScheduler as nS
        nS.notify('temp5h',notify, False)
```

The function *temp5h* retrieves the data, handles them and checks a condition to send out notifications.

1. Pay attention to the function name you choose, because the exact name has to be used in the next step. The name also has to be unique, to avoid potential overriding.
2. The if block at the end of the method is the one triggering the notification, if the given condition is met. The two lines specified in the extract should be copied in your method, to make sure you import the correct file.
3. The ns.notify() method takes three arguments:
    a. functionName of the method you defined [Mandatory]
    b. a python dict containing the message to send via twitter or mail [Mandatory]
    c. Status: the last parameter is a flag, if True, the Notifier update the status of the twitter account [Optional, default True].

⚒ create a json with the following params:

- *name*: notification name [mandatory] (notification name == function name)
- *description*: a little description of the notification
- *interval*: interval between two checks expressed in minutes
- *function*: path to the function file
- *store*: flag, if true the system store the response

```
{
    "name": "temp5h",
    "description": "mean temp in the last 5 hours",
    "interval": 300,
    "function": "path/to/function.py",
    “store”: true
}
```

⚒ Activation of the notification

Execute an HTTP POST request at the following url: http://localhost/istsos/wns/notification

Delete notifications

You can delete a notification with this simple HTTP DELETE request:

- http://localhost/istsos/wns/notification/<notification_id>

	Note that you can delete a notification only if no user are subscribed

Delete a user

You can delete a user with this simple HTTP DELETE request:

- http://localhost/istsos/wns/user/<user_id>

	When you delete a user it is automatically unsubscribed from notifications