European Commission

JRC TECHNICAL REPORTS

AirSensEUR: an open data/software /hardware multi-sensor platform for air quality monitoring.

Part C: INSPIRE and Interoperable Data Management

Alexander Kotsev, Michel Gerboles, Laurent Spinelle, Marco Signorini, Simon Jirka, Matthes Rieke, Sven Schade, Massimo Craglia and Maria Gabriella Villani

2017

EUR 28928 EN

This publication is a Technical report by the Joint Research Centre, the European Commission's in-house science service. It aims to provide evidence-based scientific support to the European policy-making process. The scientific output expressed does not imply a policy position of the European Commission. Neither the European Commission nor any person acting on behalf of the Commission is responsible for the use which might be made of this publication.

Contact information
Name: Alexander Kotsev
Address: Via Enrico Fermi, 2749, I-21027 Ispra (VA), Italy
E-mail: alexander.kotsev@ec.europa.eu
Tel.: +39 0332 78 9069

JRC Science Hub
https://ec.europa.eu/jrc

JRC109337

EUR 28928 EN

ISBN 978-92-79-77049-4

ISSN 1831-9424

doi:10.2760/93699

Abstract
The use of mobile low cost sensors for regulatory purposes or citizen observatory projects is quickly developing. Although several sensor systems have appeared during the last years, generally developers cannot tackle the whole set of open problems regarding their use and the accuracy of the information they provide. Currently air pollution using low cost sensors involves fast growing private companies and/or public institutes specialized in digital technologies. Unfortunately, a great deal of effort is wasted by repeating the same research by several private companies/public institutes or within several European projects. Moreover, when a technological solution is found, it is generally subjected to confidentiality aspects of the intellectual property right.
In order to facilitate the use of sensors by diminishing the operational and development costs, the Joint Research Centre is developing a multi-sensor platform, called AirSensEUR, for the monitoring of air pollution at low concentration levels. All development aspects owned by JRC about AirSensEUR are made available to all stockholders though the use of public licenses. AirSensEUR is developed as an open software/open hardware object that has the capacity to behave as a node within a network of multi sensors assuring interoperability and compliance with the INSPIRE Directive. This part of the JRC technical report provides overview of the procedures applied within the AirSensEUR architecture in order to ensure interoperability of the data exchange.

Executive summary

The issue of using mobile low-cost sensors by local authorities, laboratories in charge of air quality monitoring for regulatory purposes or by citizens is quickly developing. Several sensor systems have been developed in the last years by fast growing private companies and/or public institutes specialized in digital technologies. Unfortunately, these sensors do not give satisfaction for all desired functionalities including the measurement accuracy of the sensor systems, the localization and identification of the micro-environments being sampled and the reporting of these data in order to make them available to all stakeholders through the Web services. Moreover, a great deal of effort is wasted by repeating the same research by several companies/institutes or within research projects. Additionally, new technological solutions that may be found are generally subjected to confidentiality of the intellectual property rights preventing them to be widely applied.

In order to facilitate the use of sensors by diminishing the operational and development cost, the Joint Research Centre (JRC) is developing a multi-sensor platform, called AirSensEUR, for the monitoring of air pollution at low concentration levels with low-cost sensors. JRC makes all development aspects about AirSensEUR freely available through the use of public licenses. In fact, AirSensEUR is developed as an open software/open hardware object that has the capacity to behave as a node within a network of multi sensors assuring interoperability and compliance with the INSPIRE Directive (Infrastructure for Spatial Information in the European Community).

The JRC objective was to build both the AirSensEUR host platform able to connect with several sensor shields to be later developed and to develop the first sensor shield for air pollution sensors. The sensor shield is connected to the host in charge of collecting the measurements. The host supplements measurements with geographical coordinates and it sends data to a public database on the web where mapping and query services are available.

This report describes the configuration of the server and data exchange, while other aspects of the project such as shield, host, etc. are presented in other EUR reports.

AirSensEUR is a promising technology for the monitoring of air pollution at fixed sites and for population exposure in mobile context at low cost.

Contents

1. Introduction

The diffusion of sensors, smartphones and tablets, social media platforms and phenomena such as big and open data are fundamentally changing the way data underpinning policy and science are conceived. This is creating many opportunities for policy-making and science. There are however completely new challenges which need to be tackled. There can be major benefits from the deployment of the Internet of Things (IoT) in smart cities and/or for environmental monitoring, but to realize such benefits, and reduce emerging risks, there is a pressing need to address current limitations, including (i) the interoperability of sensors, (ii) data quality, (iii) security of access and (iv) new methods for spatio-temporal analysis.

Notably, recent ICT advancements have led to the production of arrays of miniaturized sensors, often embedded in existing multitasking devices (e.g., mobile phones, tablets) and using a wide range of radio standards (e.g., Bluetooth, WiFi, 4G cellular networks). Altogether, these technological evolutions coupled with the diffusion of ubiquitous Internet connectivity provide the ground line technology enablers for the IoT intended as the "world-wide network of interconnected objects uniquely addressable, based on standard communication protocols" (European Commission, DG INFSO; "Internet of Things in 2020", 2008). By 2020, the number of IoT devices is expected to exceed 50 billion units (Swan, 2012). Hence, the combined deployment of IoT-type of sensing devices and their virtual representation through the web will enable the construction of a multi-dimensional mosaic of information for better monitoring and understanding our complex environment. Nevertheless, the sheer amount of information sources and users will ensure a number of unprecedented challenges in both policy and network management.

Considering this context, this JRC technical report provides an overview of data management for the AirSensEUR platform. AirSensEUR establishes an affordable open software/hardware multi-sensor platform, which is able to monitor air pollution at low concentration levels.

This report provides information on interoperable data management, software stack and gives overview on possible use case scenarios, where reliable and timely air quality data would be essential. It is part of a series of JRC technical reports which address in an integrated manner different aspects associated with the implementation of the AirSensEUR platform. The reports include:

- Part A: Sensor Shield
- Part B: host, influx datapush and assembling of AirSensEUR
- Part C: Interoperable Data management
- Part D: Calibration of data

Particular attention in writing the report is put on reusability and reproducibility of the results and the opportunities for establishment of a community of practice around the AirSensEUR platform. The work done is an integral part of the AirSensEUR Proof of Concept JRC Work Package, being implemented by the Digital Economy and Air and Climate units of the JRC. The work package is put in place with the overall idea to transform the currently existing AirSensEUR prototype in an operational product ready for implementation by EU Member States. Furthermore, the proposed architecture and software components are generic, and can be reused in different domains where spatio-temporal observation data is involved (e.g. precision farming and/or smart cities).

Structurally the report is sub-divided into four sections. The AirSensEUR architecture is described in Section 2, together with all individual components (sensor shield, sensor host and AirSensEUR server). Section 3 starts with an overview of the required software and then goes into substantial detail on the server component of the platform. All necessary configuration steps (both on the sensor host and the server) for an independent implementation are described. The consequent section (4) gives detail on the web and

desktop clients, which can be used to consume data from AirSensEUR. Particular emphasis, provided the need to calibrate air quality observation data, is put on the configuration and use of the "R" statistical package.

2. AirSensEUR architecture

2.1 General overview

AirSensEUR is designed as an open platform based on several pillars, which ensure that individual sensor nodes are capable of interoperating with heterogeneous sources of data. The high level objective, which determines the bounding conditions of AirSensEUR, is to design and build a platform which is simultaneously:

- capable under certain conditions of producing indicative observation data that meet the legal requirements of the EU Air Quality Directive (2008/50/EC) and it's Implementing Provissions with regards the reciprocal exchange of information and reporting on ambient air quality (2011/850/EU), and
- implements a download service, as required by the EU INSPIRE Directive (2007/2/EC)

From a technical point of view, the platform consists of a bundle of software and hardware (Figure 1), which are configured to work together in a synchronized manner. The hardware (Subsystem A) consists of a sensor shield (A1) and host (A2). They are further described in Parts A and B of the AirSensEUR JRC Report. This report puts more emphasis on the software, both in terms of backend (Subsystem B) and client applications (Subsystem C). Further information about the platform is also provided on www.airsenseur.org.

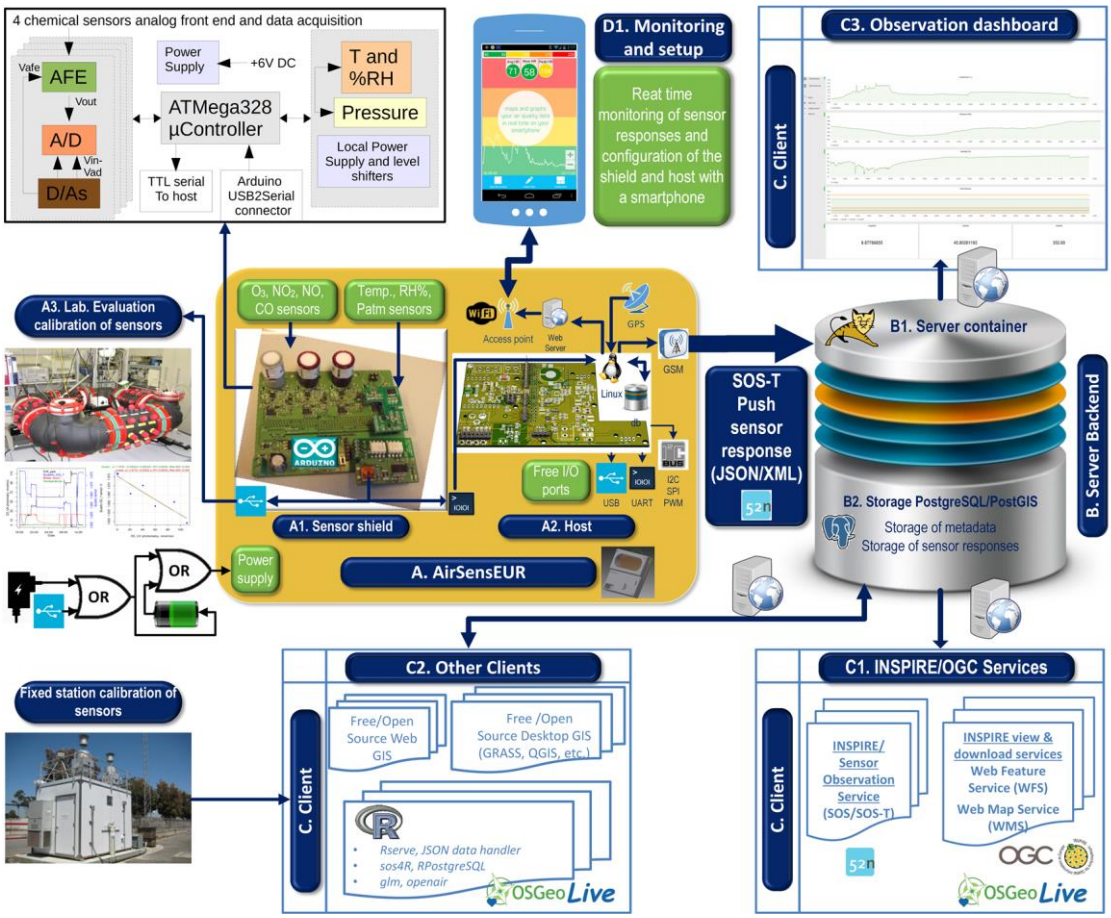

Figure 1. AirSensEUR Architecture

2.2 Hardware Components

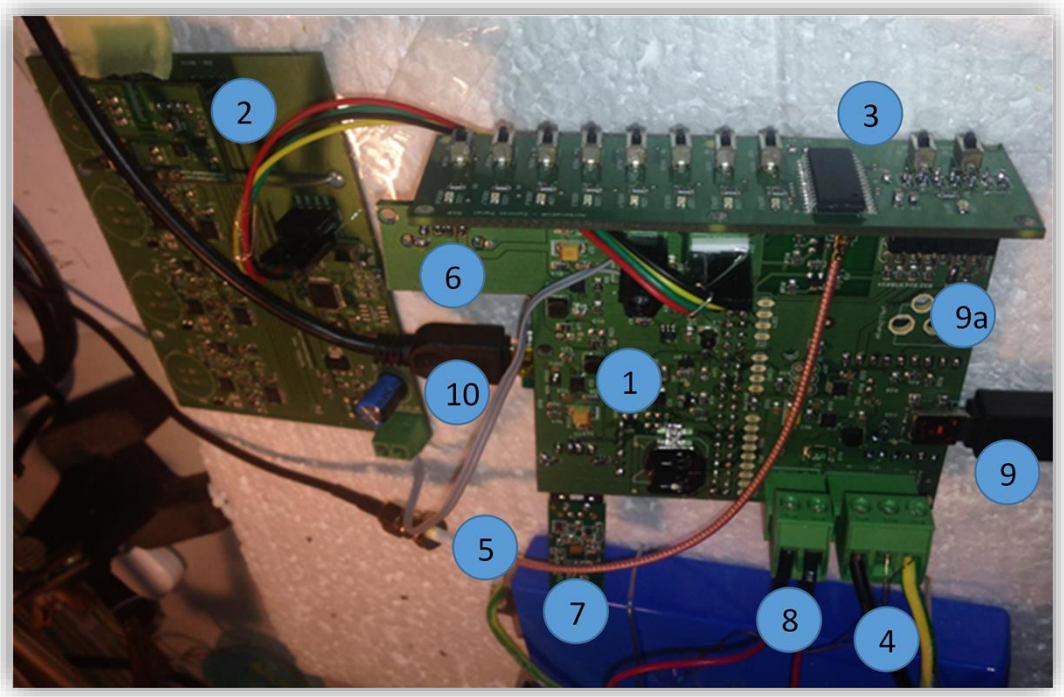

Figure 2. AirSensEUR hardware components

Table 1. Overview of AirSensEUR components

Component	
1. Host with CPU at the back side	2. Sensor shield
3. Control panel	4. Battery
5. GPS Antenna	6. USB for GSM/GPS key
7. USB WiFi key	8. On/Off switch
9. USB power supply and battery recharging	9a. Wall power supply (220 V)
10. USB to Linux console to control CPU	

2.2.1 Hardware

In terms of hardware, the platform (Figure 1) consists of a multi-sensor shield (A1), which is connected to a Linux-based host (A2). The individual components of AirSensEUR are shown in Figure 2 and described in further detail within Table 1. AirSensEUR documentation, together with computer-aided designs of boxing for 3D printing, is open by design, thus ensuring the ability to reproduce and reuse the results. All resources are made available at www.airsenseur.org, and through parts A and B of this JRC technical report.

2.2.2 Software

AirSensEUR uses open source software in order to take advantage of the rapid development cycle and outreach to existing communities. The server side component of the platform is by design based on OSGEO-Live - the free and open source bundle of the Open Source Geospatial Foundation (https://live.osgeo.org). This provides multiple

opportunities, as data from AirSensEUR can be used within both web and desktop Geographic Information System (GIS) clients. Furthermore, through using OSGEO-Live as the environment for handling data, we ensure that the open source projects that we use are supported by community of developers and users and meet baseline quality criteria (Brovelli, 2012). The components that are chained together in AirSensEUR are provided in

Table 2. Software products used by AirSensEUR

Table 2. Software products used by AirSensEUR

Functionality	Product(s)	Overview
1. Web transactions	AirSensEUR SOS-T client	• Java application, pushing data (JSON POST transactions) from the host to a server when an Internet connection is available.
	sqlite3	• Local data storage on the sensor host.
2. Storage	PostgreSQL/PostGIS	• Server-side storage, with a database schema suitable for the 52◦North52◦North SOS implementation
		• RESTful interface on top of the SOS web service
3. Web services	52 North SOS	• Mobile-friendly web client for interaction with observation data
	52◦North52◦North TimeSeries API	• Mash-up with other geospatial data and implementation of INSPIRE discovery and view services
4. Clients	52◦North Helgoland client	• JavaScript SOS client with functionality to process and analyze air quality data with R
	RStudio (including Shiny and sensorweby)	• JavaScript SOS client with functionality to process and analyze air quality data with R
5. Visualisation and data processing	R	• Post-processing of data (e.g., for calibration or further statistical analysis)

2.2.3 Applicable standards

2.2.3.1 Sensor Web Enablement (SWE)

Sensor Web Enablement (SWE) is a bundle of standards, created by the Open Geospatial Consortium (OGC), that are explicitly dedicated to sensors. Individual standards in SWE are by design meant to work together in utilizing spatio-temporal observation data. Additional information on the individual standards is provided by Bröring et al. (2011).SWE includes the following interdependent standards:

- Sensor Observation Service (SOS)
- Observations and Measurements (O&M)
- Sensor Event Service (SES)
- Sensor Processing Service (SPS)

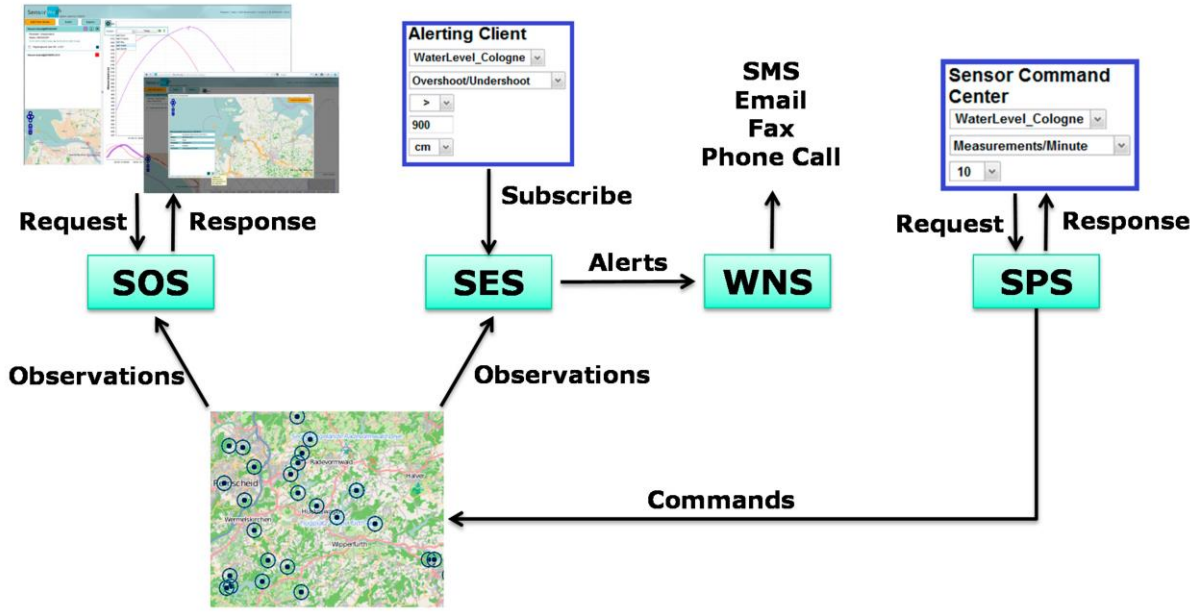

Figure 3. SWE components. Source: Kotsev et al. (2015)

The **Sensor Observation Service (SOS)** is created for serving spatio-temporal data through a standardised web-based interface. Data are encoded through the **Observations and Measurements (O&M)** ISO/OGC standard. O&M models

The SOS standard follows the reques/response paradigm. The current version (2.0) has adopted a modular structure (Figure 4) that fits a large number of use cases. AirSensEUR takes advantage of most of the SOS modules, incl. the transactional extension (SOS-T) which is used for data pushing. The operations of the standard are mapped to the legal requirements of the INSPIRE Directive by Broring et al (2013). A technical guidance document for implementing an INSPIRE download service through SOS is also made available.

A limitation of the SOS standard is related to (i) the lack of a lightweight JSON binding, and (ii) support of a RESTful architecture. This is overcome by an extension of the 52North SOS implementation which is used by AirSensEUR (see sec. 3.2.1 for further details).

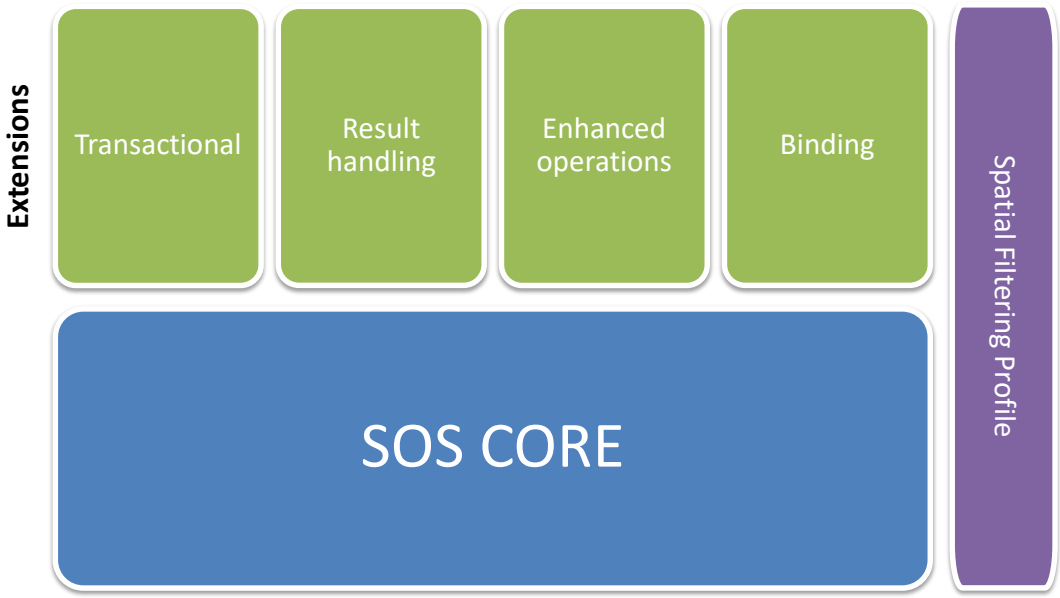

Figure 4. Structure of OGC "Sensor Observation Service 2.0"

2.2.3.1 INSPIRE

The Infrastructure for Spatial Information in the European Community (INSPIRE) aims to create a pan-European spatial data infrastructure (SDI) for the purposes of EU environmental policies, and policies that may have an impact on the environment. This European SDI (i) enables the sharing of environmental spatial information among public sector organisations, (ii) facilitates public access to spatial information across Europe, and (iii) assists in policy-making across boundaries and domains.

INSPIRE is not a standalone activity. It is based on the infrastructures for spatial information established and operated by the Member States of the European Union. The Directive addresses 34 spatial data themes (Figure 5) needed for environmental applications. The Directive came into force on 15 May 2007 and will be implemented in various stages, with full implementation required by 2021.

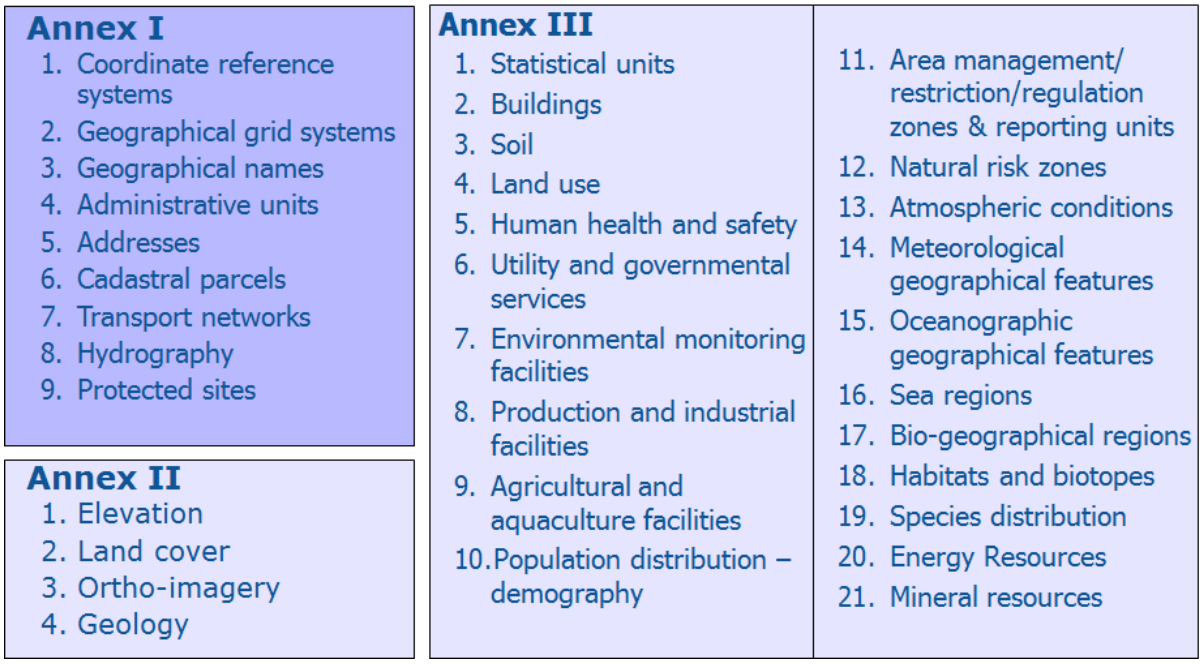

Figure 5. INSPIRE: Thematic scope

European Union providers that hold data within the scope of the Directive (Figure 5. INSPIRE: Thematic scope) are obliged to produce metadata, implement network services and encode their data in accordance with a set of commonly agreed data models. In terms of network services the Directive distinguished between several types of web-based interfaces that treat the discovery (metadata exposure), visualisation, download and transformation of the data (Figure 6. INSPIRE Architecture).

The AirSensEUR platform implements an INSPIRE download service through the use of SOS.

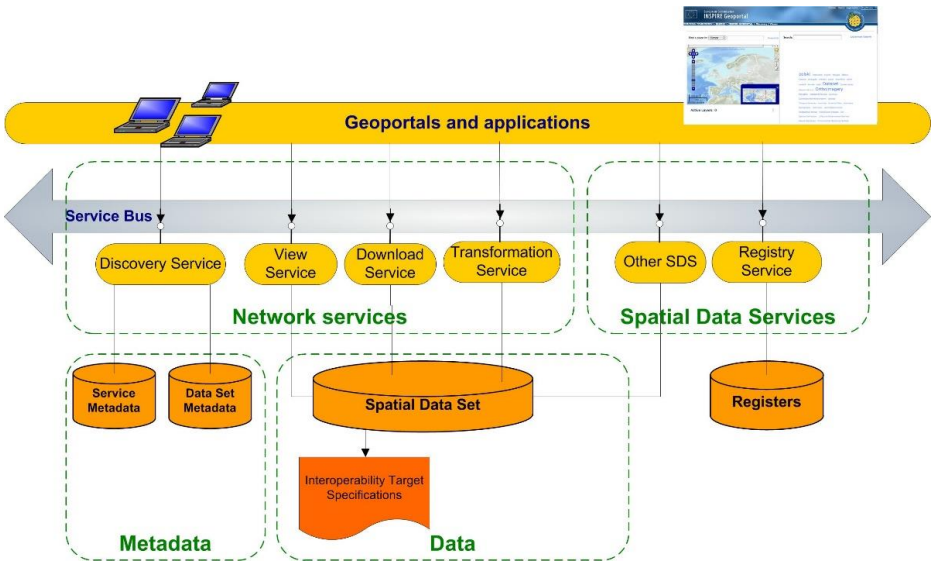

Figure 6. INSPIRE Architecture

INSPIRE is, on a conceptual level, based on a number of common principles:

- Data should be collected only once and kept where it can be maintained most effectively.
- It should be possible to combine seamless spatial information from different sources across Europe and share it with many users and applications.
- It should be possible for information collected at one level/scale to be shared with all levels/scales; detailed for thorough investigations, general for strategic purposes.
- Geographic information needed for good governance at all levels should be readily and transparently available.
- Easy to find what geographic information is available, how it can be used to meet a particular need, and under which conditions it can be acquired and used.

3. Server implementation

3.1 Required software

3.1.1 Operating system

A running Linux server is a precondition for the implementation of the AirSensEUR server components. Ideally, superuser rights should be available for the server where AirSensEUR should be implemented.

For simplicity we propose the use of the OSGEO-live virtual drive (*.vmdk)[1], where the majority of necessary tools (as described in Table 2) are already installed, thus only requiring minor configuration work. The OSGEO-live virtual drive can be used through various virtual machine applications, such as VirtualBox, VMWare player, KVM or XENServer. If a standalone Linux distribution is chosen instead we recommend Ubuntu, but other distributions would be suitable as well.

Alternatively, a docker image with the packages described below might be built through the use of a Dockerfil.

3.1.2 Specialised software

The following minimum set of software components are a precondition for the successful implementation of an AirSensEUR server:

Table 3. Required server-side software

Tool	Version	Function	Comment
Apache Tomcat[2]	6.0 or higher	Servlet container	other servlet containers are also possible
PostgreSQL[3]	9.0 or higher	Data storage	Other RDBMS are also possible with additional configuration of the 52° North SOS.
52°North SOS[4]	4.0 or higher	Implementation of the INSPIRE-compliant OGC Sensor Observation Service (SOS)	

3.2 Configuration

3.2.1 AirSensEUR Server

The following steps, described in further detail below, need to be accomplished in a consecutive manner in order for a server to be properly configured for AirSensEUR.

A) SOS server configuration

This technical report assumes that a 52°North SOS implementation (52N SOS) is already deployed and the default database (PostgreSQL/PostGIS) is installed and configured. If that is not the case please refer to the following manual (https://wiki.52north.org/bin/view/SensorWeb/SensorObservationServiceIVDocumentation#Installation). After successfully starting the 52N SOS in your preferred servlet container (Tomcat, jetty, etc.), navigate to the login page of the 52N SOS, which is available at http://localhost:8080/52nSOS/login (Figure 7). The default credentials for the pre-installed OSGeo-Live version of the software are `user/user`.

[1] Virtual drive and/or bootable ISO images can be acquired at: https://sourceforge.net/projects/osgeo-live/files/9.5/
[2] http://tomcat.apache.org/
[3] https://www.postgresql.org/
[4] http://52north.org/communities/sensorweb/sos/

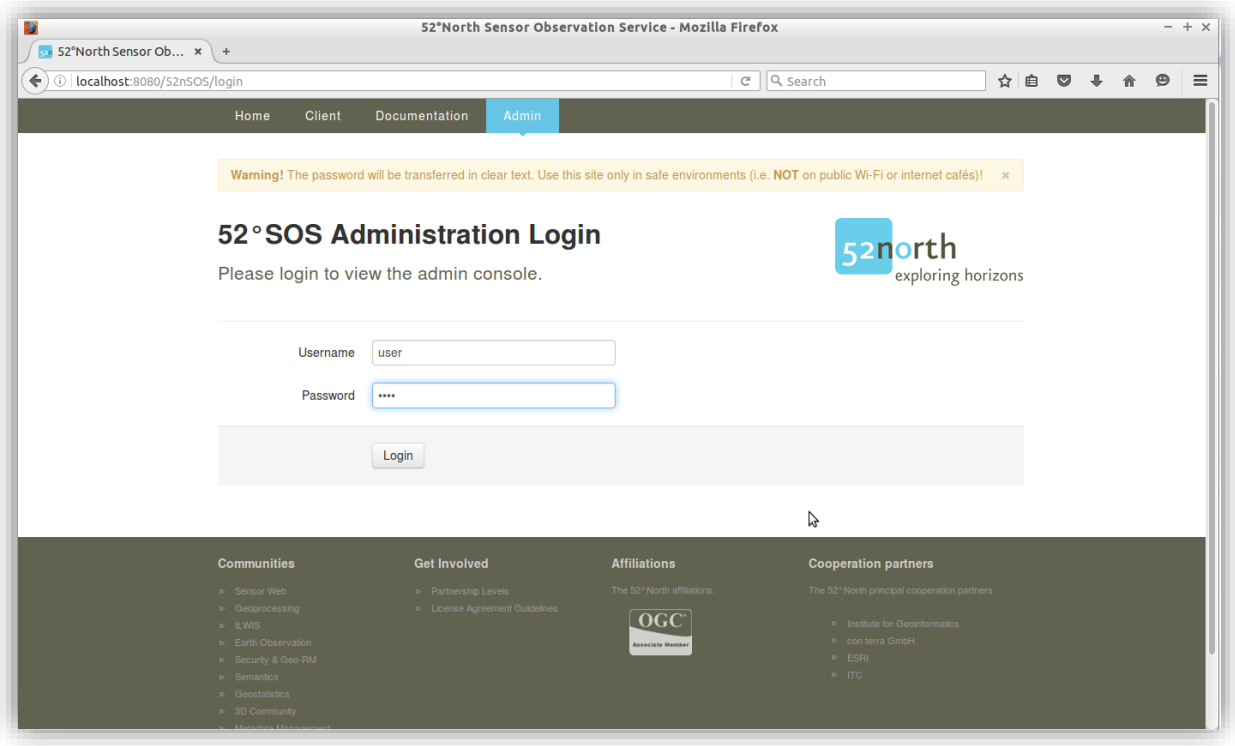

Figure 7: Administration Interface of the 52°North SOS Server

The administrative interface of the application allows users to enter generic information regarding the metadata of the SOS server, as well as mandatory information required by INSPIRE.

For allowing the publication of new observation data, it is important to ensure that the InsertSensor and InsertObservation operations are enabled via the server administration interface (see Figure 8). This menu is accessible via "Admin" → "Settings" → "Operations".

Besides this necessary setting, the user may adjust further settings of the SOS server via the Administration interface (please note: this overview only contains a selection of settings which may be relevant):

- "Admin" → "Settings"
  - Service Provider: change the provider information (Capabilities) of the server
  - Service Identification: change the identification information (Capabilities) of the server
- "Admin" → "Settings" → "Transactional Security": If you server is accessible via the Web, you should consider to limit the access to write operations via this setting (for details please see[5])
- "Admin" → "Settings" → "Streaming": Enable/disable XML response and/or data source streaming in order to improve the SOS performance in case you are dealing with large XML documents as SOS outputs
- "Admin" → "Settings" → "I18N:" Multilingualism definition so that you can provide translations of certain terms that shall be used by the SOS server
- "Admin" → "Settings" → "INSPIRE:" Define metadata for SOS as INSPIRE download service
- "Admin" → "Settings" → "Logging": Change the log level or show the latest log statements

5 https://wiki.52north.org/SensorWeb/SensorObservationServiceIVDocumentation#Transactional_Security

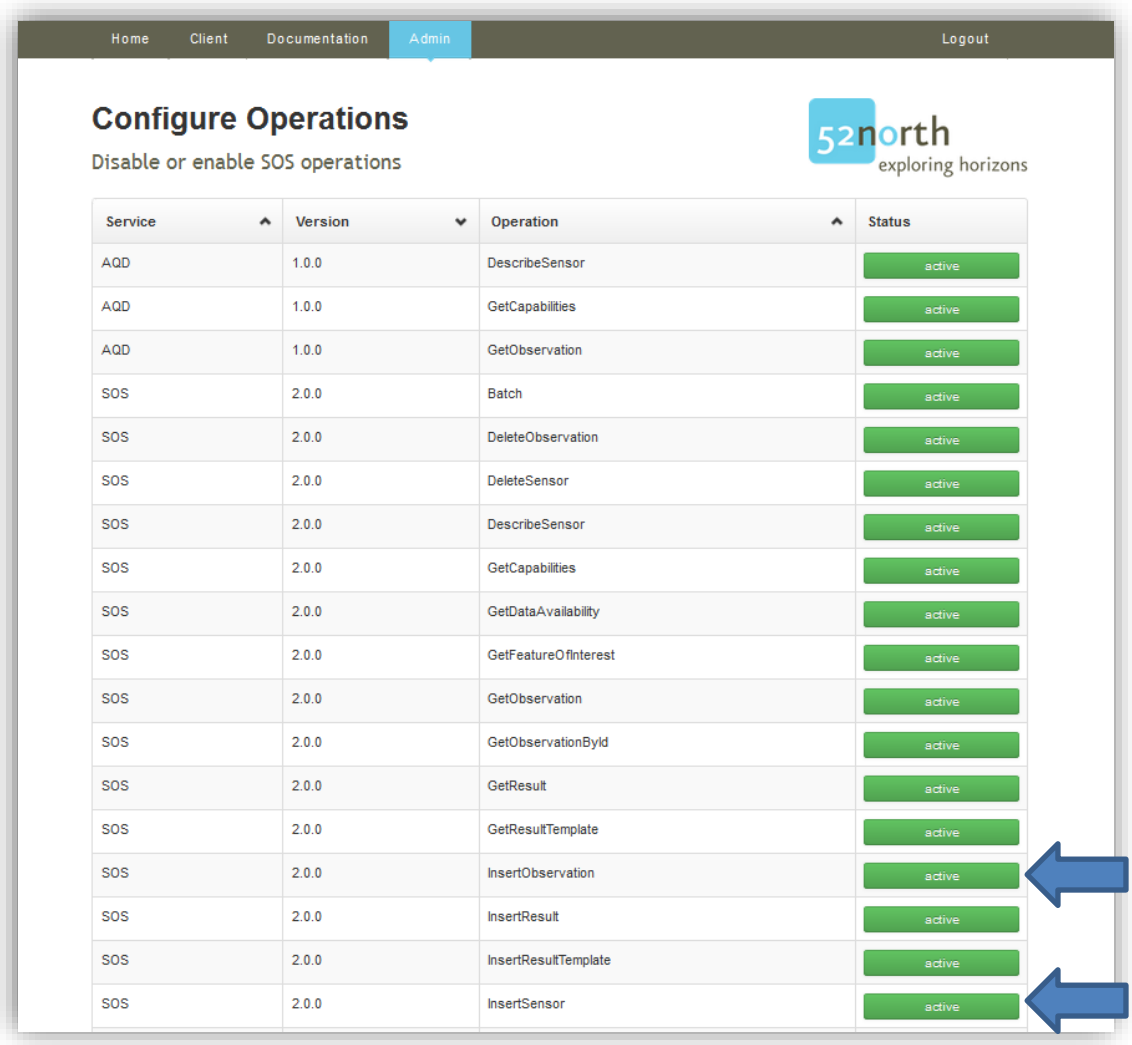

Figure 8: Administration Interface: Enabling the InsertSensor and InsertObservation operations (blue arrows)

After the installation of the SOS has been completed, the transactional SOS operations have to be executed in order to populate the SOS server with data.

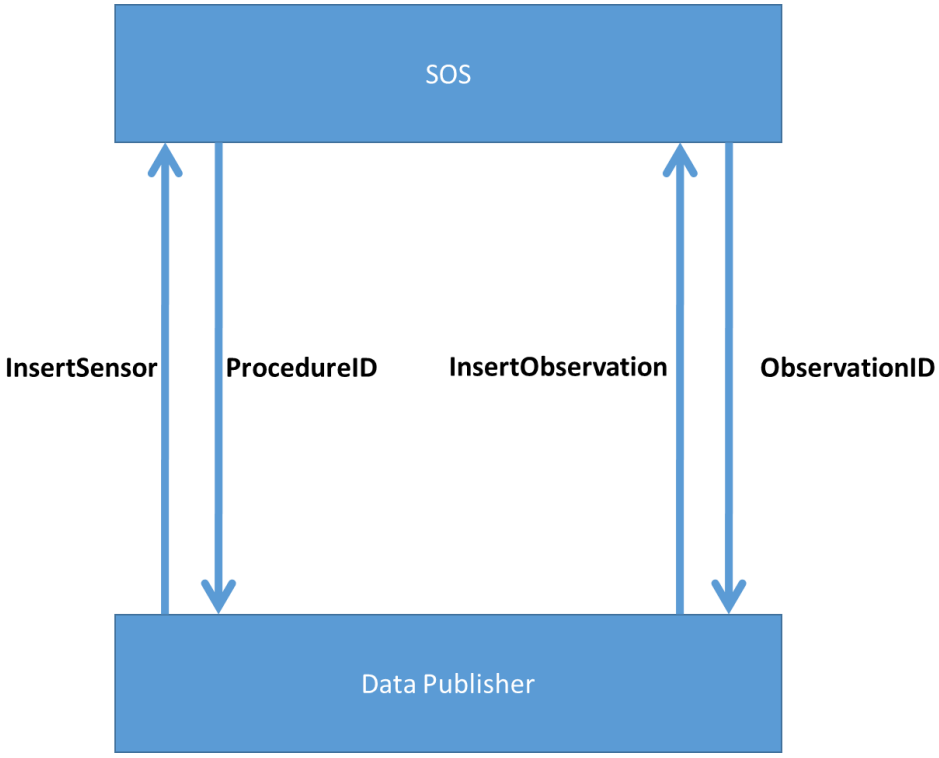

Figure 9: Using the Transactional SOS Operations for Data Loading

Figure 9 shows the workflow for data publication. In a first step the InsertSensor operation is called. This operation allows to register new sensors in the SOS server. The response to this operation call contains the ID which has been assigned by the SOS to the registered sensor. This ID allows the SOS server to associate inserted observation data to the corresponding sensors. Thus, in subsequent steps, it is possible to call repeatedly the InsertObservation operation for uploading observation data to the SOS server. This call contains besides the observation data itself also the ID of the sensor which has generated these observations.

While the AirSensEUR platform already comes with working transactional operation requests that are automatically executed, users may rely on the examples delivered with the SOS to add further data to the server Example requests illustrating the usage of the transactional operations are available here[6].

B) InsertSensor operation

Sample InsertSensor requests to the 52North SOS Server are made available at: https://webgate.ec.europa.eu/CITnet/stash/projects/AIRSENSEUR/repos/airsenseur-server.

3.2.2 Sensor host

The AirSensEUR platform is a work-in-progress. Within this context, a lot of configuration options are available through configuration files that can be modified through an SSH connection. AirSensEUR runs over a standard Debian Wheezy distribution and all AirSensEUR related configuration files and script are stored in the microSD as shown in the following table.

6 https://github.com/52North/SOS/tree/develop/webapp/src/main/webapp/static/examples/sos_v20/requests_xml/Transactional

Table 4. AirSensEUR file paths

Function / Subsystem	Path
AirSensEUR services configuration	/usr/local/etc
AirSensEUR scripts and JAVA binaries	/usr/local/airsenseur
AirSensEUR Logs	/var/log

The most important processes and services configuration files are reported in the table below.[7]

Table 5. Processes and service configuration file locations

Process / Function	File
AirSensEURHost (read Shield and publish data)	/usr/local/etc/sensor.properties
AirSensEURDataAggregator (aggregate data from GPS, Shield and store in a local database)	/usr/local/etc/aggregator.properties
AirSensEURDataPush (read data from local database and pushes to external servers)	/usr/local/etc/influxpush.properties (InfluxDB mode) /usr/loca/etc/datapushsosdb.properties (52North SOS mode)
Data Push main script (called by a cron job)	/urs/local/airsenseur/runDataPush

The AirSensEURDataPush is the application, on the AirSensEUR host, taking care of pushing collected data to an external server. Two operating modes are provided: Influx and SOS52North.

The process is started up/down through a specific script periodically called by a cron job. The AirSensEURDataPush loads data from the sqlite database generated by the AirSensEURDataAggregator.

The main operating parameters are stored within a set of configuration files locatedlocated in /etc/local/etc folder (there are two different configuration files, one for each operating mode) for easy customization.

[7] For more information on that, please refer to www.airsenseur.org available documentation.

4 Clients

The client support for the SWE standards is broad, as the output of the requests can be consumed by most GIS desktop and web applications. There is also a growing number of SWE-specific clients, such as the 52°North JavaScript SOS Client Helgoland, being prepared especially for observation data. The consequtive two sections provide overview of both Web and desktop clients which are able to consume data from AirSensEUR.

4.1 Smartphone app

A smartphone app was developed within the MYGEOSS project[8] to consume data from the AirSensEUR platform (Figure 10). The app (SenseEurAir) is available for both Android (4.4 and above) and iOS.

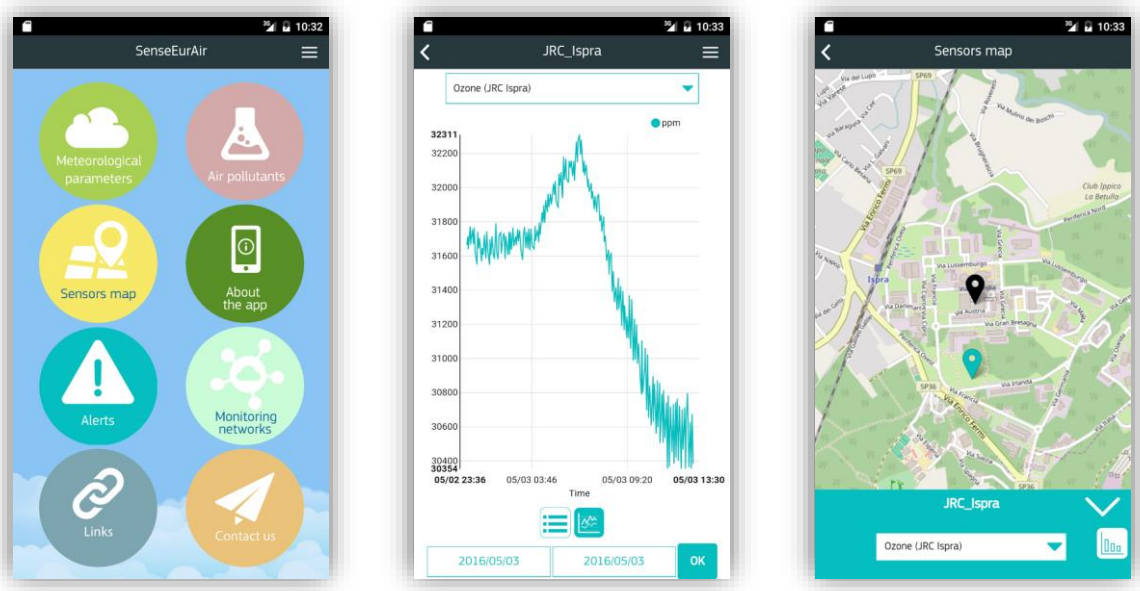

Figure 10. SenseEurAir user interface

The SenseEurAir functionalities include:

- Visualisation of the spatio-temporal distribution of pollutants
- Mash-up with data from official networks
- Notifications in case of pollution episodes

4.2 Web clients

4.2.1 52°North Helgoland

Helgoland is a lightweight Web application, which enables the exploration, analysis and visualization of sensor web data in a range of application domains such as hydrology, meteorology, environmental monitoring, and traffic management. It offers core functionality such as:

- exploring stations or mobile sensor platforms in a map (including the visualisation of the latest measurements at selected stations)

[8] http://digitalearthlab.jrc.ec.europa.eu/activities/mygeoss-applications-your-environment/57752

- data selection based and map view as well as lists of available resources
- visualize time series data as diagrams
- table view
- visualisation of trajectory data
- create favourites of selected time series
- download of selected time series (e.g. CSV files)

The application is based on HTML, JavaScript and CSS and can connect to different Sensor Web endpoints (REST-APIs). These Sensor Web REST-APIs provide a thin access layer to sensor data via RESTful Web binding with different output formats. As shown in Figure 11, Helgoland can either directly connect to a REST-API offered by a 52°North SOS server or to native SOS interfaces through an intermediary proxy component which caches relevant metadata of the SOS servers and offers a REST-API endpoint (currently this proxy support stationary in-situ measurements; an extension for mobile sensors is currently in development).

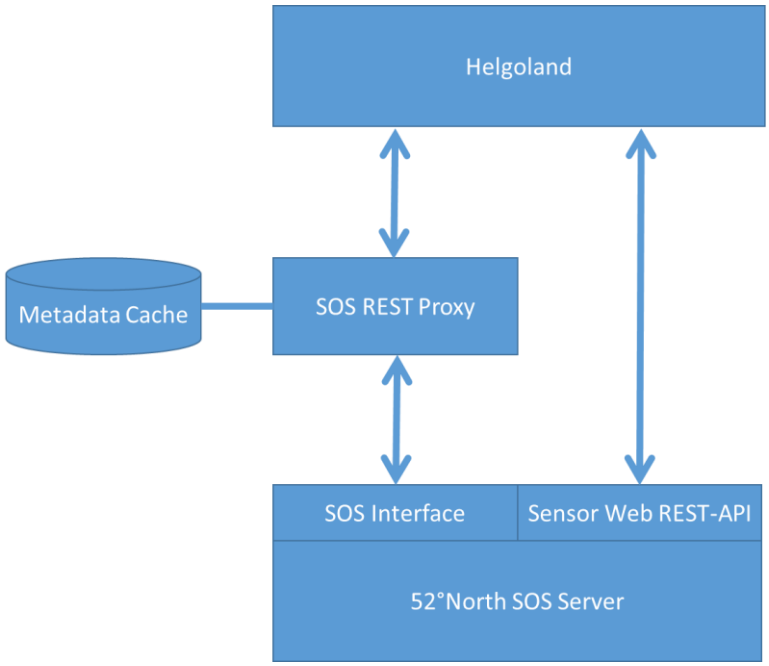

Figure 11. Helgoland Architecture

Helgoland is based on a set of JavaScript frameworks including:

- AngularJS[9]
- Bootstrap[10]
- Leaflet[11]
- Moment.js[12]
- flot[13]

[9] https://angularjs.org/
[10] http://getbootstrap.com/
[11] http://leafletjs.com/
[12] http://momentjs.com/
[13] http://www.flotcharts.org/

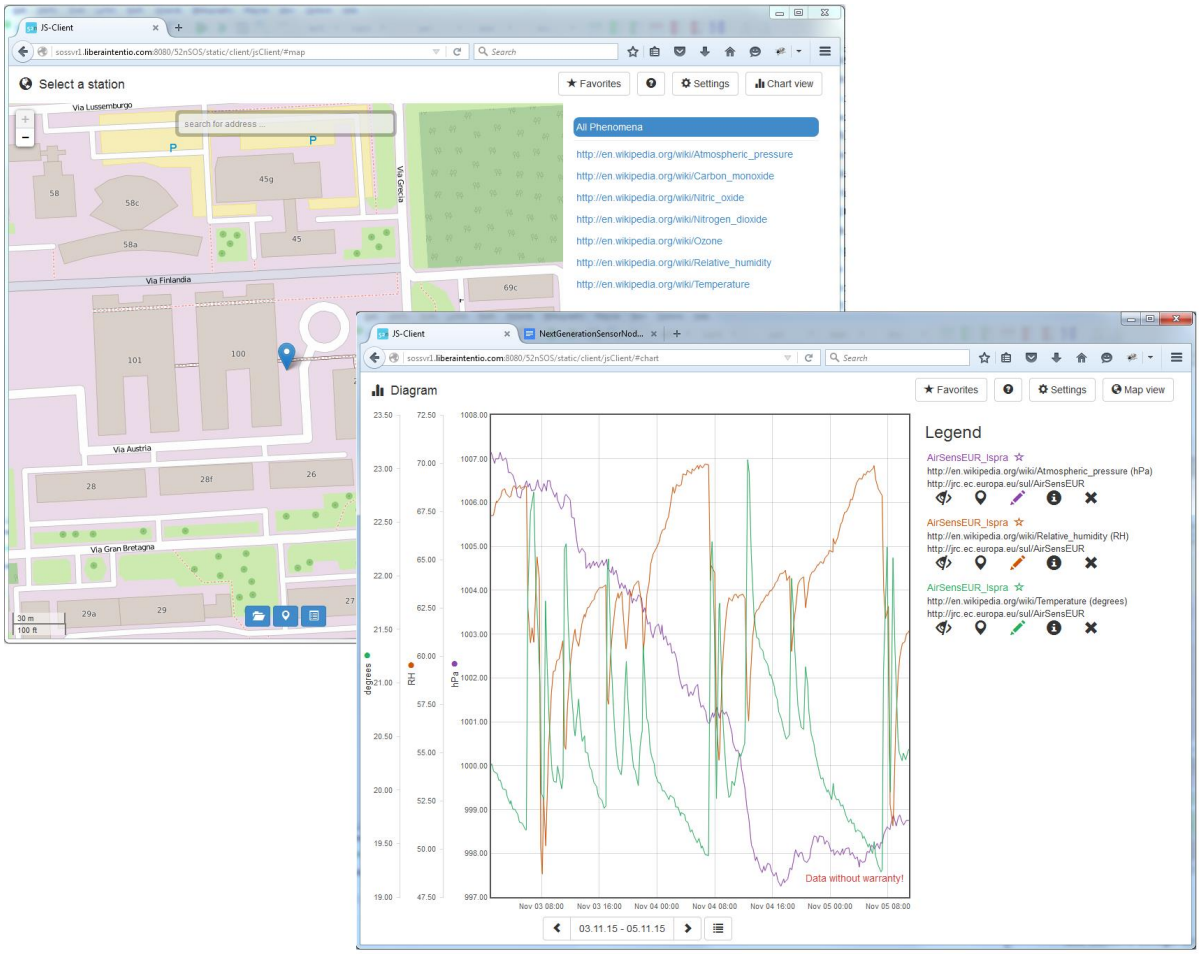

Figure 12. Screenshots of the 52°North Sensor Web Client Helgoland

For using Helgoland, users may rely on the pre-installed version on the OSGeo Live DVD. For this purpose, the URL of the SOS server has to be extended with "[SOS-URL]/static/client/jsClient/". This will open an instance of Helgoland which is pre-configured with the SOS server running on the OSGeo Live DVD.

Alternatively, users may download Helgoland as a war file from 52°North. After the download it is necessary to deploy the war file in a Web container such as Apache Tomcat or as a static web page in a web server (e.g. Apache). In addition, Helgoland needs to be configured to display the desired Sensor Web endpoints. For this purpose, the settings.json in the root folder of Helgoland can be adjusted. The main parameters are:

- defaultProvider: this is the default selected provider, when the user starts the client
- restApiUrls: this is a list of all Sensor Web REST-API instances that shall be used by the client

After this configuration, Helgoland will be able to visualise the content of all configured endpoints.

4.3 Desktop clients

4.3.1 sos4R and sensorweb4R

sos4R is a R package which enables the access to OGC Sensor Observation Service instances from R (Nüst et al., 2011). The package supports on the one hand a simple encapsulation and abstraction from the service interface for novice users as well as the creation of more comprehensive requests for specialists.

sos4R is intended to close the gap between the Sensor Web as data source (more specifically SOS servers) and tools for (geo-)statistical analyses. It implements the core

operations of the SOS specication and supports temporal, spatial, and thematical filtering of observations.

The package is published under the GPL 2 license within the geostatistics community of 52°North. It is available on CRAN: https://cran.r-project.org/package=sos4R

In detail, sos4R supports the following SOS functionality:

- Supported SOS operations:
  - GetCapabilities
  - DescribeSensor
  - GetObservation
  - GetObservationById
- Supported version of the SOS standard: 1.0.0
- Supported connection methods:
  - HTTP GET
  - HTTP POST
- Supported spatial operators:
  - BBOX
  - Contains
  - Intersects
  - Overlaps
- Supported temporal operators:
  - After
  - Before
  - During
  - Equals

Complementary to sos4R, 52°North has also developed sensorweb4R[14]. This library offers similar functionality like sos4R but enables the usage of the REST-API deployed together with the 52°North SOS instead of the pure OGC compliant SOS requests. Depending on the use case, sensorweb4R may facilitate the discovery of certain observations compared to harvesting and analysing the rather complex SOS metadata.

[14] https://github.com/52North/sensorweb4R

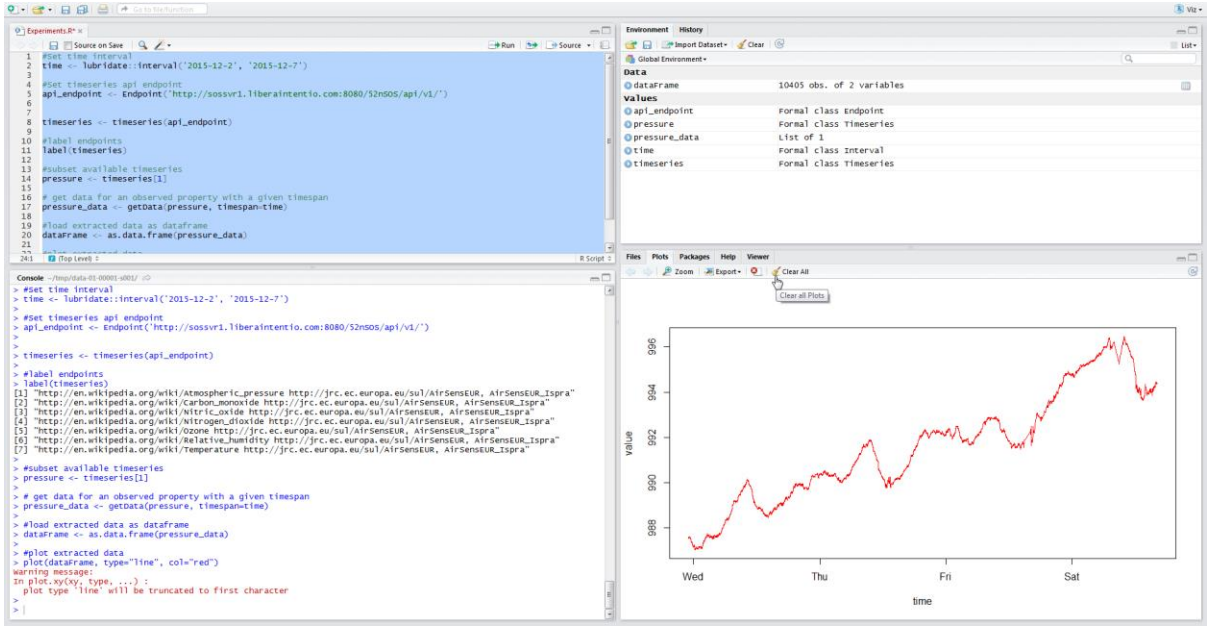

Figure 13. Data from AirSensEUR pulled in "R"

The following example illustrates the usage of sensorweb4R. The code shown in the listing below results in the diagram representation shown in Figure 13.

```
# Install libraries
install.packages("devtools")
devtools::install_github("52North/sensorweb4R")

#Set time interval for which data shall be requested (2nd to 7th of December 2015)
time <- lubridate::interval('2015-12-2', '2015-12-7')

#Set Sensor Web REST-Api endpoint (URL of the REST API)
api_endpoint <- Endpoint('http://sossvr1.liberaintentio.com:8080/52nSOS/api/v1/')

#Access the time series available at the endpoint
timeseries <- timeseries(api_endpoint)

#Label the timeseries
label(timeseries)

#Subset the available timeseries (select a timeseries with pressure data)
pressure <- timeseries[1]

# Get data for an observed property with a given timespan
pressure_data <- getData(pressure, timespan=time)

#Load extracted data as dataframe
dataFrame <- as.data.frame(pressure_data)

#Plot extracted data
plot(dataFrame, type="line", col="red")
```

4.4 Code for downloading and plotting AirSensEUR data

A script has been written in "R" to automate the downloading of data. It uses a main script called AS-Script.R, a file with a collection of functions called Functions4ASE.R and a config file called ASEConfig.R. The file of functions should never be modified. The main script is slightly amended to give the disk and directory of the scripts. Conversely, the config file is modifified to adapt to each AirSensEUR node. All scripts are made available within the AirSensEUR GIT repository at https://webgate.ec.europa.eu/CITnet/stash/projects/AIRSENSEUR/repos/rcode/browse.

Here below the list of configuration updates to be performed is given in bold font, together with the main actions carried out in the scripts:

ASEConfig_xx.R

1. Create a file system structure, starting from the main directory set in ASE?script.R. The directory structure is created under the main directory .. with the following sub directories: Calibration, Drift, Estimated_coef, General_data, Models, Modelled_gas, scriptsLog, SensorDataRetrieved_plots, Statistics, Verification_plots. A log file of all operations (console_yyyy_mm_dd.log) can be found in ..\ASEconfig_01\scriptsLog. It contians all output send to the console using "R".
2. Loading of Functions4ASE.R. the file Functions4AES.R is sourced. It is checked whether the file exists in the main directory.
3. Configuring Proxy server: **set** the proxy (if any). If a proxy server is not needed **set PROXY to FALSE.**
4. Install packages, needed in the script (CRAN + Github): the packages "plyr", "openair", "zoo", "futile.options", "lambda.r", "curl", "sp" , "httr" on CRAN and 52North/sensorweb4R on github are loaded.
5. Sensor configuration for download from and SOS server. **Set** Down.SOS to TRUE, **enter** the SOS URL and AirnSensEUR name for which data are to be downlaoded. Set time sone: sens.tzone <- "UTC" **add** any pollutant not present in List_Pollutant fow whcih data should be downloaded from the SOS server
6. Reference data, configuration for download, ftp (NOT INCLUDED)
7. Create the sensor configuration file and matching between reference and sensor names if needed.

   To perform an outlier detection procedure using the Median Average Deviation method[15], **set** sens2ref$Sensor.rm.Out to TRUE if you want to discard outliers from sensor values, and **set** values for: sens2ref$Sens.window (rolling window of continuous data on which outliers are flagged, for 24 hours of 10-min values: 24 * 6 = 144), sens2ref$Sens.threshold (coefficent that multiplied by the difference between mean and median of the rolling window will flag give outliers if exceeded), sens2ref$Sens.Ymin (Minimum digitial values in data series under which data are flagged as outliers), sens2ref$Sens.Ymax (maximum digitial values in data series under which data are flagged as outliers), sens2ref$Sens.ThresholdMin (minimum acceptable values for values – sens2ref$Sens.threshold x (mean - median)).

   Set the characteristic volt values equal to the values set in JAVAControl Panel[16] for each sensor for sens2ref$Sensor.raw.unit generally in "volt", sens2ref$Ref,

[15] ISO 13528:2015: Statistical methods for use in proficiency testing by interlaboratory comparison, Geneva, Switzerland.

[16] Michel Gerboles, Laurent Spinelle and Marco Signorini, AirSensEUR: an open data/software /hardware multi-sensor platform for air quality monitoring. Part A: sensor shield, Luxembourg: Publications Office of the European Union, 2015, EUR 27469 EN, EUR – Scientific and Technical Research series – ISSN 1831-9424 (online), ISBN 978-92-79-51896-6 (PDF), doi: http://dx.doi.org/10.5162/4EuNetAir2015/03

sens2ref$RefAD, sens2ref$board.zero.set, sens2ref$Intercept and sens2ref$Slope to the sensor sensitivity in V/ppb and volage offset. The Sensitivity of CO that is normally given in V/ppm can be transformed in V/ppb for homogeneity with the other sensors to get the same unit), sens2ref$Sensor.Unit ater calibration with sens2ref$Intercept , sens2ref$Slope (genrally ppb).

set sens1 to sens4 a list of the parameters you want to plot for each sensor

8. Average time for sensor data. **Set** Usermins to the periodicity of the reference values. Minimum values 1 (min), default value 10 (mins)
9. temperature and relative humidity thresholds for sensors validity. **Set** the min and max values of temperature and relative humidity for each sensor. Sensor data out of the interval of tolerance are discarded (set to NA)
10. Calibration with Etalonnage
11. Valid Periods (NOT USED)

Set Delay, the delay in minutes (positive and negative as the values are added) of sensor data compared to reference data if available, default values should be 0.

12. SET TIME PARAMETERS -> see in ASE_OPER_SCRIPT.R (NOT USED)
13. SET Models for NO2 and O3 system resolution (NOT USED)

ASE_Script.R

1. Line 46: **enter** the disk name where the scripts and data are located using the syntax of file.path function. Make sure that the directory exists. (variable Disque created) in List.Disque.
2. Line 64: **enter** the exact Directory where the script ASE_script is located (variable DisqueFieldtest created) in List.DisqueFieldtest.
3. Line 74: **enter** ASEConfig, the name of the config file of the AIrSensEUR node that shall be in the same directory as ASE_Script.R and Functions4ASE.R. The script will stop if the file does not exist with an error message

The script is run by sourcing the ASE_script.R in rstudio[17] once the 3 update in bold font given above are performed. Here is a list of the output of the code:

1. Lines 104, the function Down_SOS is run to download data from the SOS server. It returns a dataframe named SensData incuding all data of the AirSensEUR node. Additionally, 2 files are savedd under ../General_data named SOSData.Rdata and SOSData.csv. If the process is stopped at any moment you can run it again for update starting in line 95.
2. Lines 149 to 237: the newly downloaded dated are appended to any previously existing data
3. Lines 242-248: data with NA and NaN only are discarded
4. Lines 250-257: plotting the whole raw dataset, saving plot in ../General_data, name AirSensEUR.name_Full_Time_Series_yyyy_mm_dd_yyyy_mm_dd.png with starting and ending date in the file name.
5. Lines 259-278: saving General.Rdata/.csv and AirsensEur.name.Rdata/.csv in ../General_data
6. Lines 280 -340: plotting only the newly downloaded data, saving file AirSenSEUR.name_yyyy_mm_dd_yyyy_mm_dd.png in ../ Verification_plots. The sensor data are also plotted together with the varaiable set in Sens1 to Sens4, see ASEConfig_xx.R above.
7. Lines 351-355: discarding sensor data for 12 hours after each warming (by changing line 360 from 12 to 24, 24 hours of data are discarded after warming)

[17] https://www.rstudio.com/ and https://cran.r-project.org/

8. Lines 357 -391: Discarding outliers in sensor data (1st pass). The plots are saved in ../General_data as AirsensEur.name_Outliers_nameSensors.png". Outliers are shown with coloured dots ().
9. Lines 393-435: **Set** iteration (default is 2), the number of times that outliers in sensor data are discarded with the same procedure. The plots are saved in ../General_data as AirsensEur.name_Outliers_nameSensors_1 or 2.png". Outliers are shown with coloured dots .
10. Lines 437 to 456: sensor values out of the interval of tolerances of temperature and humidity are discarded.
11. Lines 458-476: sensor dat are converted from digital values to volts
12. Lines 479-512: plotting times series for each sensor in digital values together with the parameters set in Sens1 to Sens4 in ASEConfig.R (see here above). The plot are save in ../Verfication_plots as AirSensEUR.name_yyyy_mm_dd_yyyy-mm_dd.png with the starting date and ending date. Digital values are plotted.
13. Lines 514-546: plotting times series of validated data in volts (outliers discarded) for each sensor together with the parameters set in Sens1 to Sens4 in CASEConfig.R (see here above). The plot are save in ../Verfication_plots as AirSensEUR.name_yyyy_mm_dd_yyyy-mm_dd.png with the starting date and ending date.
14. Lines 548-590: plotting times series of sensor values in ppb (calibration using slope and intercept of the sens2ref dataframe see ASEConfig-xx.R) for each sensor. The plot are save in ../Modelled_gas as AirSensEUR.name_ppb_yyyy_mm_dd_yyyy-mm_dd.png with the starting date and ending date.

Annex I: R scripts

```
#=================================================================
# Licence:
# Copyright 2017 EUROPEAN UNION
# Licensed under the EUPL, Version 1.2 or subsequent versions of the EUPL (the "License");
# You may not use this work except in compliance with the License.
# You may obtain a copy of the License at: http://ec.europa.eu/idabc/eupl
# Unless required by applicable law or agreed to in writing, the software distributed
# under the License is distributed on an "AS IS" basis, WITHOUT WARRANTIES OR CONDITIONS
# OF ANY KIND, either express or implied. See the License for the specific language
# governing permissions and limitations under the License.
# Date: 05/11/2017
#
# Authors
# - Michel Gerboles       , michel.gerboles@jrc.ec.europa.eu  - European Commission - Joint Research Centre
# - Laurent Spinelle      , laurent.spinelle@jrc.ec.europa.eu - European Commission - Joint Research Centre
# - Maria Gabriella Villani,                                  - ENEA
#=================================================================

#=================================================================
# Content
#=================================================================
#  0 - Clear memory and restart R-session
#  1 - Get configuration parameters in ASEconfig_MG.R - Create file system structure check for General.Rdata availbility,
#      create log file, decide if data need be retrieve
#  2 - Retrieve data if needed
#  3 - Linear Calibration

#=================================================================
# 0 START
#=================================================================
# Clear memory and restart R-session
remove(list=ls())

#=================================================================
#    - Setting Working Directory, get configuration parameters in ASEconfig.R
#    - Create file system structure check for General.Rdata availbility,
#    - create log file, decide if data need be retrieve
#=================================================================
cat("[main] INFO, Get configuration parameters")

#-----------------------------------------------------------------
# Selecting the disque directory, the 1st directory that exist will be selected
#-----------------------------------------------------------------
```

```
### Shiny input  Window 1 ###
# On which directory should be the data treatment. Maybe this step is not needed as it can be combined with the next one
List.Disque        <- c(    file.path("C:", "Users","gerbomi","Desktop"),
                            file.path("/","media", "sf_Box_Sync"), # virtual Box
                            file.path("s:", "Box Sync"),           # at office JRC
                            file.path("D:", "Profil_Michel", "Desktop", "Bureau","Diffusion", "Box Sync"), # at home
                            file.path("I:", "Bureau", "Diffusion", "Box Sync"), # USB disk
                            file.path("Volumes","My\ Passport\ for\ Mac","MGV_personal","00AirSensEUR_operativo","mybox-selected") # MAC
)
### END of Shiny input ###
for (i in List.Disque) {if (file.access(i) == 0) {
  Disque <- i; cat(paste0("[main] INFO, Disque found: ",i), sep ="\n"); break} else {
    cat(paste0("[main] INFO, Disque not found: ",i), sep ="\n")
  }
}

# checking if the directory does not exist
if(!dir.exists(Disque)) stop(paste0("[main] ERROR Dir ", Disque, "not found, stopping the process"))

#-----------------------------------------------------------------
## Directory for sensor Field tests script, create directory tree. The file ASEconfig_MG.R and Functions4ASE.R shall be in this Directory
#-----------------------------------------------------------------
### Shiny input  Window 1 ###
# Directory of the Data treament. The files Functions4ASE.R and ASEConfig_xx.R shall be included in the Directory
List.DisqueFieldtest     <- file.path(Disque,"AirSensEUR","Fieldtests", "ReportC")
### END of Shiny input ###
for (i in List.DisqueFieldtest) if (file.access(i) == 0) {DisqueFieldtest <- i; cat(paste0("[main] INFO, Field test dir found: ",DisqueFieldtest), sep = "\n"); break}
if(!dir.exists(DisqueFieldtest)) stop(paste0("[main] ERROR Dir ", DisqueFieldtest, "not found, stopping the process"))

#-----------------------------------------------------------------
## Setting the ASEConfig.R file
#-----------------------------------------------------------------
### Shiny input  Window 1 ###
# Name of the configuration file, the extension shall be .R
ASEConfig <- "ASEconfig_01.R"
### END of Shiny input ##
if(!file.exists(file.path(DisqueFieldtest,ASEConfig))){
  stop(paste0("[main] ERROR Dir ", file.path(DisqueFieldtest,ASEConfig), "not found, stopping the process"))
} else cat(paste0("[main] INFO, Config file found ", file.path(DisqueFieldtest,ASEConfig)))
# NOTE: DisqueFieldtest is the working directory

# Get configuration parameters
source(file.path(DisqueFieldtest, ASEConfig))

#================================================================
#  2 - Retrieve data if needed - Downloading Sensor data only SOS
```

```
#================================================================

cat("", sep="\n")
cat(paste0("[main] INFO, Starting downloading data for ", AirsensEur.name), sep="\n")
# Setting directory for savind sensor data
WDoutput = file.path(DisqueFieldtest, "General_data")

#----------------------------------------------------------
# Checking if Rdata file exists and setting the DownloadSensor$Retrieve.data.Ref and Retrieve.data.Sens true or false
# Checking if there are previously downloaded data in General.Rdata and settings DownloadSensor
#----------------------------------------------------------
DownloadSensor <- Check_Download(AirsensEur.name = AirsensEur.name, WDinput = file.path(DisqueFieldtestDir, "General_data"), UserMins = UserMins)

# SOSData
if (DownloadSensor$Retrieve.data.SOS){

  if(Down.SOS){

    #----------------------------------------------------------
    # Checking if Rdata file exists and setting the DownloadSensor$Retrieve.data.Ref and Retrieve.data.Sens true or false
    # Checking if there are previously downloaded data in General.Rdata and settings DownloadSensor
    #----------------------------------------------------------
    DownloadSensor <- Check_Download(AirsensEur.name = AirsensEur.name, WDinput = file.path(DisqueFieldtestDir, "General_data"), UserMins =
UserMins)
    # Check that your Proxy is correctly set before running the next command
    SOSData <- Down_SOS(AirsensEur.name = AirsensEur.name, UserMins = UserMins,
                        DownloadSensor = DownloadSensor, AirsensWeb = AirsensWeb, Duration = 1) # add DownloadSensor = DownloadSensor, to force the
DateIN for download,
    # in case of error during download set Duration = 1 and then set vack Duration = 7, launch a Check_Download in between
    # Down_SOS returns the whole dataFrame (old and new data) and save SOSData.Rdata and SOSdata.csv in general_data

    # setting the name of sensors
    var.names.meteo <-c("Temperature","Relative_humidity",  "Atmospheric_pressure")
    if(exists("SOSData")){
      # List of Pollutant/sensor installed in the AirSensEUR
      var.names.sens <- colnames(SOSData)[-which(colnames(SOSData)=="date")]
      #
      if(length(var.names.sens) == 0){
        stop(paste0("[main] ERROR no sensor variable downloaded for ",AirsensEur.name," at the apiEndPoint. Please check in SOS client -> STOP"))
      } else cat(paste0("[main] INFO, Sensor variables existing in the dataframe downloaded at the apiEndPoint: ", paste0(var.names.sens, collapse = ", "), ",
plus date added"), sep = "\n")
      #
      # Setting the Sensor names
      var.name.GasSensors      <- var.names.sens[-(which(var.names.sens %in% var.names.meteo))]
    } else { # if we do not have new data for sensors we use the names of sensors in sens2ref
      var.name.GasSensors <- na.omit(sens2ref$gas.sensor)
      var.names.sens      <- c(var.name.GasSensors, var.names.meteo)
```

```
  }

  # Saving Sensor data - It is alredy saved, it is not needed to save again, just use Make.old
  SOS.Rdata.file  = file.path(WDoutput, "SOSData.Rdata")
  SOS.csv.file    = file.path(WDoutput, "SOSData.csv"  )
  #save(SOSData, file = SOS.Rdata.file)
  #write.csv(SOSData, file = SOS.csv.file)
  cat(paste0("[main] INFO, Influx Sensor data saved in ", SOS.Rdata.file, " & ", SOS.csv.file,". Updating copies in .old files."), sep = "\n")
  Make.Old(File = SOS.Rdata.file)
  Make.Old(File = SOS.csv.file)

 } else cat(paste0("[main] INFO, there is no request of sensor data download from SOS server in ASEConfig.R (Down.SOS set to FALSE)"), sep = "\n")

} else cat(paste0("[main] INFO, sensor data download from SOS already updated (DownloadSensor$Retrieve.data.SOS set to FALSE)"), sep = "\n")

#------------------------------------------------------------
# Merging SensData/SOSData and RefData
#------------------------------------------------------------
if(exists("RefData")){
  if(exists("SensData")){

  # Fine adjusting of SensData$date due to delays
  if(Delay != 0) SensData$date <- SensData$date + Delay * 60


  # Trying to rbind.fill SensData and SOSdata
  # we prefer SensData data over SOSData if they exist
  if(exists("SOSData")){

    # Fine adjusting of SensData$date due to delays
    if(Delay != 0) SOSData$date <- SOSData$date + Delay * 60

    # if SOSData and SensData give subsequent data
    index.Date.SOSData <- which(SOSData$date %in% SensData$date)
    General <- merge(x = rbind.fill(SOSData[!(row.names(SOSData) %in% index.Date.SOSData),], SensData), y = RefData, by = "date", all.x = TRUE )
    General <- General[General$date >= min(min(SensData$date),min(SOSData$date)) & General$date <= max(max(SensData$date),max(SOSData$date)),]
    # if(ASEConfig == "ASEconfig_02.R" ){
    # General <- merge(x = rbind.fill(SOSData, SensData)[rbind.fill(SensData,SOSData)$date>= as.POSIXct("2016-09-09",tz="UTC"),],
    #               y = RefData[RefData$date>= as.POSIXct("2016-09-09",tz="UTC"),], by = "date", all.x = TRUE )
    # }
  } else {

    # NO SOSData
    # In case of names with raw data, the digital values in raw form are not saved in Genera.data, they are only kept in the airsenseur.db if Down_Influx is used
    if(any(grepl(pattern = "_raw", x = colnames(SensData)))) {
```

```
      General <- merge(x = SensData[-grep(pattern = "_raw", x = colnames(SensData)),], Y = RefData, by = "date", all = TRUE )

    } else General <- merge(x = SensData, y = RefData, by = "date", all = TRUE )

    General <- General[General$date >= min(SensData$date) & General$date <= max(SensData$date),]
  }

  cat("[main] INFO General data frame, there are new sensor data and new reference data ", sep = "\n")


 } else {

  if(exists("SOSData")){

    # Fine adjusting of SensData$date due to delays
    if(Delay != 0) SOSData$date <- SOSData$date + Delay * 60

    General <- merge(x = SOSData, y = RefData, by = "date", all = TRUE ) # we keep all data if SOS data exist
    General <- General[General$date >= min(SOSData$date) & General$date <= max(SOSData$date),]
    # For ASEConfig02, if SOSData and SensData give  consecutive data
    if(ASEConfig == "ASEconfig_02.R"){
      General <- merge(x = rbind.fill(SOSData, SensData)[rbind.fill(SensData,SOSData)$date>= as.POSIXct("2016-09-09",tz="UTC"),],
                       y = RefData[RefData$date>= as.POSIXct("2016-09-09",tz="UTC"),], by = "date", all.x = TRUE )
    }
  } else {
    General <- RefData
    stop("[main] ERROR General data frame, there are no new sensor data while there new reference data ", sep = "\n")
  }
 }

} else {
 if(exists("SensData")){

   # Fine adjusting of SensData$date due to delays
   if(Delay != 0) SensData$date <- SensData$date + Delay * 60

   # raw data, the digital values in raw form are not saved in Genera.data, they are only kept in the airsenseur.db
   General <- timeAverage(SensData, avg.time = paste0(toString(UserMins)," ","min"), statistic = "mean", start.date = round(min(SensData$date), units =
"hours"))
   if(any(grepl(pattern = "_raw", x = colnames(SensData)))) {
     General <- General[-grep(pattern = "_raw", x = colnames(General)),]
   }
   cat("[main] ERROR General data frame, there are new sensor data while there are new reference data ", sep = "\n")

 } else {
```

```
   if(exists("SOSData")){

     # Fine adjusting of SensData$date due to delays
     if(Delay != 0) SOSData$date <- SOSData$date + Delay * 60

     General     <- timeAverage(SOSData, avg.time = paste0(toString(UserMins)," ","min"), statistic = "mean"
                          , start.date = round(min(SOSData$date), units = "hours"))
   } else {
     General <- NA
     cat("[main] ERROR General data frame, there are no new sensor data nor reference data ", sep = "\n")
   }
  }
}

# Select only the dataframe (not tbl_df" and "tbl")
General <- data.frame(timeAverage(General, avg.time = paste0(toString(UserMins)," ","min"), statistic = "mean", start.date = round(min(General$date), units = "hours") - 60 * 60, end.date = round(max(General$date), units = "hours")))

#---------------------------------------------------------------
# discarding rows with NA and NaN for All gas sensors
#---------------------------------------------------------------
ind <- apply(General[,var.name.GasSensors], 1, function(x) !all(is.na(x)))
General <- General[ind,]
# replacing NaN with NA
for(i in names(General)[which(names(General)!="date")]) General[is.nan(General[,i]),i] <- NA

#---------------------------------------------------------------
# Plotting downloaded data in General_data
#---------------------------------------------------------------
timePlot(General, pollutant = names(General)[which(names(General)!="date")], date.pad = TRUE, auto.text = FALSE, y.relation = "free")
WDoutput <- file.path(DisqueFieldtestDir, "General_data")
dev.copy(png,filename = file.path(WDoutput, paste0(AirsensEur.name,"_Full_time_series_",format(min(General$date, na.rm = TRUE),"%Y%m%d"),"_",format(max(General$date, na.rm = TRUE),"%Y%m%d"),".png"))
      , units = "cm", res = 600, width = 40, height = 20);
dev.off()

#---------------------------------------------------------------
# Saving downloaded data in General_data
#---------------------------------------------------------------
# Saving Sensor data - It is alredy saved maybe it is not needed to save again, just use Make.old - No it is needed General does not exist
WDoutput = file.path(DisqueFieldtestDir, "General_data")
General.Rdata.file  = file.path(WDoutput, "General.Rdata")
General.csv.file    = file.path(WDoutput, "General.csv"  )
if(file.exists("Reference.name")){
  cat(paste0("[main] INFO, General data for ",Reference.name," saved in ", General.Rdata.file, " & ", General.csv.file,". Updating copies in .old files."), sep = "\n")
```

```
} else{
  cat(paste0("[main] INFO, General data saved in ", General.Rdata.file, " & ", General.csv.file,". Updating copies in .old files."), sep = "\n")
}
save(General, file = General.Rdata.file)
write.csv(General, file = General.csv.file)
Make.Old(File = General.Rdata.file)
Make.Old(File = General.csv.file)
# Free memory
#DEL if(exists("RefData")) rm(RefData)
#DEL if(exists("SensData")) rm(SensData)
if(exists("SOSData")) rm(SOSData)

#------------------------------------------------------------
#   plot Retrieved data (only the newly added data in Retrieved_data
#------------------------------------------------------------
# creating General.plot for later use in plotting
if(!is.null(DownloadSensor$DateIN.Sens.prev)){
  General.plot <- General[General$date > DownloadSensor$DateIN.Sens.prev, ]
} else {
  General.plot <- General
}

# Which reference raw variable are common to sensor variables?
if(exists("sens2ref")) Ref2SensCol <- match(x=paste0(var.name.GasSensors), table = sens2ref$gas.sensor) else {
  sens2ref          <- data.frame(name.gas         = var.name.GasSensors,
                                  check.names      = FALSE,
                                  stringsAsFactors  = FALSE)
  sens2ref$gas.sensor <- var.name.GasSensors
  Ref2SensCol       <- 1:length(var.name.GasSensors)
}

cat("", sep = "\n")
cat("[main] INFO, Plot new sensor variables", sep = "\n")
WDoutput <- file.path(DisqueFieldtestDir, "Verification_plots")
#
if(all(is.na(General.plot[,var.names.sens]))){
  cat("[main] ERROR All newly downloaded sensor time series are empty, not plotting new times series", sep = "\n")
} else {
  timePlot(General.plot, pollutant = paste0(sens2ref$gas.sensor[Ref2SensCol]), date.pad=TRUE, auto.text = FALSE, y.relation = "free")
  dev.copy(png,filename          =          file.path(WDoutput,          paste0(AirsensEur.name,"_",format(min(General.plot$date,          na.rm          =
TRUE),"%Y%m%d"),"_",format(max(General.plot$date, na.rm = TRUE),"%Y%m%d"),".png"))
        , units = "cm", res = 600, width = 40, height = 20);
  dev.off()

  #
  # Plotting reference values
```

```
 if(exists("var.names.ref")){
  cat("[main] INFO, Plot reference data", sep = "\n")
  if(all(is.na(General.plot[,var.names.ref]))){
   cat("[main] ERROR All newly downloaded sensor time seeries are empty, not plotting new times series", sep = "\n")
  } else {
   timePlot(mydata = General.plot, pollutant = paste0(var.names.ref), date.pad=TRUE, auto.text = FALSE, y.relation = "free")
   dev.copy(png,filename = file.path(WDoutput, paste0(Reference.name[Ref2SensCol],"_",format(min(General.plot$date, na.rm = TRUE),"%Y%m%d"),"_",format(max(General.plot$date, na.rm = TRUE),"%Y%m%d"),".png"))
        , units = "cm", res = 600, width = 40, height = 20);
   dev.off()

   # Sensor relationships with other variables
   for(i in Ref2SensCol){
    cat(paste0("[main] INFO, Plot for Sensor ", sens2ref$name.sensor[i], " relationships with other variables"), sep = "\n")
    Relationships <- na.omit(colnames(General.plot)[colnames(General.plot) %in% paste0(t(Effect_Sens[[i]]))])
    AddOut <- which(Relationships %in% c(var.name.GasSensors,var.names.ref))
    Relationships[AddOut] <- paste0(Relationships[AddOut])
    timePlot(mydata = General.plot, pollutant = Relationships, date.pad=TRUE, auto.text = FALSE, y.relation = "free")
    # save plots in files
    dev.copy(png,filename = file.path(WDoutput, paste0(sens2ref$name.sensor[i],"_",format(min(General.plot$date, na.rm = TRUE),"%Y%m%d"),"_",format(max(General.plot$date, na.rm = TRUE),"%Y%m%d"),".png"))
        , units = "cm", res = 600, width = 40, height = 20);
    dev.off()

   }
  }
 }
}
# Cleaning memory space
remove(WDoutput, General.plot)  # MG General was included in this remove, I deleted General from this list otherwise we cannot treat data after this loop!

#-------------------------------------------------------------
# Flagging warming data and outliers for each sensor
#-------------------------------------------------------------
General.t.Valid <- selectByDate(General, start = Start <- min(General$date, na.rm = TRUE ), end = max(General$date, na.rm = TRUE ))
cat(paste0("[main] INFO, number of valid measurement: ",length(General.t.Valid$date)), sep = "\n")

# Setting output directory
WDoutput <- file.path(DisqueFieldtestDir, "General_data")

# Looking for start of sensors (when sensors change from all NA to numbers) removing one day
ind <- apply(General.t.Valid[var.name.GasSensors], 1, function(x) !all(is.na(x)))
ind <- which(ind[2:length(ind)] & !ind[1:(length(ind)-1)])
# 12 hours of NAs after warming
for(i in ind) General.t.Valid[i:(i+round(12*60/UserMins)),var.name.GasSensors] <- NA

# Testing for outliers for Sensor data. Done one sensor by sensor for the TRUE: sens2ref$Sensor.rm.Out[match(x=i, table = sens2ref$gas.sensor)]
```

```
for(i in var.name.GasSensors){
  if(sens2ref$Sensor.rm.Out[match(x=i, table = sens2ref$gas.sensor)]){
    if(all(is.na(General.t.Valid[,i]))){
      cat(paste0("[main] ERROR All newly data sensor for ", i, " are empty, cannot filter outliers"), sep = "\n")
    } else {
      cat(paste0("[main] INFO, filtering sensor data for outliers in ", i), sep = "\n")
      cat(paste0("[main] INFO, plotting the outliers for ", i), sep = "\n")
      Outli <- My.rm.Outliers(ymin         = sens2ref$Sens.Ymin[match(x=i, table = sens2ref$gas.sensor)],
                              ymax         = sens2ref$Sens.Ymax[match(x=i, table = sens2ref$gas.sensor)],
                              ThresholdMin = sens2ref$Sens.ThresholdMin[match(x=i, table = sens2ref$gas.sensor)],
                              Date         = General.t.Valid$date,
                              y            = General.t.Valid[,i],
                              window       = sens2ref$Sens.window[match(x=i, table = sens2ref$gas.sensor)],
                              threshold    = sens2ref$Sens.threshold[match(x=i, table = sens2ref$gas.sensor)]
      )
      title(main=paste0("Outliers of ",i, ", initial test"))
      dev.copy(png,filename = file.path(WDoutput, paste0(AirsensEur.name,"_Outliers_",i,".png"))
               , units = "cm", res = 600, width = 40, height = 20)
      dev.off()
      All_Outliers <- unique(c(which(Outli$Low_values),which(Outli$High_values),which(Outli$OutliersMin),which(Outli$OutliersMax)))
      # replace outliers with Na only lenth All_Outliers
      if(!length(All_Outliers)==0){

        # if there are outliers put NA for rows All_Outliers
        General.t.Valid[,paste0("Out.",i)] <- NA
        General.t.Valid[-All_Outliers,paste0("Out.",i)] <- General.t.Valid[-All_Outliers,i]
      } else General.t.Valid[,paste0("Out.",i)] <- General.t.Valid[,i]

    }
  } else {
    cat(paste0("[main] INFO, discarding of outliers not requested for ", i, " in the config file "), sep = "\n")
    General.t.Valid[,paste0("Out.",i)] <- General.t.Valid[,i]
  }
}
remove(Outli)

#-------------------------------------------------------------
# Re-Testing for outliers for sensors Before calibration
#-------------------------------------------------------------
# Done one sensor by sensor for the TRUE: sens2ref$Sensor.rm.Out[match(x=i, table = sens2ref$gas.sensor)]
### Shiny inputs number of iteration of precise outlier discarding WINDOW 5
iterations <- 2
### End of shiny input
for(n in 1:iterations){
  for(i in var.name.GasSensors){
    if(sens2ref$Sensor.rm.Out[match(x=i, table = sens2ref$gas.sensor)]){
```

```
    if(all(is.na(General.t.Valid[,i]))){
      cat(paste0("[main] ERROR All data sensor for ", i, " are empty, cannot filter outliers"), sep = "\n")
    } else {
      cat(paste0("[main] INFO, precise filtering sensor data for outliers in ", i), sep = "\n")
      cat(paste0("[main] Plotting the outliers for ", i), sep = "\n")
      Outli <- My.rm.Outliers(ymin        = sens2ref$Sens.Ymin[match(x=i, table = sens2ref$gas.sensor)],
                              ymax        = sens2ref$Sens.Ymax[match(x=i, table = sens2ref$gas.sensor)],
                              ThresholdMin = sens2ref$Sens.ThresholdMin[match(x=i, table = sens2ref$gas.sensor)],
                              Date        = General.t.Valid$date,
                              y           = General.t.Valid[,paste0("Out.",i)],
                              window      = sens2ref$Sens.window[match(x=i, table = sens2ref$gas.sensor)],
                              threshold   = sens2ref$Sens.threshold[match(x=i, table = sens2ref$gas.sensor)]
      )
      title(main=paste0("Outliers for ", i, " , iteration ",n))
      dev.copy(png,filename = file.path(WDoutput, paste0(AirsensEur.name,"_Outliers_",i,"_",n,".png"))
               , units = "cm", res = 600, width = 40, height = 20);
      dev.off()
      All_Outliers <- unique(c(which(Outli$Low_values),which(Outli$High_values),which(Outli$OutliersMin),which(Outli$OutliersMax)))
      # replace outliers with Na only lenth All_Outliers
      if(!length(All_Outliers)==0){

        # if there are outleirs put NA for rows All_Outliers
        General.t.Valid[All_Outliers,paste0("Out.",i)] <- NA
        #General.t.Valid[-All_Outliers,paste0("Out.",i)] <- General.t.Valid[-All_Outliers,i]

      } #else General.t.Valid[,paste0("Out.",i)] <- General.t.Valid[,i] # if there are no outliers or maybe do nothing?
    }
  } else {
    cat(paste0("[main] INFO, discarding of outliers not requested for ", i, " in the config file "), sep = "\n")
    General.t.Valid[,paste0("Out.",i)] <- General.t.Valid[,i]
  }
 }
}

#-------------------------------------------------------------
# Select sensor data within temperature and RH validity ranges in out.i
#-------------------------------------------------------------
cat("[main] INFO, Select sensor data within temperature and RH validity ranges", sep = "\n")
length(General.t.Valid$date)
index.temp <- which(colnames(General.t.Valid) %in% var.names.meteo[1])   # Temperature
index.rh   <- which(colnames(General.t.Valid) %in% var.names.meteo[2])   # Humidity
for(i in var.name.GasSensors){
  if(any((General.t.Valid[, index.temp] < sens2ref$temp.thres.min[match(x=i, table = sens2ref$gas.sensor)] &
          General.t.Valid[, index.temp] > sens2ref$temp.thres.max[match(x=i, table = sens2ref$gas.sensor)]) |
         (General.t.Valid[, index.rh]   < sens2ref$rh.thres.min[match(x=i, table = sens2ref$gas.sensor)] &
          General.t.Valid[, index.rh]   > sens2ref$rh.thres.max[match(x=i, table = sens2ref$gas.sensor)]), na.rm = TRUE)){
```

```
   General.t.Valid[(General.t.Valid[, index.temp] < sens2ref$temp.thres.min[match(x=i, table = sens2ref$gas.sensor)] &
                    General.t.Valid[, index.temp] > sens2ref$temp.thres.max[match(x=i, table = sens2ref$gas.sensor)]) |
                   (General.t.Valid[, index.rh]   < sens2ref$rh.thres.min[match(x=i, table = sens2ref$gas.sensor)] &
                     General.t.Valid[, index.rh]   > sens2ref$rh.thres.max[match(x=i, table = sens2ref$gas.sensor)]),paste0("Out.",i)] <- NA
 }

}
remove(index.temp,index.rh)

#------------------------------------------------------------
# digits2volt conversion for whole data retrieved, either with or without outliers
#------------------------------------------------------------
cat("", sep = "\n")
cat("[main] INFO, digits2volt conversion for sensor data", sep = "\n")
for(i in var.name.GasSensors){
  Ref2SensColi <- match(x=i, table = sens2ref$gas.sensor)
  if(sens2ref$Sensor.rm.Out[Ref2SensColi]){
    General.t.Valid[,paste0(sens2ref$name.gas[Ref2SensColi],"_volt")] <-
      as.numeric(ASEDigi2Volt(Sensors_Cal = sens2ref[Ref2SensColi,], Digital = General.t.Valid[,paste0("Out.",sens2ref$gas.sensor[Ref2SensColi])]))
  } else {
    General.t.Valid[,paste0(sens2ref$name.gas[Ref2SensColi],"_volt")] <-
      as.numeric(ASEDigi2Volt(Sensors_Cal = sens2ref[Ref2SensColi,], Digital = General.t.Valid[,paste0(sens2ref$gas.sensor[Ref2SensColi])]))
  }
}
# index of sensor data in sens2ref and valid General dataframe
Ref2SensCol <- match(x=paste0(var.name.GasSensors), table = sens2ref$gas.sensor)
General.t.Valid[,paste0(sens2ref$name.gas[Ref2SensCol],"_DV")] <- General.t.Valid[,paste0(sens2ref$name.gas[Ref2SensCol],"_volt")] -
  t(matrix(data = rep(x = (sens2ref$Ref-sens2ref$RefAD), times = nrow(General.t.Valid)), ncol = nrow(General.t.Valid) ))[,Ref2SensCol]



#------------------------------------------------------------
#   plot validated data with covariates
#------------------------------------------------------------
# save plots in files
# Sensor variables
cat("", sep = "\n")
cat("[main] INFO, Plot new sensor variables", sep = "\n")
WDoutput <- file.path(DisqueFieldtestDir, "Verification_plots")
#
if(all(is.na(General.t.Valid[,var.names.sens]))){
  cat("[main] ERROR All sensor time series are empty, not plotting new times series", sep = "\n")
} else {

   # Sensor relationships with other variables
   for(i in Ref2SensCol){
     cat(paste0("[main] INFO, Plot for Sensor ", sens2ref$name.sensor[i], " relationships with other variables"), sep = "\n")
```

```
    Relationships        <- na.omit(colnames(General.t.Valid)[colnames(General.t.Valid) %in% paste0(t(Effect_Sens[[i]]))])
    AddOut               <- which(Relationships %in% c(var.name.GasSensors))
    Relationships[AddOut] <- paste0(Relationships[AddOut])
    if(any(grepl(pattern = "DateIN", names(sens2ref))) & any(grepl(pattern = "DateEND", names(sens2ref)))){
      date.index         <- which(General.t.Valid$date >= sens2ref$DateIN[i] & General.t.Valid$date <= sens2ref$DateEND[i])
    } else {
      date.index         <- which(General.t.Valid$date >= min(General.t.Valid$date,na.rm = TRUE)
                              & General.t.Valid$date <= max(General.t.Valid$date,na.rm = TRUE) )
    }
    timePlot(mydata = General.t.Valid[date.index,], pollutant = Relationships, date.pad=TRUE, auto.text = FALSE, y.relation = "free",main = sens2ref$name.sensor[i])
    # save plots in files
    dev.copy(png,filename = file.path(WDoutput, paste0(sens2ref$name.sensor[i],"_",format(min(General.t.Valid$date, na.rm = TRUE),"%Y%m%d"),"_",format(max(General.t.Valid$date, na.rm = TRUE),"%Y%m%d"),".png"))
          , units = "cm", res = 600, width = 40, height = 20)
    dev.off()
   }
}
# Cleaning memory space
remove(WDoutput)

#------------------------------------------------------------
# Plot validated variables full data series
#------------------------------------------------------------
# Updating the var.names.sens, removing outliers if any
var.names.sens       <- paste0("Out.",sens2ref$gas.sensor[Ref2SensCol])
var.names.sens_Volt  <- paste0(sens2ref$name.gas[Ref2SensCol],"_volt")

### we should use another variable name here in order not to loose sens2ref$gas.reference2use
if(!grepl(pattern = "Out.", x = sens2ref$gas.reference2use)) sens2ref$gas.reference2use <- paste0("Out.",sens2ref$gas.reference2use)

cat("", sep = "\n")
cat("[main] INFO, Selecting plot validated variables full data series", sep = "\n")

#Set time interval, with function interval of package lubridate
cat("[main] INFO, Plot sensor variables", sep = "\n")
if(any(grepl(pattern = "DateINmeas", names(sens2ref))) & any(grepl(pattern = "DateENDmeas", names(sens2ref)))){
  DateIN  <- min(sens2ref$DateINmeas, na.rm = TRUE)
  DateEND <- max(sens2ref$DateENDmeas, na.rm = TRUE)
} else {
  DateIN  <- min(General.t.Valid$date,na.rm = TRUE)
  DateEND <- max(General.t.Valid$date, na.rm = TRUE)
}
#
timePlot(selectByDate( General.t.Valid, start = DateIN, end = DateEND), pollutant = var.names.sens_Volt
       , date.pad=TRUE, auto.text = FALSE, y.relation = "free")
# save plots in files
```

```
WDoutput <- file.path(DisqueFieldtestDir, "SensorDataRetrieved_plots")
dev.copy(png,filename = file.path(WDoutput, paste0(AirsensEur.name,"_",format(DateIN,"%Y%m%d"),"_",format(DateEND,"%Y%m%d"),".png"))
      , units = "cm", res = 600, width = 40, height = 20);
dev.off()

remove(WDoutput)

#------------------------------------------------------------
# Plot calibrated sensor data using slope and intercept of ASEConfig.R
#------------------------------------------------------------
# Updating the var.names.sens, removing outliers if any
var.names.sens       <- paste0("Out.",sens2ref$gas.sensor[Ref2SensCol])
var.names.sens_Volt  <- paste0(sens2ref$name.gas[Ref2SensCol],"_volt")
WDoutput             <- file.path(DisqueFieldtestDir, "Modelled_gas")

cat("", sep = "\n")
cat("[main] INFO, plot calibrated sensor data using slope and intercept in the ASEConfig. R file", sep = "\n")

# calibrating
for(i in var.names.sens_Volt){
  ippb <- gsub(pattern = "volt", replacement = "ppb", i)
  name.gas <- paste0(unlist(strsplit(i, split = "_"))[1:(length(unlist(strsplit(i, split = "_")))-1)], collapse = "_")
  index_gas <- match(x = name.gas, table = sens2ref$name.gas)
  General.t.Valid[,ippb] <- (General.t.Valid[,i] - sens2ref$Intercept[index_gas]) / sens2ref$Slope[index_gas]
}
var.names.sens_ppb <- gsub(pattern = "volt", replacement = "ppb", var.names.sens_Volt)
# plotting
timePlot(selectByDate(General.t.Valid, start = DateIN, end = DateEND), pollutant = var.names.sens_ppb
      , date.pad=TRUE, auto.text = FALSE, y.relation = "free")
# save plots in files
dev.copy(png,filename = file.path(WDoutput, paste0(AirsensEur.name,"_ppb_",format(DateIN,"%Y%m%d"),"_",format(DateEND,"%Y%m%d"),".png"))
      , units = "cm", res = 600, width = 40, height = 20);
dev.off()

# Save Rdata and csv, adding new line to General.Rdata if general.prev was saved
cat("", sep = "\n")
Rdata.file    = file.path(WDoutput, paste0(AirsensEur.name, ".Rdata"))
csv.file      = file.path(WDoutput, paste0(AirsensEur.name, ".csv" ))
if(file.exists("Reference.name")){
  cat(paste0("[main] INFO, Saving sensor (",AirsensEur.name,") & Reference data (",Reference.name,
        ") in ", AirsensEur.name,".csv & ", AirsensEur.name,".Rdata. Updating copies in .old files."), sep = "\n")
} else{
  cat(paste0("[main] INFO, Saving sensor (",AirsensEur.name,") in ", AirsensEur.name
        ,".csv & ", AirsensEur.name,".Rdata. Updating copies in .old files."), sep = "\n")
}
save(General.t.Valid, file = Rdata.file)
```

```
write.csv(General.t.Valid, file = csv.file)
Make.Old(File = Rdata.file)
Make.Old(File = csv.file)
remove(WDoutput)
```

```
Functions4ASE.R
#================================================================
# This script contains R-functions written for ASE$_Load_Data_Sensorweb -- to be included into Draft Sensor_Toolbox
#================================================================
# Licence:
# Copyright 2017 EUROPEAN UNION
# Licensed under the EUPL, Version 1.2 or subsequent versions of the EUPL (the "License");
# You may not use this work except in compliance with the License.
# You may obtain a copy of the License at: http://ec.europa.eu/idabc/eupl
# Unless required by applicable law or agreed to in writing, the software distributed
# under the License is distributed on an "AS IS" basis, WITHOUT WARRANTIES OR CONDITIONS
# OF ANY KIND, either express or implied. See the License for the specific language
# governing permissions and limitations under the License.
# Date: 02/11/2016
#
# Authors
# - Michel Gerboles       , michel.gerboles@jrc.ec.europa.eu  - European Commission - Joint Research Centre
# - Laurent Spinelle      , laurent.spinelle@jrc.ec.europa.eu - European Commission - Joint Research Centre
# - Maria Gabriella Villani,                                  - ENEA
#================================================================
#
# Functions:
#
# 1603?? MGV: fromGeneral2DataIN       general data prepared to be fed into the Solve linear equations models. With na removed (NOT INCLUDED)
# 141114 MGV: Cal_Line_mgv             calibration and plot of calibration line, modified version of Cal_Line with option Mod_type=='Linear.Robust', to model
the median of y as a function of x, rather than modelling the mean of y as a function of x, in the case of least squares regression. (NOT INCLUDED)
# 160418 MGV: Validation.tool          for validations (of what? ) (NOT INCLUDED)
# 160503 MGV: citytechNO2O3ModelDelta   model to solve gas sensors components  (only NO2 and O3) (NOT INCLUDED)
# 160505 MGV: SensorModel              model to solve gas sensors components  (for NO2, CO, O3 and NO?) (NOT INCLUDED)
# 161129 MG : SensorModel              Modification of the equation to make it workign for any corrections function and set of sensors (NOT INCLUDED)

# 161029 MG : Check_Download           Check if General.Rdata file exists and if it is necessary to retrieve.data (true or false)
# 161030 MG : Down_SOS                 Download Sensor data from SOS?
# 161030 MG : Down_Ref                 Download Reference for data retrieving (NOT INCLUDED)
# 161031 MG : My.rm.Outliers           Removing outliers using the Median Average Deviation method
# 161107 MG : ASEDigi2Volt             converting sensor digital values to voltages
# 161107 MG : ASEVolt2Conc             converting sensor voltages to concetrations
# 161107 MG : Convert.ASE              converting sensor digital values to concetrations
```

```
# 161112 MG : Invalid.out            Removing Invalid data when ASE is indoor or not at a monitoring station (NOT INCLUDED)
# 161120 MG : Down_Influx              Downloading AirSensEUR data using the Influx protocol, create or update the airsenseur.db SQLite database, get timezone (NOT INCLUDED)
# 161120 CR : get_google_tz          Getting time zone from Google API (NOT INCLUDED)
# 161123 MG : Sqlite2df              converting a local airsenseur.db into a General dataframe (NOT INCLUDED)
# 161125 MG : Make.Old               Creating a copy of the file as old file
# 170420 MG : load.packages          install and/or load packages
#
#=================================================================

#=================================================================
# Version History
#=================================================================
# 2017-01-26 First Commit
# 2017-02-03 SQLite2df, line 223: storing Page when reading airsenseur.db to be used for Aggregating in tabulated form
#                   line 299: adding the date when a sensor name changes
#                   line 302 - 364: paging the "Putting data in tabulated dataframe" to be progress of casting and timeAverage
#         Down_Influx, line 691 - 697: correction for determining the time zone.
#         Validation_Tool, line 2093 - : remove time from the names of files when saving a plot
#

################################################################################
###
### Function Load.Packages (170420)
###
################################################################################
Load.Packages <- function(list.Packages) {
 # list.Packages              vector of names of the packages to load
 #
 cat("", sep = "\n")
 cat("[Load.Packages] INFO CHECK Installed packages and Toolbox to run the script", sep = "\n")
 #
 for(i in list.Packages) {

  # Installing packages
  if(i %in% rownames(installed.packages()) == FALSE) {
   cat(sprintf("[Load.Packages] INFO Installing %s", i), sep = "\n")
   install.packages(i)
  } else {
   cat(sprintf("[Load.Packages] INFO Package %s already installed",i), sep = "\n")
  }
  # cat(i,quote=FALSE)
  do.call("library", as.list(i))
  #library(i, character.only = TRUE)
  cat(sprintf("[Load.Packages] INFO Package %s loaded",i), sep = "\n")
```

```
 }
 #
 # List of loaded packages
 cat("[Load.Packages] INFO List of installed packages", sep = "\n")
 print(search(), quote = FALSE)
 cat("", sep = "\n")

}


#=================================================================
# 161125 MG : Make.Old              Creating a copy of the file as old file
#=================================================================
Make.Old <- function(File, File.old = NULL){
   # Make.Old updates or make a copy of File into File.old  . File is updated only if the date of File is newer than the one of File.old
   # File File.old is NULL, .old is added to File ans is used as the name of File.old

   # File                    : charater, name of the file to be copied
   # File.old                : charater, default is NULL
   #
   # Return                  : Send a message il File.old was Updated or created or not modified


   # creating File.old if it does not exist
   if(is.null(File.old)){

      File.old <- paste0(File, ".old")
      # cat(paste0("[Make.Old] INFO, the current ", File, " is being copied into ", File.old), sep = "\n")
      file.copy(from = File, to = File.old, overwrite = TRUE, copy.mode = TRUE, copy.date = TRUE)
      return(cat(paste0("[Make.Old] INFO, ", File.old, " was created."), sep = "\n"))

   } else {

      if(file.exists(File.old)) {

         #checking for newer dates
         if(file.info(File)$mtime > file.info(File.old)$mtime){

            # New version of File, save File.Old
            # cat(paste0("[Make.Old] INFO, the current ", Rdata.file, " is being copied into ", File.old), sep = "\n")
            file.copy(from = Rdata.file, to = Rdata.Old.file, overwrite = TRUE, copy.mode = TRUE, copy.date = TRUE)
            return(cat(paste0("[Make.Old] INFO, ", File.old, " was updated."), sep = "\n"))


         } else return(cat(paste0("[Make.Old] INFO, the current ", File.old, " is up to date. No need to make a a copy.", File.old), sep = "\n"))
```

```
    } else {

      # cat(paste0("[Make.Old] INFO, the current ", File, " is being copied into ", File.old), sep = "\n")
      file.copy(from = File, to = File.old, overwrite = TRUE, copy.mode = TRUE, copy.date = TRUE)
      return(cat(paste0("[Make.Old] INFO, ", File.old, " did not exist. It was created."), sep = "\n"))

    }

  }

  # if(file.exists(File.old)) return(cat(paste0("[Make.Old] INFO, ", File.old, " was created."), sep = "\n")) else return(cat(paste0("[Make.Old] INFO ", File.old, " could not be created."), sep = "\n"))

}

#================================================================================
# 161107 MG : functions for converting digital values
#================================================================================
ASEDigi2Volt <- function(Sensors_Cal, Digital){
  # Sensors_Cal: dataframe with the following column vectors:
  #       Sensors   : a vector of string with the names of sensors mounted on the AirSensEUR
  #       Ref       : a float vector, one value per sensor in Sensors_Cal$Sensors, giving the voltage in the middle of the analogue to digital converter (ADC) on the shield
  #       RefAD     : a float vector, one value per sensor in Sensors_Cal$Sensors, corresponding to the range of the ADC conversion (ref ? RefAD)
  #       Intercept : a float vector, one value per sensor in Sensors_Cal$Sensors, intercept of conversion equation Conc[] = Intercept + Slope x Voltage
  #       Slope     : a float vector, one value per sensor in Sensors_Cal$Sensors, intercept of conversion equation Conc[] = Intercept + Slope x Voltage
  #       Unit      : vector of strings, one value per sensor in Sensors_Cal$Sensors, with the Unit names after conversion of Digital values into concetration values
  # Digital    : a vector of floats, with the Digital values to be converted into Voltages
  # return     : the function return a vector with 1 column vector, giving the voltages for all sensors
  return ( t(t(2*Sensors_Cal$RefAD * (Digital+1)/(2^16)) + Sensors_Cal$Ref - Sensors_Cal$RefAD ) )
  # The name of the sensors are given by the names of the column in Digital
}
ASEVolt2Conc <- function(Sensors_Cal, Voltage){
  # Sensors_Cal: dataframe with the following column vectors:
  #       Sensors   : a vector of string with the names of sensors mounted on the AirSensEUR
  #       Ref       : a float vector, one value per sensor in Sensors_Cal$Sensors, giving the voltage in the middle of the analogue to digital converter (ADC) on the shield
  #       RefAD     : a float vector, one value per sensor in Sensors_Cal$Sensors, corresponding to range of the ADC conversion (ref ? RefAD)
  #       Intercept : a float vector, one value per sensor in Sensors_Cal$Sensors, intercept of conversion equation Conc[] = Intercept + Slope x Voltage
  #       Slope     : a float vector, one value per sensor in Sensors_Cal$Sensors, intercept of conversion equation Conc[] = Intercept + Slope x Voltage
  #       Unit      : vector of strings, one value per sensor in Sensors_Cal$Sensors, with the Unit names after conversion of Digital values into concetration values
  # Voltage    : a dataframe with 4 columns representing the voltage values of the 4 sensors of an AirSensEUR, each columns are floats vector, to be corrected with Sensors_Cal$Sensors
  #       gives the Voltage values to be converted into concentration values
  # Curemtly a simple linear calibration is applied
```

```
  return (  t( (t(Voltage) -  Sensors_Cal$Intercept) / Sensors_Cal$Slope ))
}
Convert.ASE  <- function(Sensors_Cal, Digital){
  # Sensors_Cal: dataframe with the following column vectors:
  #           Sensors  : a vector of string with the names of sensors mounted on the AirSensEUR
  #           Ref      : a float vector, one value per sensor in Sensors_Cal$Sensors, with the voltage in the middle of the analogue to digital converter (ADC) on the shield
  #           RefAD    : a float vector, one value per sensor in Sensors_Cal$Sensors, corresponding to range of the ADC conversion (ref ? RefAD)
  #           Intercept: a float vector, one value per sensor in Sensors_Cal$Sensors, intercept of conversion equation Conc[] = Intercept + Slope x Voltage
  #           Slope    : a float vector, one value per sensor in Sensors_Cal$Sensors, intercept of conversion equation Conc[] = Intercept + Slope x Voltage
  #           Unit     : vector of strings, one value per sensor in Sensors_Cal$Sensors, with the Unit names after conversion of Digital values into concetration values
  #
  # Digital    : vector of digital values to be converted to Volt and Concentration
  # return a dataframe with voltage and concetration values
  Voltage <- ASEDigi2Volt(Sensors_Cal = Sensors_Cal, Digital = Digital)
  Conc    <- ASEVolt2Conc(Sensors_Cal = Sensors_Cal, Voltage = Voltage)

  return (data.frame(Voltage_V=Voltage
                , Conc=Conc
                , check.names = FALSE, stringsAsFactors = FALSE)  )
}

#================================================================
# 161031 MG : Removing outliers using the Median Average Deviation method
#================================================================
# REMOVING Ouliers - An upper threshold ("utmax" and "utmin) calculation based on the MAD:
# Can be replaced by mad(x, center = median(x), constant = threshold, na.rm = TRUE, high = TRUE) and 2nd , low = TRUE instead of , high = TRUE
utmax <- function(x, threshold) {m = median(x, na.rm = TRUE); return(median(x, na.rm = TRUE) + threshold * median(abs(x - m), na.rm = TRUE))}
utmin <- function(x, threshold) {m = median(x, na.rm = TRUE); return(median(x, na.rm = TRUE) - threshold * median(abs(x - m), na.rm = TRUE))}
My.rm.Outliers <- function(ymin = NULL, ymax = NULL, ThresholdMin = NULL, Date, y, window, threshold){
  # ymin              = minimum values for y, for example to remove negative values
  # ThresholdMin      = minimum values for zmin, the minimum values that evidence outliers when exceeded
  # Date              = x axis, Date values Posixct
  # y                 = time series on which ouliers are detected
  # window            = width of the window used to compute median and average
  # threshold         = coefficient that muliplied by the difference between median and average that is exceeded result in outlier
  # Return            = dataframe with Logical for low values, logical if value exceed lower MAD, logical if value exceed High MAD,
  #                     low MAD and high MAD

  # Removing values lower than ymin
  if(!is.null(ymin)){Low_values  <- (y<= ymin)}
  if(!is.null(ymax)){High_values <- (y>= ymax)}
  # max limits
  zmax <- zoo::rollapply(zoo(y), width = window, FUN = utmax, threshold = threshold, align="center", partial = TRUE)
  #zmax <- c(rep(zmax[1], window-1), zmax) # Use z[1] throughout the initial period. # Not needed if align center and partial = TRUE
  OutliersMax <- y > zmax
```

```
  #
  # min limits
  zmin <- zoo::rollapply(zoo(y), width = window, FUN = utmin, threshold = threshold, align="center", partial = TRUE)
  # Changing negative values in ThresholdMin
  if(!is.null(ThresholdMin)){zmin[which(zmin<ThresholdMin)] <- ThresholdMin}

  #zmin <- c(rep(zmin[1], window-1), zmin) # Use z[1] throughout the initial period. # Not needed if align center and partial = TRUE
  OutliersMin <- y < zmin

  # Graph the data, show the ut() cutoffs, and mark the outliers:
  # saving the original par values
  op <- par(no.readonly = TRUE)
  Xlim <- c(min(Date, na.rm = TRUE), max(Date, na.rm = TRUE))
  Ylim <- c(min(y, na.rm = TRUE), max(y, na.rm = TRUE))
  plot(Date , y, type="p", lwd=2, col="#E00000", xlim = Xlim, ylim = Ylim, ylab = "Raw Sensor values", xlab = "", cex = 0.2, xaxt = "n", yaxt = "n")
  lines(Date, y, col="#E00000")
  dates <- pretty(Date, n = 25)
  par(xaxp = c(dates[1], dates[length(dates)], (length(dates)-1)), yaxp = c(min(y, na.rm = TRUE), max(y, na.rm = TRUE), 10))
  grid (NULL,NULL, col = "lightgray", lty = "dotted")
  axis.POSIXct(1, at = dates, las=1, format = "%d-%b")
  Ylim <- format(seq(min(y, na.rm = TRUE), max(y, na.rm = TRUE), by = (max(y, na.rm = TRUE)-min(y, na.rm = TRUE))/10), scientific =FALSE, digits = 0)
  axis(2, at = Ylim, las=1, tcl = NA, mgp = c(3, 0.3, 0))

  lines(Date, zmax, col="Gray")
  points(Date[OutliersMax] , y[OutliersMax], pch=19)
  lines(Date, zmin, col="Gray")
  points(Date[OutliersMin] , y[OutliersMin], pch=19)
  points(Date[Low_values]  , y[Low_values] , pch=19, col = "Blue")
  points(Date[High_values] , y[High_values], pch=19, col = "Violet")
  # Passing and resuming the par values
  np <- par(op) # to be returned

  # returnng the data frame
  df <- data.frame(Low_values = Low_values, High_values = High_values, OutliersMin = OutliersMin, OutliersMax = OutliersMax, zmin = zmin, zmax = zmax)
  return(df)
}
#BeginDate <- as.POSIXct("2016-10-10")
#EndDate   <- as.POSIXct("2016-10-26")
#My.rm.Outliers(General[General$date>as.POSIXct("2016-01-01"),  "date"],  General[General$date>as.POSIXct("2016-01-01"),  "Ozone"],  window, threshold)
#Essai <- My.rm.Outliers(ymin = ymin, ThresholdMin = ThresholdMin, General[General$date>BeginDate & General$date<EndDate, "date"],
#                  General[General$date>BeginDate & General$date<EndDate, "Ozone"],
#                  window = sens2reference$window[grep(pattern="Ozone", x = sens2reference$gas.sensor)],
#                  threshold = sens2reference$threshold[grep(pattern="Ozone", x = sens2reference$gas.sensor)])

#=============================================================
```

```
# 161030 MG :  Download Sensor data from SOS and later from InfluxDB?
#================================================================
Down_SOS <- function(AirsensEur.name, UserMins, DownloadSensor = NULL, AirsensWeb, Duration = NULL){

  # AirsensEur.name      = Name of for AirSensEUR for SOS download
  # UserMins             = periodicity of data requested for the returned dataframe
  # List_Pollutant       = Names of pollutants that are mesured by sensors (Not NEEDED THE NAME ARE AUTOMATICALLy SET)
  # DownloadSensor       = a list with possible values
  #                  General.Rdata, the name of the data frame with downloaded data in directory WDinput
  #                  WDinput, the directory where the Rdata are saved
  #                  Retrieve.data.SOS, true if data need be retrieved
  #                  DateIN.SOS.prev, date to start download of SOS sensor data if SensData.Rdata already exist (may not exist)
  #                  The time zone is the one of SOS (GMT).
  #                  Default value for DownloadSensor is NULL, nothing passed. In this case the Down_SOS
  #                  will create new Rdata/csv determining DateIN and DateEND using SOS information
  # AirSensWeb           = URI of the SOS server
  # Duration             = integer, the number of days to downlaod per page (as Limit in SQL), default is NULL, data ae downlaoded in slices of 7 days
  # return               = dataframe SensData with the data to be added + 2 files are savedd SOSData.Rdata and SOSData.csv

  #-------------------------------------------------------------
  # Sensor Data retrieving at apiEndpoint
  #-------------------------------------------------------------
  cat(paste0("[Down_SOS] INFO, ", AirsensEur.name," sensor data retrieving"), sep = "\n")
  # connect
  apiEndpoint <- Endpoint(AirsensWeb)
  # number of category at the apiEndpoint
  cat(paste0("[Down_SOS] INFO, in total ", length(timeseries(apiEndpoint)), " Sensors at the SOS client."), sep = "\n")

  # Selecting service "AirSensEUR" with name
  srv <- services(apiEndpoint)[match(x="AirSensEUR", table=label(services(apiEndpoint)))]

  # get all phenomena
  phe <- phenomena(apiEndpoint)
  label(phenomena(apiEndpoint))
  #cat("[Down_SOS] INFO, phenomena: ", paste0(label(phe), collapse = ","), sep = "\n")

  # get station AirsensEur.name
  if(AirsensEur.name %in% label(stations(srv))) sta <- stations(srv)[match(x=AirsensEur.name, table=label(stations(srv)))] else
    stop(cat(paste0("[Down_Influx] ERROR, ", AirsensEur.name, " not found at the apiEndpoint. Correct the name of AirSensEUR or set Down.Influx to FALSE in the config file"), sep = "\n"))

  # Select the timeseries of the station AirsensEur.name
  ts <- timeseries(sta)
  if(AirsensEur.name=="JRC_C5_01") {
    ts <- ts[-grep(pattern = "http://www.airsenseur.org/ch1/o3_3e1f/11915854-335/1", x = label(ts))]
  }
```

```
  if(AirsensEur.name=="JRC_C5_05") {
    ts <- ts[-grep(pattern = "1_old", x = label(ts))]
  }

  cat(label(ts),sep = "\n")
  # fetch all the meta data of ts
  ts <- fetch(ts)
  # Position
  geom <- sp::geometry(sta)
  cat(paste0("Position of station ", AirsensEur.name, ":"),sep = "\n")
  head(geom@coords)

  # Phenomenon at the station
  Sensors <- data.frame(label(phenomenon(ts)), stringsAsFactors = FALSE)

  ### Trying to determine the name of variable using label(phenomenon(ts)), we will check if "Temperature", "Relative humidity" and "Atmospheric pressure"
are in the label(phenomenon(ts))
  label.variable = any(grepl(pattern = "Temperature", x=Sensors[,1]), na.rm = FALSE) &
    any(grepl(pattern = "Relative humidity", x=Sensors[,1]), na.rm = FALSE) &
    any(grepl(pattern = "Atmospheric pressure", x=Sensors[,1]), na.rm = FALSE)
  if(label.variable){

    # Removing Institute from the Category and replace blank spaces with _
    Sensors <- data.frame(apply(Sensors, 1, function(x) {x <- substr(x, start=1, stop=unlist(gregexpr(pattern = " (", text = x, fixed="TRUE"))-1); return(x)}))
    Sensors <- data.frame(Pollutants = apply(Sensors, 1, function(x) {x <- sub(pattern = " ", replacement = "_",x); return(x)}), stringsAsFactors = FALSE)

  } else {

    # Defining names and variables for meteo and gas sensors - Used the same names of variables as in SOS for compatibility reasons
    # # meteo
    Sensors$Pollutants[which(grepl(pattern = "ch6", x = Sensors[,1]))] <- "Relative_humidity"
    Sensors$Pollutants[which(grepl(pattern = "ch5", x = Sensors[,1]))] <- "Temperature"
    Sensors$Pollutants[which(grepl(pattern = "ch4", x = Sensors[,1]))] <- "Atmospheric_pressure"
    # sensors
    #-----------------------------------------------------------------
    # Adding sensor model type if more than 1 model of sensors then the model type are separated with "!" - The last sensor mdel type is used
    #-----------------------------------------------------------------
    Sensor.names      <- list(Nitrogen_dioxide    = c("no2_b43f","NO2-B43F", "NO2_C_20", "NO2/C-20", "NO23E50"),
                      Carbon_monoxide    = c("co_a4","CO-B4", "CO_MF_200", "CO_MF_20", "CO/MF-200", "CO3E300","co_mf_200"),
                      Ozone              = c("o3_a431","O3_M-5","O3/M-5", "O3-B4", "O33EF1","o3_m_5"),
                      Nitric_oxide       = c("no_b4","no-b4_op1","NO-B4", "NO_C_25", "NO/C-25","NO3E100")) # Add new sensor model to be recognized
if needed
    # Setting gas names in Sensors$Pollutants
    for(i in 1:length(Sensor.names)) {
      for(j in 1: length(Sensor.names[[i]])) {
        if(any(grepl(pattern = Sensor.names[[i]][j], x = Sensors[,1]))){
          Sensors$Pollutants[which(grepl(pattern = Sensor.names[[i]][j], x = Sensors[,1]))] <- names(Sensor.names)[i]
```

```
            break
          }
        }
      }
    remove(Sensor.names)
    cat("[Down_SOS] INFO, sensors found in the SOS APi:", sep = "\n")
    print(Sensors)


}
##Sensors$ts <- as.numeric(ts)

#-------------------------------------------------------------
# Downloading sensor data
#-------------------------------------------------------------
# Saving files : names
SOS.Rdata.file  = file.path(DownloadSensor$WDinput, "SOSData.Rdata")
SOS.csv.file    = file.path(DownloadSensor$WDinput, "SOSData.csv"  )
# Determining DateIn and DateEND for data downlaod with a lubridate::interval
DateEND <- max(time(lastValue(ts)))
# set DateIN for data retrieving, either from origin or last date in previous DataFrame
if(is.null(DownloadSensor)){
   if("SOSData.Rdata" %in% list.files(DownloadSensor$WDinput)){#  Rdata file exists : take the last date in dataframe General

      load(SOS.Rdata.file); DateIN <- max(SOSData$date, na.rm = TRUE)

   } else {# Rdata file dos not exist: take the value for SOS with minimum vaue at 2015-12-01

      DateIN  <- max(as.POSIXct(strptime("2015-12-01 00:00:00", format= "%Y-%m-%d %H:%M:%S", tz = "UTC")), max(time(firstValue(ts))) ) # Tz is set to "UTC" to avoid conflict with Refdata which is in UTC although SOS used GMT

   }
} else {# DownloadSensor exists: check if we have a "DateIN.SOS.prev"

   if(any(grepl(pattern = "DateIN.SOS.prev", x = objects(DownloadSensor)))){# # DateIN.SOS.prev does not exist: take the value for SOS with minimum vaue at 2015-12-01

      if(!is.null(DownloadSensor$DateIN.SOS.prev))
         DateIN  <- DownloadSensor$DateIN.SOS.prev else DateIN  <- max(as.POSIXct(strptime("2015-12-01 00:00:00", format= "%Y-%m-%d %H:%M:%S", tz = "UTC")),
                                                                       max(time(firstValue(ts))) )
   } else DateIN  <- max(as.POSIXct(strptime("2015-12-01 00:00:00", format= "%Y-%m-%d %H:%M:%S", tz = "UTC")), max(time(firstValue(ts))) )

}
# Setting end date to curent date
if(is.null(Duration)) Duration <- 7 # length of interval to download in days
```

```
  DateIN.partial  <- DateIN
  DateEND.partial <- DateIN + 3600 * 24 * Duration

  while(DateIN.partial < DateEND ){

    date.partial <- lubridate::interval(DateIN.partial, DateEND.partial)
    # Downloading
    cat(paste0("[Down_SOS] INFO, downloading from ", DateIN.partial, " to ", DateEND.partial), sep = "\n")
    Buffer <- lapply(ts, function(x){Buffer <-getData(x, timespan=date.partial);return(Buffer)})
    #Buffer  <-  mapply(function(x){Buffer  <-getData(x,  timespan=date.partial);  colnames(Buffer)  <-  c("date",  Sensors$Pollutant[i]);return(Buffer)},  ts,
Sensors$Pollutant)

    if(exists("Frame")) rm(Frame)
    for(i in 1:length(Sensors$Pollutant)){
      #Buffer <- getData(ts[i], timespan=date.partial) # I suspect that if the timespan has no data, getData returns a error
      Buffer.df <- data.frame(Buffer[[i]])
      colnames(Buffer.df) <- c("date", Sensors$Pollutant[i])
      if(exists("Frame")) Frame <- merge(Frame,Buffer.df, by = "date", all = TRUE) else Frame <- Buffer.df
    }


    #Writing file
    if(nrow(Frame)!= 0){
      Frame <- timeAverage(Frame, avg.time = paste0(toString(UserMins)," ","min"), statistic = "mean", start.date = round(DateIN.partial, units = "hours"),
fill = TRUE)
      #})

      if(!"SOSData.Rdata" %in% list.files(DownloadSensor$WDinput)){

        SOSData <- data.frame(Frame)

      } else {

        load(SOS.Rdata.file)
        SOSData <- rbind.fill(SOSData, data.frame(Frame)) # if merge, add , by = "date", all = TRUE
      }

      # convert SOSData$date to UTC to be consistent with reference data
      if(any(base::format(SOSData$date, format="%Z") != "UTC")) attr(SOSData$date, "tzone") <- "UTC"

      save(SOSData, file = SOS.Rdata.file)
      write.csv(SOSData, file = SOS.csv.file  )

    }

    # Setting time interval to one duration more
```

```
    DateEND.partial <- DateEND.partial + 3600 * 24 * Duration
    DateIN.partial  <- DateIN.partial  + 3600 * 24 * Duration

  }

  load(SOS.Rdata.file)
  return(SOSData)

}

#================================================================
# 161029 MG: Check General.R data file and the retrieve.data true or false
#================================================================
Check_Download <- function(AirsensEur.name, WDinput, UserMins){
  # AirsensEur.name        = Name of for AirSensEUR
  # WDinput                = Sub directory of DisqueFieldtest where are the Refdata and Sensdata Rdata files
  # UserMins               = periodicity of data requested after final data treatment
  # Return                 = a list with
  #                        Ref.Rdata.file and Sens.Rdata.file, the name of the dataframe with with reference and sensor downloaded data
  #                        WDinput, the directory where the Rdata are saved
  #                        Retrieve.data.Ref, true if reference data need be retrieved
  #                        Retrieve.data.Sens, true if sensor data need be retrieved (Influxdb)
  #                        Retrieve.data.SOS, true if sensor data need be retrieved (SOS)
  #                        DateIN.Ref.prev, date to start download of reference data (last existing date), Null if "RefData.Rdata" does not exist
  #                        DateIN.Sens.prev, date to start download of sensor data (last existing date), Null if ""SensData.Rdata" does not exist
  #                        DateIN.SOS.prev, date to start download of sensor data (last existing date), Null if ""SOSData.Rdata" does not exist

  # Set the Rdata file of input data
  Ref.Rdata.file     = file.path(WDinput, "RefData.Rdata")
  Sens.Rdata.file    = file.path(WDinput, "SensData.Rdata")
  SOS.Rdata.file     = file.path(WDinput, "SOSData.Rdata")
  cat(paste0("[Check_Download] INFO, Check presence of ",Ref.Rdata.file, " and ",Sens.Rdata.file," in ", WDinput), sep = "\n")

  if(!dir.exists(WDinput)) {

    cat(paste0("[Check_Download] INFO, Directory", WDinput, "does not exist. It is going to be created. All sensor and reference data are going to be downloaded."), sep = "\n")
    dir.create(WDinput, showWarnings = TRUE, recursive = FALSE, mode = "0777")

    # in this case dwonloading of all sensor and reference data is necessary
    Retrieve.data.Ref  = TRUE
    DateIN.Ref.prev    = NULL
    Retrieve.data.Sens = TRUE
    DateIN.Sens.prev   = NULL
    Retrieve.data.SOS  = TRUE
    DateIN.SOS.prev    = NULL
```

```
} else {

  if(!file.exists(Ref.Rdata.file)){# Ref.Rdata.file does not exist

    Retrieve.data.Ref   = TRUE
    DateIN.Ref.prev     = NULL
    cat(paste0("[Check_Download] INFO, ", Ref.Rdata.file, " does not exist. It is going to be created, data will be retrieved."), sep = "\n")

  } else {# Ref.Rdata.file exists

    cat(paste0("[Check_Download] INFO, ", Ref.Rdata.file, " exists."), sep = "\n")
    # need be ckecked
    load(Ref.Rdata.file)
    # Not considering end rows with only NA values for sensors
    ind <- apply(RefData[names(RefData)!="date"], 1, function(x) !all(is.na(x)))
    DateIN.Ref.prev <- max(RefData[ind,"date"])
    if(difftime(Sys.time(), DateIN.Ref.prev, units = "mins") > UserMins){    ### MG , I doubt about the tz here, I think all is changed to UTM, as it is a difference maybe it does not matter

      Retrieve.data.Ref = TRUE
      # re-assign initial date for data retrieval  used in step 5
      cat(paste0("[Check_Download] INFO, reference data are going to be retrieved. Start new reference data at : ", DateIN.Ref.prev), sep = "\n")

    } else {

      Retrieve.data.Ref = FALSE
      DateIN.Ref.prev     = NULL
      cat(paste0("[Check_Download] INFO, No Data are going to be retrieved. The latest data are already downloaded, please restart in at least ", UserMins, "mins."), sep = "\n")

    }
  }

  if(!file.exists(Sens.Rdata.file)){

    Retrieve.data.Sens  = TRUE
    DateIN.Sens.prev    = NULL
    cat(paste0("[Check_Download] INFO, ", Sens.Rdata.file, " does not exist. It is going to be created, sensor data will be retrieved."), sep = "\n")

  } else {

    cat(paste0("[Check_Download] INFO, ", Sens.Rdata.file, " exists."), sep = "\n")
    load(Sens.Rdata.file)
    if(difftime(Sys.time(), max(SensData$date), units = "mins") > UserMins){   ### MG , I doubt about the tz here, I think all is changed to UTM, as it is a difference maybe it does not matter
```

```
            Retrieve.data.Sens  = TRUE
            DateIN.Sens.prev <- max(SensData$date)
            cat(paste0("[Check_Download] INFO, sensor data are going to be retrieved. Start date for data download: ", DateIN.Sens.prev), sep = "\n")

          } else {

            Retrieve.data.Sens  = FALSE
            DateIN.Sens.prev    = NULL
            cat(paste0("[Check_Download] INFO, no sensor data are going to be retrieved. The latest data are already downloaded, please restart in at least ", UserMins, "mins."), sep = "\n")

          }
        }

        if(!file.exists(SOS.Rdata.file)){

          Retrieve.data.SOS   = TRUE
          DateIN.SOS.prev     = NULL
          cat(paste0("[Check_Download] INFO, ", SOS.Rdata.file, " does not exist. It should be created, SOS sensor data should be retrieved."), sep = "\n")

        } else {

          cat(paste0("[Check_Download] INFO, ", SOS.Rdata.file, " exists."), sep = "\n")
          load(SOS.Rdata.file)
          if(difftime(Sys.time(), max(SOSData$date), units = "mins") > UserMins){   ### MG , I doubt about the tz here, I think all is changed to UTM, as it is a difference maybe it does not matter

            Retrieve.data.SOS   = TRUE
            DateIN.SOS.prev     <- max(SOSData$date)
            cat(paste0("[Check_Download] INFO, SOS sensor data are going to be retrieved. Start date for data download: ", DateIN.Sens.prev), sep = "\n")

          } else {

            Retrieve.data.SOS   = FALSE
            DateIN.SOS.prev     = NULL
            cat(paste0("[Check_Download] INFO, no SOS sensor data are going to be retrieved. The latest data are already downloaded, please restart in at least ", UserMins, "mins."), sep = "\n")

          }
        }
      }

    # Showing DownloadSens
    print(list(Ref.Rdata.file = Ref.Rdata.file, Sens.Rdata.file  = Sens.Rdata.file, SOS.Rdata.file = SOS.Rdata.file,
               WDinput = WDinput,
               Retrieve.data.Ref  = Retrieve.data.Ref , DateIN.Ref.prev  = DateIN.Ref.prev,
```

```
            Retrieve.data.Sens = Retrieve.data.Sens, DateIN.Sens.prev = DateIN.Sens.prev,
            Retrieve.data.SOS  = Retrieve.data.SOS , DateIN.SOS.prev  = DateIN.SOS.prev), quote = FALSE)
   return(list(Ref.Rdata.file = Ref.Rdata.file, Sens.Rdata.file  = Sens.Rdata.file, SOS.Rdata.file = SOS.Rdata.file,
            WDinput = WDinput,
            Retrieve.data.Ref  = Retrieve.data.Ref , DateIN.Ref.prev  = DateIN.Ref.prev,
            Retrieve.data.Sens = Retrieve.data.Sens, DateIN.Sens.prev = DateIN.Sens.prev,
            Retrieve.data.SOS  = Retrieve.data.SOS , DateIN.SOS.prev  = DateIN.SOS.prev))

}
```

ASEConfig.R

```
#=============================================================
# Configuration of parameters for AirSensEUR calibration - file: ASEconfig.R
#=============================================================
# Licence:
# Copyright 2017 EUROPEAN UNION
# Licensed under the EUPL, Version 1.2 or subsequent versions of the EUPL (the "License");
# You may not use this work except in compliance with the License.
# You may obtain a copy of the License at: http://ec.europa.eu/idabc/eupl
# Unless required by applicable law or agreed to in writing, the software distributed
# under the License is distributed on an "AS IS" basis, WITHOUT WARRANTIES OR CONDITIONS
# OF ANY KIND, either express or implied. See the License for the specific language
# governing permissions and limitations under the License.
# Date: 05/16/2017
#
# Authors
# - Michel Gerboles       , michel.gerboles@jrc.ec.europa.eu  - European Commission - Joint Research Centre
# - Laurent Spinelle      , laurent.spinelle@jrc.ec.europa.eu - European Commission - Joint Research Centre
# - Maria Gabriella Villani,                                  - ENEA
#=============================================================

#=============================================================
# Version History
#=============================================================
# 2017-01-26 First Commit
# 2017-02-03 Valid periods defined per sample and implementation in General.t.valid
# 2017-02-08 Temperature and humidity ranges defined per sample and implementation in General.t.valid
# 2017-02-08 Calibration model defined per sample and implementation in General.t.valid
# 2017-02-08 Calibration and extrapolation periods defined per sample and implementation in General.t.valid

#=============================================================
# Contents ASEconfig.R
#=============================================================
# USER CONFIG PARAGRAPH
# LIST OF PARAMETERS TO BE SET BY THE USER (1--9):
```

```
#  1. create file system structure, going to root files system, create log file
#  2. Loading ToolBox                                          (ONLY FUNCTIONS4ASE.R)
#  3. Configuring Proxy server
#  4. Install packages needed in the script (CRAN + Github)
#  5. Sensor configuration for download for Influx and SOS.                  (ONLY SOS)
#  6. Reference data, configuration for download, ftp                        (NOT INCLUDED)
#  7. Create sensor configuration file and matching between reference and sensor names
#  8. SET Average time for sensor data
#  9. SET temperature and relative humidity thresholds for sensors validity
# 10. Calibration with Etalonnage
# 11. Valid Periods                                           (NOT USED)
# 12. SET TIME PARAMETERS -> see in ASE_OPER_SCRIPT.R                       (NOT USED)
# 13. SET  Models for NO2 and O3 system resolution                          (NOT USED)
#==============================================================

#==============================================================
#  1. create file system structure, create log file
#==============================================================

#--------------------------------------------------------------
# Check directories existence or create them, create log file
#--------------------------------------------------------------
#  setting the current directory to the root of the file system
if(tail(unlist(strsplit(DisqueFieldtest, split="/")), n =1) != unlist(strsplit(ASEConfig, split = ".R"))) {
  DisqueFieldtestDir <- file.path(DisqueFieldtest, unlist(strsplit(ASEConfig, split = ".R")) )
}
cat(paste0("[ASEConfig] INFO Checking if file system for data treatment exist or create it at ", DisqueFieldtestDir), sep = "\n")
#
List.Dirs                                                                                                                                                    <-
c("Calibration","Drift","Estimated_coef","General_data","Models","Modelled_gas","scriptsLog","SensorDataRetrieved_plots","Statistics","Verification_plots")
for(i in List.Dirs){
   if(!dir.exists(file.path(DisqueFieldtestDir, i))){
      dir.create(file.path(DisqueFieldtestDir, i), showWarning = TRUE, recursive = TRUE)
      cat(paste0("[ASEConfig] INFO Dir. created: ", file.path(DisqueFieldtestDir,i)), sep = "\n\r")
   } else cat(paste0("[ASEConfig] INFO Dir. already exists: ", file.path(DisqueFieldtestDir,i)), sep = "\n")
}

#--------------------------------------------------------------
# Change to the script directory to DisqueFieldtestDir
#--------------------------------------------------------------
cat(paste0("[ASEConfig] INFO Change to the working directory: ",DisqueFieldtestDir), sep = "\n")
# Store the current directory
initial.dir <- getwd()
# Change directory
if(!(initial.dir == DisqueFieldtestDir)) setwd(DisqueFieldtestDir)
```

```
#------------------------------------------------------------
# sending console to a file
#------------------------------------------------------------
sink(file.path(DisqueFieldtestDir, "scriptsLog",paste0("console_", Sys.Date(),".log"))
    , type=c("output", "message"), split = TRUE, append=TRUE ) # with split = TRUE we get the file on the screen and in log file
cat(paste0("[ASEConfig] INFO Starting log file ", file.path(DisqueFieldtestDir, "scriptsLog",paste0("console_", Sys.Date(),".log"))), sep = "\n")
cat("", sep = "\n")

#============================================================
#  2. Loading toolBox
#============================================================
# Directory for Sensor_ToolBox
### Shiny input  Window 1 ###
#DEL List.Sensor_Toolbox_dir <- c(file.path(Disque, "MACPoll", "WP4", "Task 4.3", "D435"))  # for Sensor_ToolBox
### END of Shiny input ###
#DEL for (i in List.Sensor_Toolbox_dir) if (dir.exists(i)) {
#DEL    Sensor_Toolbox_dir <- i
#DEL    cat(paste0("[ASE_Config] INFO ToolBox dir found: ",List.Sensor_Toolbox_dir), sep = "\n"); break}
#DEL if(!dir.exists(Sensor_Toolbox_dir)) {
#DEL    stop(cat(paste0("[[ASE_Config] ERROR Dir ", Sensor_Toolbox_dir, "not found, stopping the process"), sep = "\n"))
#DEL }

# Sensor ToolBox Files checking  and file "ASEconfig_MG.R"
#DEL SensorToolboxDirCal <- file.path(Sensor_Toolbox_dir, "151016 Sensor_Toolbox.R") # USARE Draft Sensor_Toolbox (laurent.spinelle@jrc.ec.europa.eu).r
#DEL SensorToolboxDir  <- file.path(Sensor_Toolbox_dir, "Draft Sensor_Toolbox.R" ) # USARE Draft Sensor_Toolbox (laurent.spinelle@jrc.ec.europa.eu).r
if(!"Functions4ASE.R" %in% list.files(DisqueFieldtest)){
   stop(paste0("[main] ERROR files Functions4ASE.R not found, stopping the process. Please copy Functions4ASE.R in ", DisqueFieldtest))
} else {
   cat(paste0("[main] ERROR files Functions4ASE.R is found"), sep = "\n")
}
setwd(".."); ASETools <- file.path(getwd(), "Functions4ASE.R"    ); setwd(DisqueFieldtestDir)

#HighstatLibFile    <- file.path(Sensor_Toolbox_dir, "HighstatLibV6.R"       )

#DELif (!file.exists(SensorToolboxDirCal)) {
#DEL    stop(paste0("[ASE_Config] ERROR file ", SensorToolboxDirCal, " not found, stopping the process"))
#DEL} else print(paste0("[ASE_Config] INFO file ", SensorToolboxDirCal, " found and ready to be sourced"), quote = FALSE)
#DELif (!file.exists(SensorToolboxDir)) {
#DEL    stop(paste0("[ASE_Config] ERROR file ", SensorToolboxDir, "not found, stopping the process"))
#DEL} else print(paste0("[ASE_Config] INFO file ", SensorToolboxDir, " found and ready to be sourced"), quote = FALSE)
if (!file.exists(ASETools)) {
   stop(paste0("[ASE_Config] ERROR file ", ASETools, " not found, stopping the process"))
} else print(paste0("[ASE_Config] INFO file ", ASETools, " found and ready to be sourced"), quote = FALSE)

#------------------------------------------------------------
```

```
# Source Sensor_Toolbox
#------------------------------------------------------------
cat("[ASE_Config] Sourcing Sensor_Toolboxes", sep = "\n")
#DELsource(SensorToolboxDirCal)
#DELsource(SensorToolboxDir)
#source(HighstatLibFile)
source(ASETools)
# remove unnecessary variables
remove(i,  List.Dirs, ASETools) #Del List.Sensor_Toolbox_dir,

#============================================================
#  3. Configuring Proxy server
#============================================================
# At JRC we have now a proxy that block downlaoding from github, using the following code, we can download from Github
### Shiny input  Window 1: PROXY, url, port, login and password ###
PROXY = TRUE
URL     = "10.168.209.72"; PORT     = 8012; LOGIN    = NULL; PASSWORD = NULL
# no login and no password on our proxy
### END of Shiny input ###
if(PROXY){
   # checking that we have the httr package to use function Set_Config()
   if("httr" %in% rownames(installed.packages()) == FALSE) {
      cat("[ASEConfig] INFO Installing httr", sep = "\n")
      install.packages("httr")
   } else cat("[ASEConfig] INFO Package httr already installed", sep = "\n")
   require("httr")
   cat("[ASEConfig] INFO Package %s loaded", sep = "\n")

   # implement PROXY
   set_config(use_proxy(url=URL, port=PORT, username = LOGIN, password = PASSWORD))
}

#============================================================
#  4. Install packages needed in the script (CRAN + Github)
#============================================================
cat("[ASEConfig] INFO Check or install packages needed to run the script", sep = "\n")
# Packages to be loaded
# Clean and consistent tools to split-apply-combine pattern in R       --> plyr # use the function plyr::rbind.fill to add number of dataframes together
# To locate current file and dir                                       --> R.utils
# To read sensor data                                -                 --> stringi
# Date format for Influxdbr                                            --> xts
# When removing ouliers, using rollapply()                                --> zoo
# Easier management of time interval                                      --> lubridate
# To plot time series                                                  --> openair
# Package needed for devtools::install_github("52North/sensorweb4R")    --> curl
# Two packages needed for github sensorweb4R if you have a proxy       --> futile.options, lambda.r,
```

```
# To configure the proxy when using github to install sensoreb4r       --> httr
# To install libraries for reading sensor urls:sensorweb4r             --> devtools, sp, curl
# To configure the proxy when using github to install sensoreb4r       --> httr
# To solve linear robust linear regression (median)                    --> quantreg
# Function: to solve nls with Levenberg Marquardt method               --> minpack.lm  # We use function nlsLM
# To solve system of linear equation                                   --> limSolve
# to retrieve EMEP data using function: getURL                         --> Rcurl
# To read the airsenseur.db SQLite database                            --> RSQLite, sqldf, RODBC
# TO assemble data frame                                               --> reshape , function colsplit(), function cast in SQLite2df to pass from sequential to tabulated
dataframe
# To get the time zone using the ggogle API in Down_Influx             --> RJSONIO,  XML
# To work with the SQLite, airsenseur.db                               --> sqldf,
# corelation matrix                                                    --> corrplot
# For general additive models, function gam()                          --> mgcv

# library(bitops)
#
list.Packages <- c( "plyr", "openair", "zoo", "futile.options", "lambda.r", "curl", "sp", "httr"
) # , "XLConnect"   is making trouble -> removed
Load.Packages(list.Packages)
# if error on plyr then type install.packages("plyr") at the console

# Install package from Github to read sensor urls, sensorweb4r -
# The packages futile.options and lambda.r need be installed before in order to be able to load sensorweb4r in case of proxy (see above)
list.packages.github <- c("52North/sensorweb4R") #, "dleutnant/influxdbr@0.11.1", "rundel/timezone"
for(i in list.packages.github){

   # removing author name anad version number
   lib.i <- tail(unlist(strsplit(i, split = "/")), n = 1)
   lib.i <- head(unlist(strsplit(lib.i, split = "@")), n = 1)

   if(!(lib.i %in% rownames(installed.packages()))) {
      devtools::install_github(i)
      cat(sprintf("Package ", i, " installed"), sep = "\n")
   } else cat(paste0("[ASEConfig] INFO Package ", i, " already installed"), sep = "\n")

   do.call("library", as.list(lib.i))
   cat(sprintf("[ASEConfig] INFO Package %s loaded",lib.i), sep = "\n")

}
cat("[ASEConfig] INFO List of installed packages", sep = "\n")
print(search(), quote = FALSE)
cat("", sep = "\n")

#============================================================
#  5. Sensor configuration for download for Influx and SOS. InfluxDB has more info and is preferred over SOS
```

```
#============================================================
#------------------------------------------------------------
# SOS
#------------------------------------------------------------
### Shiny input  Window 2 ###
Down.SOS          <- TRUE          # is download from SOS possible ?
AirsensWeb        <- "http://sossvr1.liberaintentio.com:8080/52nSOS/api/v1/"  ### # web site to retrieve Sensor data from SOS
AirsensEur.name   <- "JRC_C5_01"   # Sensor identifier, we have C05_EMEP_01 to C05_EMEP_05 + BO6_EMEP_01, This shall be the last identifier in each line of timeseries(apiEndpoint), see in Down_SOS
# Coordinates of the EMEP sation, coord.sens <- "45 49' N, 8 38' E", altitude of the sensor alt.sens <- "209 m"
# Define time zone
sens.tzone <- "UTC"
### End of Shiny input

### Shiny input  Window 2 ###
# List_Pollutantcontains the possible pollutant names used in SOS
# It may be needed to update the list when there are new pollutants but this list should be sufficient for now
# This list will be used in function Down_SOS that will look for the names of parameters to be downloaded by Down_SOS
# These names are used by SOS-T and we have to replace " " with "_"
List_Pollutant <- c("Carbon dioxide"      , "Carbon monoxide"       , "Nitric oxide"      , "Nitrogen dioxide"   , "Sulfur dioxide",
                    "Particle matter 10"   , "Particles matter 2.5"   , "Particle matter 1"  ,
                    "Relative humidity"    , "Atmospheric pressure"   , "Temperature" )
### End of Shiny input

#============================================================
# 7. Create sensor configuration file and matching between reference and sensor names
#============================================================
# This is to insert both sensors and reference configuratio into a dataframe and file
# Be aware that this is done "ad hoc" according to the reference data files

### Shiny input  Window 4 ###
# Read config file (TRUE) or manually create it (FALSE)?
File_Sensor.config <- FALSE

#1 gas name to refer to
name.gas             <- c("CO"            , "NO"          , "NO2"          , "O3"     , "NOx"    , "SO2"   )
sens2ref             <- data.frame(name.gas        = name.gas,
                                   check.names      = FALSE,
                                   stringsAsFactors  = FALSE)
rm(name.gas)
#3 gas sensor: IMPORTANT - put name.gas and gas.sensor under the same column
# Order of reference parameters not of the sensors on AirSensEUR
sens2ref$gas.sensor    <- c("Carbon_monoxide" , "Nitric_oxide" , "Nitrogen_dioxide", "Ozone"  , NA       , NA      )
# Insert the name given in SOS
sens2ref$name.sensor   <- c("CO_3E300"       , "NO_3E100"     , "NO2_3E50"       , "O3_3E1F" , NA       , NA      )
```

```
### End of Shiny input

if(File_Sensor.config){

  # reading the configuration files sens2reference
  Config.file <- file.path(DisqueFieldtest,"General_data",paste0(AirsensEur.name,".cfg"))
  if(file.exists(Config.file)){
    sens2reference <- read.csv(file = Config.file, header = TRUE, stringsAsFactors = FALSE)
    cat(paste0("[ASEConfig] Info, the config file ", AirsensEur.name, ".cfg for the configuration of AirSensEUR ", AirsensEur.name, " exists"), sep = "\n")
    cat(str(sens2reference), sep = "\n")
  } else stop(cat(paste0("[ASEConfig] The config file of sensors configuration for AirSensEUR: ", AirsensEur.name, " does not exist. Please change File_Sensor.config <- FALSE "), sep = "\n"))

  # reading the files with affecting variables
  for(i in 1:4){
    nameFile <- file.path(DisqueFieldtest,"General_data",paste0(AirsensEur.name,"_Effect_Sens",i,"_",sens2ref$name.sensor[i],".csv"))
    if(file.exists(nameFile)){
      cat(paste0("[ASEConfig] INFO, the file with variables affecting sensor ", nameFile, " exists "), sep = "\n")
      assign(paste0("Sens",i), read.csv(file = nameFile, header = TRUE, comment.char = "#", stringsAsFactors = FALSE))
      cat(paste0("[ASEConfig] INFO, ",nameFile, ": ",get(paste0("Sens",i))), sep = "\n")
    } else stop(cat(paste0("[ASEConfig] ERROR, the file with variables affecting sensor ", nameFile, " exists. Please change File_Sensor.config <- FALSE "), sep = "\n"))
  }
  Effect_Sens <- list(Sens1, Sens2, Sens3, Sens4 )

  # reading the files with period of valid data
  for(i in 1:4){
    nameFile <- file.path(DisqueFieldtest,"General_data",paste0(AirsensEur.name,"_Valid",i,"_",sens2ref$name.sensor[i],".csv"))
    if(file.exists(nameFile)){
      cat(paste0("[ASEConfig] INFO, the file with valid periods of sensor data ", nameFile, " exists "), sep = "\n")
      assign(paste0("valid",i), read.csv(file = nameFile, header = TRUE, comment.char = "#", stringsAsFactors = FALSE))
      cat(paste0("[ASEConfig] INFO, ", get(paste0("valid",i))), sep = "\n")
    } else {
      stop(cat(paste0("[ASEConfig] ERROR, the file with valid periods of sensor data ", nameFile, " exists. Please change File_Sensor.config <- FALSE "), sep = "\n"))
    }
  }
  Valid <- list(valid1,valid2, valid3, valid4)

} else {

  ### Shiny input  Window 4 ###

  #2 gas reference
  sens2ref$gas.reference   <- c("CO_ppm"      , "NO"       , "NO2"          , "O3"      , "NOx"   , "SO2"   )
  sens2ref$gas.reference2use  <- paste0("Ref.", sens2ref$gas.reference)
  sens2ref$ref.unitgas    <- c("ppm"         , "ppb"      , "ppb"          , "ppb"     , "ppb"   , "ppb"   )
```

```
# Filtering reference values? TRUE: outliers are discarded, False outliers are not examined
sens2ref$Ref.rm.Out     <- c(FALSE         , FALSE       ,FALSE           , FALSE     ,FALSE    , FALSE    )
# Outliers filtering for Reference:
## Rolling window: if we want to have 24 hours before sampling : 24 * 6  = 144, 18 --> 3 hours
sens2ref$Ref.window     <- c(18            , 18          ,18              , 18        , 18      , 18      )
# MG coefficent that multiplied by difference between mean and median evidences outliers if exceeded
sens2ref$Ref.threshold  <- c(20            , 30          ,25              , 20        , 25      , 25      )
# Minimum values in data series
sens2ref$Ref.Ymin       <- c(0             ,0            ,0               , 0         , 0       , 0       )
# Minimum values in data series, in unit of reference data series
sens2ref$Ref.Ymax       <- c(30            ,2000         ,500             , 200       , 3000    , 30      )
# Minimum z values for testing
sens2ref$Ref.ThresholdMin<- c(0            ,0            ,0               , 0         , 0       , 0       )

#5 AIRSENSNEUR BOARD: Characteristic volt values, this is set in JAVAControl Panel
sens2ref$Sensor.raw.unit <- c("V"          , "V"         , "V"            , "V"       , NA      , NA      )
sens2ref$Ref            <- c(1.699         , 1.342       , 1.674          , 1.674     , NA      , NA      )
sens2ref$RefAD          <- c(0.5           , 0.5         , 0.5            , 0.5       , NA      , NA      )
sens2ref$board.zero.set  <- c(1.678        , 1.344       , 1.672          , 1.667     , NA      , NA      )
sens2ref$Intercept      <- c(1.678         , 1.344       , 1.672          , 1.667     , NA      , NA      )
sens2ref$Slope          <- c(-0.02189/1000 , 0.00001194  , -0.00004302    , -0.00002379, NA     , NA      )
sens2ref$Sensor.Unit    <- c("ppb"         , "ppb"       , "ppb"          , "ppb"     , NA      , NA      )
# the slopes correspond to the sensor sensitivity in V/ppb. The Sensitivity of CO that is normally given in V/ppm
# was transformed in V/ppb for homogeneity with the other sensors: same unit

#4 Outliers filtering for sensors:
# Filtering Sensors? TRUE: outliers are discarded, False outliers are not examined
sens2ref$Sensor.rm.Out  <- c(TRUE          , TRUE        ,TRUE            ,TRUE       , NA      , NA      )
## Rolling window: if we want to have 24 hours before sampling : 24 * 6  = 144, 18 --> 3 hours
sens2ref$Sens.window    <- c(18            , 18          ,18              ,18         , NA      , NA      ) # use these defaults values first
# MG coefficent that multiplied by difference between mean and median evidences outliers if exceeded
sens2ref$Sens.threshold <- c(20            , 20          ,20              ,20         , NA      , NA      ) # use these defaults values first
# Minimum values in data series, digital values
sens2ref$Sens.Ymin      <- c(28000         ,25000        ,28000           ,26000      , NA      , NA      )
# Maximum values in data series, digital values
sens2ref$Sens.Ymax      <- c(2^16-1        ,2^16-1       ,2^16-1          ,2^16-1     , NA      , NA      )
# Minimum z values for testing
sens2ref$Sens.ThresholdMin<- c(28000       ,500          ,28000           ,26000      , NA      , NA      )

# DEFINE The lists of variables to plot in retrieving data () to understand the interferences - use report of lab. tests
### All shiny input
Sens1  <- data.frame( Effects  = c("Out.Carbon_monoxide", "Relative_humidity", "Temperature", "Atmospheric_pressure", "Out.Nitric_oxide"))
Sens2  <- data.frame( Effects  = c("Out.Nitric_oxide","Relative_humidity", "Temperature", "Atmospheric_pressure", "Out.Carbon_monoxide"))
Sens3  <- data.frame( Effects  = c("Out.Nitrogen_dioxide", "Relative_humidity", "Temperature", "Atmospheric_pressure", "Out.Ozone"))
Sens4  <- data.frame( Effects  = c("Out.Ozone", "Relative_humidity", "Temperature", "Atmospheric_pressure", "Out.Nitrogen_dioxide"))
### END Shiny Input
```

```
Effect_Sens <- list(Sens1, Sens2, Sens3, Sens4 )
for(i in 1:4){
  SENS <- get(paste0("Sens",i))
  write.csv(SENS, file = file.path(DisqueFieldtestDir,"General_data",paste0(AirsensEur.name,"_Effect_Sens",i,"_",sens2ref$name.sensor[i],".csv")),
row.names = FALSE)
}
rm(SENS, Sens1,Sens2,Sens3,Sens4)

#================================================================
# 08. SET Average time for reference and sensor data
#================================================================
# set number of minutes (<60). The unit shall be in minutes
UserMins <- 10

#================================================================
#  9. Calibration with Etalonnage
#================================================================
### All Shuny inputs
# This is to model data with etalonage, MGV suggested Linear.Robust that avoid the ffect of very high or very low values together with remove.neg =TRUE
# Options: "Linear", "Linear.Robust" and Michel's + gam
sens2ref$mod.eta.model.type <- c("Linear.Robust", "Linear.Robust"  , "Linear.Robust", "Linear.Robust", NA      , NA      )
# this is to validate modelled data vs referenced measurements
sens2ref$eta.model.type    <- c("Linear"     , "Linear"        , "Linear"     , "Linear"     , NA      , NA      )
sens2ref$remove.neg        <- c(TRUE         , TRUE            , TRUE         , TRUE         , NA      , NA      )

### END Shuny inputs

#================================================================
# 10. SET temperature and relative humidity thresholds for sensors validity
#================================================================
### All Shuny inputs
sens2ref$temp.thres.min    <- c(-20          , -20             , -20          , -20          , NA      , NA      )
sens2ref$temp.thres.max    <- c(40           , 40              , 40           , 40           , NA      , NA      )
sens2ref$rh.thres.min      <- c(15           , 15              , 15           , 15           , NA      , NA      )
sens2ref$rh.thres.max      <- c(97           , 97              , 97           , 92           , NA      , NA      )

### END Shuny inputs

#================================================================
# 11 - Valid periods - and delays between AirSensEUR and Reference data
#================================================================
# delays in minutes (positive and negative, the values are added) in SensData, default values should be 0
### All Shuny inputs
Delay                  <- 10
### END Shuny inputs
```

```
if(exists("General")){

  cat("[main] INFO, SET TIME parameters", sep = "\n")
  # Set inital date for data retrieving (i.e. maximum length time period for data retrieval).
  # These dates may change according to the data availability
  # UserDateIN.0, sens.tzone is set in ASEConfig_MG.R
  #
  # Set correct time zone

  #-----------------------------------------------------------------
  # Valid Period to be used for sensor Evaluation with reference values, a life cycle of each sensor is needed - time zone shall be the same as sens.tzone (UTC?)
  #-----------------------------------------------------------------
  ### All shiny input Window 5
  # Selecting valid period of Sensors? (valid because ASE is at the station?)
  sens2ref$Sens.Inval.Out  <- c(TRUE          , TRUE        ,TRUE            , TRUE     , NA       , NA      )

  Valid1 <- rbind(c("2015-12-24 00:00", "2016-01-14 14:10"), c("2016-03-01 12:00", "2016-04-01 22:00"), c("2016-04-05 13:20", "2016-06-02 15:00"),
          c("2016-06-07 11:20", "2016-08-06 20:00"), c("2016-08-19 18:50", "2016-09-10 15:30"), c("2016-10-05 12:00", "2016-11-14 16:30"),
          c("2016-11-23 16:00", "2016-11-26 16:30"),c("2016-12-02 16:35", "2016-12-24 10:24"), c("2017-01-09 11:00", "2017-01-13 11:43"),
          c("2017-01-13 13:30", "2017-01-13 17:50"),c("2017-01-13 18:25", "2017-01-18 18:00"), c("2017-01-19 12:00", "2017-02-03 10:45"),
          c("2017-02-05 16:00", "2017-02-07 06:50"))
  Valid2 <- rbind(c("2015-12-24 00:00", "2016-01-14 14:10"), c("2016-03-01 12:00", "2016-04-01 22:00"), c("2016-04-05 13:20", "2016-06-02 15:00"),
          c("2016-06-07 11:20", "2016-08-06 20:00"), c("2016-08-19 18:50", "2016-09-10 15:30"), c("2016-10-05 12:00", "2016-11-14 16:30"),
          c("2016-11-23 16:00", "2016-11-26 16:30"),c("2016-12-02 16:35", "2016-12-24 10:24"), c("2017-01-09 11:00", "2017-01-13 11:43"),
          c("2017-01-13 13:30", "2017-01-13 17:50"),c("2017-01-13 18:25", "2017-01-18 18:00"), c("2017-01-19 12:00", "2017-02-03 10:45"),
          c("2017-02-05 16:00", "2017-02-07 06:50"))
  Valid3 <- rbind(c("2015-12-24 00:00", "2016-01-14 14:10"), c("2016-03-01 12:00", "2016-04-01 22:00"), c("2016-04-05 13:20", "2016-06-02 15:00"),
          c("2016-06-07 11:20", "2016-08-06 20:00"), c("2016-08-19 18:50", "2016-09-10 15:30"), c("2016-10-05 12:00", "2016-11-14 16:30"),
          c("2016-11-23 16:00", "2016-11-26 16:30"),c("2016-12-02 16:35", "2016-12-24 10:24"), c("2017-01-09 11:00", "2017-01-13 11:43"),
          c("2017-01-13 13:30", "2017-01-13 17:50"),c("2017-01-13 18:25", "2017-01-18 18:00"), c("2017-01-19 12:00", "2017-02-03 10:45"),
          c("2017-02-05 16:00", "2017-02-07 06:50"))
  Valid4 <- rbind(c("2015-12-24 00:00", "2016-01-14 14:10"), c("2016-03-01 12:00", "2016-04-01 22:00"), c("2016-04-05 13:20", "2016-06-02 15:00"),
          c("2016-06-07 11:20", "2016-08-06 20:00"), c("2016-08-19 18:50", "2016-09-10 15:30"), c("2016-10-05 12:00", "2016-11-14 16:30"),
          c("2016-11-23 16:00", "2016-11-26 16:30"),c("2016-12-02 16:35", "2016-12-24 10:24"), c("2017-01-09 11:00", "2017-01-13 11:43"),
          c("2017-01-13 13:30", "2017-01-13 17:50"),c("2017-01-13 18:25", "2017-01-18 18:00"), c("2017-01-19 12:00", "2017-02-03 10:45"),
          c("2017-02-05 16:00", "2017-02-07 06:50"))
  ### END shiny input
  Valid <- list(Valid1,Valid2, Valid3, Valid4)
  NewValid<-function(x){
    # making each element a dataframe of POSIXct
    x <-data.frame( x, stringsAsFactors = FALSE)
    colnames(x)<- c("In", "End")
    x$In  <- as.POSIXct(strptime(x$In , "%Y-%m-%d %H:%M"), tz = sens.tzone)
    x$End <- as.POSIXct(strptime(x$End, "%Y-%m-%d %H:%M"), tz = sens.tzone)
```

```
    return(x)
  }
  Valid.date <- lapply(Valid, NewValid)
  for(i in 1:4){
    write.csv(Valid.date[[i]],  file  =  file.path(DisqueFieldtestDir,"General_data",paste0(AirsensEur.name,"_Valid",i,"_",sens2ref$name.sensor[i],".csv")),
row.names = FALSE)
  }
  ## write(Valid, file = file.path(paste0(AirsensEur.name,"_Valid.Rdata")))
  Take.Valid <- function(x, General.df, Valid.General.df){
    # TIMESERIES shall have a date field in POSIXct
    # x is a data frame of interval of POSIXct
    General.df <- General.df[which(General.df$date>=head(x, n=1) & General.df$date<=tail(x, n=1)),]
    Valid.General.df <- rbind(Valid.General.df, General.df)
    return(Valid.General.df)
  }
  # Create the df of Valid date
  for(i in 1:4) if(exists("Start")) Start <- min(Start, min(Valid.date[[i]]$In )) else Start <- min(Valid.date[[i]]$In )
  for(i in 1:4) if(exists("End"))   End   <- max(End  , max(Valid.date[[i]]$End)) else End   <- max(Valid.date[[i]]$End)
  General.t.Valid <- selectByDate(General, start = Start, end = End)
  for(i in 1:length(Valid.date)){ # list of valid date per sensor
    for(j in 1:nrow(Valid.date[[i]]))
      # dates before In date of first row
      if(j==1) General.t.Valid[which(General.t.Valid$date < Valid.date[[i]]$In[j]),paste0(sens2ref$gas.sensor[i])]  <- NA else
        General.t.Valid[which(General.t.Valid$date < Valid.date[[i]]$In[j] &
                              General.t.Valid$date > Valid.date[[i]]$End[j-1]),paste0(sens2ref$gas.sensor[i])] <- NA
      # dates after the End f the last row
      if(j==nrow(Valid.date[[i]])) General.t.Valid[which(General.t.Valid$date > Valid.date[[i]]$End[j]),paste0(sens2ref$gas.sensor[i])] <- NA
      # data between 2 rows of valid dates
  }
}

#=================================================================
# 12. SET TIME PARAMETERS
#=================================================================
if(exists("General.t.Valid")){

  # UserDateIN, UserDateINCal and UserDateINmeas can  be a date to be chosen, "System" or NULL (""); in these cases the code will set UserDateIN.0
and UserDateEND.0.
  # If not, user shall enter the values UserDateIN.0 and UserDateEND.0,  UserDateINCal and UserDateENDCal, UserDateINmeas and
UserDateENDmeas  as charater in the format %Y-%m-%d %H:%M
  # e. g. for
  # UserDateIN    <- "2015-12-24 00:00"    # defined value, the interval is determined by the user
  # UserDateIN    <- ""                    # NULL value   , the interval is determined automatically
  # UserDateIN    <- "System"              # system value , the interval is determined automatically

  ### Shiny input window 6
  # Define how interval for calibration extrapolation is determined, time in TimeZone
```

```
    sens2ref$UserDateIN        <- c("2015-12-24 00:00", "2015-12-24 00:00", "2015-12-24 00:00", "2015-12-24 00:00", NA       , NA      ) # or "System" or
""
    sens2ref$UserDateINCal     <- c("2015-12-24 00:00", "2015-12-24 00:00", "2015-12-24 00:00", "2015-12-24 00:00", NA       , NA      )
    sens2ref$UserDateENDCal    <- c("2016-01-01 00:00", "2016-01-01 00:00", "2016-01-01 00:00", "2016-01-01 00:00", NA       , NA      )
    sens2ref$UserDateINmeas    <- c("2016-01-01 00:00", "2016-01-01 00:00", "2016-01-01 00:00", "2016-01-01 00:00", NA       , NA      )
    sens2ref$UserDateENDmeas   <- c("2017-02-07 06:50", "2017-02-07 06:50", "2017-02-07 06:50", "2017-02-07 06:50", NA       , NA      )
    ### End of Shiny input

    # Initial values
    if(exists("SensData")) TimeZone <- TZ else TimeZone <- sens.tzone
    sens2ref$DateIN        <- as.POSIXct(sens2ref$UserDateIN    , format = "%Y-%m-%d %H:%M", tz = TimeZone)
    sens2ref$DateEND       <- as.POSIXct(sens2ref$UserDateENDmeas, format = "%Y-%m-%d %H:%M", tz = TimeZone)
    sens2ref$DateINCal     <- as.POSIXct(sens2ref$UserDateINCal  , format = "%Y-%m-%d %H:%M", tz = TimeZone)
    sens2ref$DateENDCal    <- as.POSIXct(sens2ref$UserDateENDCal , format = "%Y-%m-%d %H:%M", tz = TimeZone)
    sens2ref$DateINmeas    <- as.POSIXct(sens2ref$UserDateINmeas , format = "%Y-%m-%d %H:%M", tz = TimeZone)
    sens2ref$DateENDmeas   <- as.POSIXct(sens2ref$UserDateENDmeas, format = "%Y-%m-%d %H:%M", tz = TimeZone)

    for(i in which(!is.na(sens2ref$gas.sensor))){
      if(sens2ref$UserDateINCal[i] == "System"){
        # For OPTION == "System"
        # Number of days to take for calibration in case the user defines: "System" as date time option
        # Make a warning Note: if DateEND.0-DateIN.0 < system.cal, "System" becomes "", the NULL option
        sens2ref$system.cal[i]       <- 7  # this is also a Shiny input
      }

      #----------------------------------------------------------------
      # Set inital and final dates for data time series UserDateIN.0
      #----------------------------------------------------------------
      # Be careful with the user time zone. I think both Inlfuxdb and SOS are in UTC. It seems that InfluxDB is in local time
      if( sens2ref$UserDateIN[i] != "System" &  sens2ref$UserDateIN[i] != ""){
        sens2ref$DateIN[i]       <- as.POSIXct(strptime(sens2ref$UserDateIN[i], format = "%Y-%m-%d %H:%M", tz = TimeZone))
        sens2ref$DateEND[i]      <- max(General.t.Valid$date, na.rm = TRUE)
      } else {
        sens2ref$DateIN[i]       <- min(General.t.Valid$date, na.rm = TRUE)
        sens2ref$DateEND[i]      <- max(General.t.Valid$date, na.rm = TRUE) # MG I do not understand why we have tz = "CET" ! I think it should be tz =
TimeZone
      }

      #----------------------------------------------------------------
      # Set initial and final dates to start calibration functions, values for UserDateINCal/UserDateENDCal
      #----------------------------------------------------------------
      if(sens2ref$UserDateINCal[i] != "System" & sens2ref$UserDateINCal[i] != ""){
        sens2ref$DateINCal[i]      <- as.POSIXct(sens2ref$UserDateINCal[i] , format = "%Y-%m-%d %H:%M", tz =TimeZone )
        sens2ref$DateENDCal[i]     <- as.POSIXct(sens2ref$UserDateENDCal[i], format = "%Y-%m-%d %H:%M", tz =TimeZone )
      } else {
        #----------------------------------------------------------------
        # Automatic calibration period
```

```
    #------------------------------------------------------------------
    cat("[main] INFO, SET TIME parameters for Calibration and measuring functions when users do not define dates")
    # number of days between DateEND.0 and DateIN.0
    diff.days <-difftime(sens2ref$DateEND[i], sens2ref$DateIN[i], units = "days")
    diff.days <-as.double(diff.days)
    if(diff.days < system.cal){
      cat(sprintf(paste0("NOTE: Dates Mode changed from system  to NULL -- retrieval time less that %s days"), system.cal))
      # NULL datetime option
      sens2ref$UserDateINCal[i]   <- ""
      # UserDateENDCal  <- "" # This is not useful for now as we do not make the difference between UserDateINCal and UserDateINCal
    }
    # Convert when User does not defines dates for calibration empty string or "System"
    if (sens2ref$UserDateINCal[i] == "" ){# I removed & UserDateENDCal == ""that we so not check for now
      cat("[main] INFO, SET TIME parameters for Calibration functions for NULL dates")
      sens2ref$DateINCal[i]    <- sens2ref$DateIN[i]
      sens2ref$DateENDCal[i]   <- sens2ref$DateEND[i]
    } else if (sens2ref$UserDateINCal[i] == "System"){
      cat("[main] INFO, SET TIME parameters for Calibration functions system dates")
      sens2ref$DateINCal[i]  <- sens2ref$DateIN[i]
      sens2ref$DateENDCal[i] <- sens2ref$DateINCal[i] + days(system.cal)
    }
  }


  #------------------------------------------------------------------
  # Set initial and final dates to start measuring functions values for UserDateINmeas (after calibration)
  #------------------------------------------------------------------
  if(sens2ref$UserDateINmeas[i] != "System" & sens2ref$UserDateINmeas[i] != ""){
    #UserDateENDmeas    <- as.character(now(tzone = TimeZone)- 60) # MGV set it to "2016-01-01 00:00", I hope now() will more convenient.
    # If we use now() rather than now()-60sec, UserDateENDmeas is later than UserDateEND.0, which returns an error wehn checking date consistency
in main code
    sens2ref$DateINmeas[i]  <- as.POSIXct(sens2ref$UserDateINmeas[i] , format = "%Y-%m-%d %H:%M", tz =TimeZone )
    sens2ref$DateENDmeas[i] <- as.POSIXct(sens2ref$UserDateENDmeas[i], format = "%Y-%m-%d %H:%M", tz =TimeZone )
  } else {
    #------------------------------------------------------------------
    # Set dates to start measuring functions when User does not defines dates for calibration empty string or "System"
    #------------------------------------------------------------------
    if (sens2ref$UserDateINmeas[i] == "" ){
      cat("[main] INFO, SET TIME parameters for Measuring functions for NULL dates")
      sens2ref$DateINmeas[i]  <- sens2ref$DateIN[i]
      sens2ref$DateENDmeas[i] <- sens2ref$DateEND[i]
    } else if (sens2ref$UserDateINmeas[i] == "System"){
      cat("[main] INFO, SET TIME parameters for Measuring functions system dates")
      sens2ref$DateINmeas[i]  <- sens2ref$DateENDCal[i]
      sens2ref$DateENDmeas[i] <- sens2ref$DateEND[i]
    }
  }
```

```
#-----------------------------------------------------------------
# Set time zone to UTC  and Check dates consistance
#-----------------------------------------------------------------
attr(sens2ref[,c("DateIN")]    , "tzone") <- "UTC"
attr(sens2ref[,c("DateEND")]   , "tzone") <- "UTC"
attr(sens2ref[,c("DateINCal")] , "tzone") <- "UTC"
attr(sens2ref[,c("DateENDCal")] , "tzone") <- "UTC"
attr(sens2ref[,c("DateINmeas")] , "tzone") <- "UTC"
attr(sens2ref[,c("DateENDmeas")], "tzone") <- "UTC"

if(sens2ref$UserDateIN[i]     != "" & sens2ref$UserDateIN[i]     != "System"  &
  sens2ref$UserDateINCal[i]  != "" & sens2ref$UserDateINCal[i]  != "System"  &
  sens2ref$UserDateINmeas[i] != "" & sens2ref$UserDateINmeas[i] != "System"){

   cat("Checking dates consistancy for User defined dates", sep = "\n")
   if(sens2ref$DateIN[i]  >= sens2ref$DateEND[i]){
      stop("[main] ERROR DateIN.0 >= Today ")
   }
   if(sens2ref$DateINCal[i] >= sens2ref$DateENDCal[i]){
      stop("[main] ERROR DateINCal >= DateENDCal ")
   }
   if(sens2ref$DateINmeas[i] >= sens2ref$DateENDmeas[i]){
      stop("[main] ERROR DateINmeas >= DateENDmeas ")
   }
   if(sens2ref$DateINCal[i] < sens2ref$DateIN[i] | sens2ref$DateENDCal[i] > sens2ref$DateEND[i] ){
      cat(paste0("  DateIN.0: ",sens2ref$DateIN[i],"   DateEND.0: ",sens2ref$DateEND[i], "    DateINCal: ",sens2ref$DateINCal[i],"   DateENDCal: ",sens2ref$DateENDCal[i]), sep = "\n")
      stop("[main] ERROR Calibration dates does NOT stand within retrieval range ")
   }
   if(sens2ref$DateINmeas[i] < sens2ref$DateIN[i] | sens2ref$DateENDmeas[i] > sens2ref$DateEND[i] ){
      cat(paste0("  DateIN.0: ",sens2ref$DateIN[i]," DateEND.0: ",sens2ref$DateEND[i],"  DateINmeas: ",sens2ref$DateINmeas[i],"  DateENDmeas: ",sens2ref$DateENDmeas[i] ), sep = "\n")
      stop("[main] ERROR Dates for Modelling NOT within retrieval range ")
   }
   cat("[main] INFO, periods of available data, calibration period and extrapollation method are consistent", sep = "\n")
}

#-----------------------------------------------------------------
# reporting of the periods of available data, calibration period and extrapollation method
#-----------------------------------------------------------------
cat(paste0("Sensor: ", sens2ref$name.sensor[i]), sep ="\n")
cat(paste0("[main] INFO: available data from DateIN      = " , format(sens2ref$DateIN[i]     , "%Y-%m-%d %H:%M:%S"), " to DateEND      = " , format(sens2ref$DateEND[i]    , "%Y-%m-%d %H:%M:%S")), sep = "\n")
cat(paste0("[main] INFO: calibration     from DateINCal  = " , format(sens2ref$DateINCal[i] , "%Y-%m-%d %H:%M:%S"), " to DateENDCal  = " , format(sens2ref$DateENDCal[i] , "%Y-%m-%d %H:%M:%S")), sep = "\n")
cat(paste0("[main] INFO: extrapolation  from DateINmeas = " , format(sens2ref$DateINmeas[i], "%Y-%m-%d %H:%M:%S"), " to DateENDmeas = " , format(sens2ref$DateENDmeas[i], "%Y-%m-%d %H:%M:%S")), sep = "\n")
```

```
        if(any(grepl(pattern = "General.prev", x = objects(DownloadSensor)))){
          remove(DownloadSensor)
        }
      }
    }

    #=================================================================
    # 13. SET  Models for NO2 and O3 system resolution
    #=================================================================
    # Apply Etalonnage and Cal_line for all gases?
    ### All Shuny inputs Window 7
    Cal.Line          <- TRUE
    # function model for NO2 and O3 to find coefficient for linear solving
    CO.NO.NO2.O3.model  <- TRUE     # do you want the NO2.O3 model to be applied?
    LM.model.lab.coeff  <- FALSE    # If TRUE Coefficients for LM model are provided from lab-tests
    # if FALSE -> fitting
    # if LM.model.lab.coeff = TRUE, the following needs to be specified
    file.coeff.dir  <-  paste0(DisqueFieldtestDir,"lab_test_coeff/")
    ### END Shuny inputs

    # Saving config file
    write.csv(sens2ref, file = file.path(DisqueFieldtestDir,"General_data",paste0(AirsensEur.name,".cfg")))
}
```

List of abbreviations and definitions

Acronym	Description
ASE	AirSensEUR object identifier
EEA	European Environment Agency
EUPL	European Union Public Licence
F/OSS	Free and Open Source Software
INSPIRE	Infrastructure for Spatial Information in the European Community
ISO	International Standardisation Organisation
JSON	JavaScript Object Notation data interchange format
MS	Member States of the European Union
O&M	Observations and Measurements
OGC	Open Geospatial Consortium
REST	Representational state transfer
SEIS	Shared Environmental Information System
SOS	Sensor Observation Service
SWE	Sensor Web Enablement
XML	Extensible Markup Language

List of figures

List of tables

References

- Swan, M. (2012). Sensor mania! the internet of things, wearable computing, objective metrics, and the quantified self 2.0. Journal of Sensor and Actuator Networks, 1(3), 217-253.
- Brovelli, M. A., Mitasova, H., Neteler, M., & Raghavan, V. (2012). Free and open source desktop and Web GIS solutions. *Applied Geomatics*, *4*(2), 65-66.
- Nüst, D., Stasch, C. and Pebesma, E. J. Connecting R to the Sensor Web in Geertman, S.; Reinhardt, W. and Toppen, F. (Eds.) Advancing Geoinformation Science for a Changing World, Springer Lecture Notes in Geoinformation and Cartography, 2011, 227 – 246
- Kotsev, A.; Pantisano, F.; Schade, S.; Jirka, S. Architecture of a Service-Enabled Sensing Platform for the Environment. *Sensors* **2015**, *15*, 4470-4495.
- Gerboles, M., Spinelle, L., & Signorini, M. (2015). *AirSensEUR: An Open Data/Software/Hardware Multi-Sensor Platform for Air Quality Monitoring. Part A: Sensor Shield*. EUR Technical Report JRC97581.

GETTING IN TOUCH WITH THE EU

In person

All over the European Union there are hundreds of Europe Direct information centres. You can find the address of the centre nearest you at: http://europea.eu/contact

On the phone or by email

Europe Direct is a service that answers your questions about the European Union. You can contact this service:

- by freephone: 00 800 6 7 8 9 10 11 (certain operators may charge for these calls),

- at the following standard number: +32 22999696, or

- by electronic mail via: http://europa.eu/contact

FINDING INFORMATION ABOUT THE EU

Online

Information about the European Union in all the official languages of the EU is available on the Europa website at: http://europa.eu

EU publications
You can download or order free and priced EU publications from EU Bookshop at: http://bookshop.europa.eu. Multiple copies of free publications may be obtained by contacting Europe Direct or your local information centre (see http://europa.eu/contact).

KJ-NA-28928-EN-N

JRC Mission

As the science and knowledge service of the European Commission, the Joint Research Centre's mission is to support EU policies with independent evidence throughout the whole policy cycle.

EU Science Hub
ec.europa.eu/jrc

@EU_ScienceHub

EU Science Hub - Joint Research Centre

Joint Research Centre

EU Science Hub

Publications Office

doi:10.2760/93699

ISBN 978-92-79-77049-4