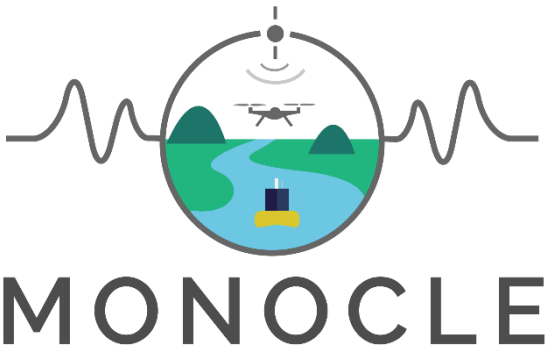
Project	MONOCLE H2020 (grant 776480)	**Start / Duration**	1 February 2018/ 48 Months
Dissemination	PUBLIC	**Nature**	REPORT
Date	08 Mar 2021	**Version**	1.0



Multiscale Observation Networks for Optical monitoring of Coastal waters, Lakes and Estuaries

Deliverable 5.3

D5.3 - System user and developer handbook

Project Description

Funded by EU H2020 MONOCLE **creates sustainable *in situ* observation solutions for Earth Observation (EO) of optical water quality in inland and transitional waters.** MONOCLE develops essential research and technology to lower the cost of acquisition, maintenance, and regular deployment of *in situ* sensors related to optical water quality. The MONOCLE sensor system includes handheld devices, smartphone applications, and piloted and autonomous drones, as well as automated observation systems for e.g. buoys and shipborne operation. The sensors are networked to establish interactive links between operational Earth Observation (EO) and essential environmental monitoring in inland and transitional water bodies, which are particularly vulnerable to environmental change.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 776480

Project	MONOCLE H2020 (grant 776480)	**Start / Duration**	1 February 2018/ 48 Months
Dissemination	PUBLIC	**Nature**	REPORT
Date	08 Mar 2021	**Version**	1.0

Deliverable Contributors:	Name	Organisation	Role / Title
Deliverable Lead	Darren Snee	PML	WP5 lead developer
Contributing Author(s)	Steef Peters	WI	WP5 lead
	Stefan Simis	PML	Contributing author
	Liesbeth de Keukelaere	VITO	Contributing author
	Bart Ooms	VITO	Contributing author
	John Wood	PEAK	Contributing author
	Jaume Piera	CSIC	Contributing author
	Carlos Rodero	CSIC	Contributing author
	James Sprinks	EW	Contributing author
	Norbert Schmidt	DDQ	Contributing author
Reviewer	Stefan Simis	PML	Scientific coordinator
Final review and approval	Stefan Simis	PML	Scientific coordinator

Document History:

Release	Date	Reason for Change	Status	Distribution
0.1		Initial outline	Draft	Internal
0.2		Drafts and Review Ch 1-5	Draft	Internal
0.3		Updated Ch1-5 and draft Ch 6	Draft	Internal
1.0	8 Mar 2021	First release	Final	Public

To cite this document:

Snee D., Peters, S., Simis, S., De Keukelaere, L., Ooms, B., Wood, J., Piera, J. (2020). D5.3 System user and developer handbook. *Deliverable report of project H2020 MONOCLE (grant 776480)*. doi: 10.5281/zenodo.4589028 (this version)

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 776480

TABLE OF CONTENTS

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 776480

Project	MONOCLE H2020 (grant 776480)	**Start / Duration**	1 February 2018/ 48 Months
Dissemination	PUBLIC	**Nature**	REPORT
Date	08 Mar 2021	**Version**	1.0

1. Executive Summary

This report provides an overview of the data requirements and interfaces that were adopted in the sensors and observation platforms of MONOCLE. The methodology builds on Open Geospatial Consortium service standards wherever possible. This document describes both the functionality and the configuration of the MONOCLE backend and distributed sub-systems.

In Chapter 3, we present the operating principles of the data ecosystem and the minimum requirements for data services front- and backends and metadata that follow from these principles. This chapter also introduces a generic terminology compliant with the standards of the Open Geospatial Consortium (OGC). Chapter 5 presents, for each sensor (system), which data and user interfaces have been configured to ensure the flow of data from sensors through quality control and wider availability of the data in an automated fashion. Open source and interoperable components are selected wherever feasible, and alternatives are discussed. Chapter 6 and Appendix 1 are intended for system developers wishing to replicate part of the data and interface structure. It provides a stepwise guide to the configuration of essential system elements.

2. Scope

This handbook is for current and future developers of the MONOCLE system backend and sensors compatible with the standards used in this system. We discuss a series of implementation targeting specific improvements in MONOCLE for a streamlined flow of information from individual sensors through data aggregation systems and towards visualisation tools. It is intended to shorten development time for sensor developers who wish to reach compliance with these recommended standards.

3. Introduction

3.1. The MONOCLE data ecosystem

The MONOCLE data ecosystem is being developed to demonstrate how in situ sensors and other data sources can be connected in near real time (NRT) so that data are efficiently shared between data producers and data consumers.

The objectives of this development are to provide a network of sensors and data nodes which can fulfil the following *operating principles*:

- Allow operators to control their sensors with ease both in situ and remotely
- Reduce operator involvement in distributing sensor data to accessible locations
- Provide capabilities to inspect, verify and correct observation data
- Deploy standardized machine-to-machine interfaces wherever possible

MONOCLE includes sensors and platforms that encompass a diversity of deployment procedures and data types, ranging from mobile applications to imagery from drones and high-frequency automated systems, in reachable and remote locations and operated by experts and non-experts. Because the sensors and platform span a range of complexity in data gathering modes, data types and data transfer mechanisms, it is our ambition that this guide will aid future developers in choosing appropriate mechanisms.

The sensor network and interfaces that were chosen for each component of the network were originally foreseen to organise along the data flows illustrated in Figure 1, with network nodes connecting along data interoperability standards of the Open Geospatial Consortium (detailed further below). Some of the benefits of building around these self-describing data standards are:

- In-built validation, versioning and traceability of data offerings.
- Responsibility to provide up-to-standard data offerings lies with the developer/operator.
- Tested templates can be widely re-used.

However, there are potential drawbacks to consider:

- Small recurrent data offerings can be seen to have disproportionate data transfer overhead.
- Complex data offerings require templates which are not straightforward to manage.
- Training may be required to construct sensor and result templates.

Our development has followed the principles of aiming to develop along OGC standard services, while allowing for bespoke APIs or hybrid solutions when these are more feasible. The guiding principle was to ensure wider uptake in the industry of data interoperability, and to avoid undocumented data exchange mechanisms.

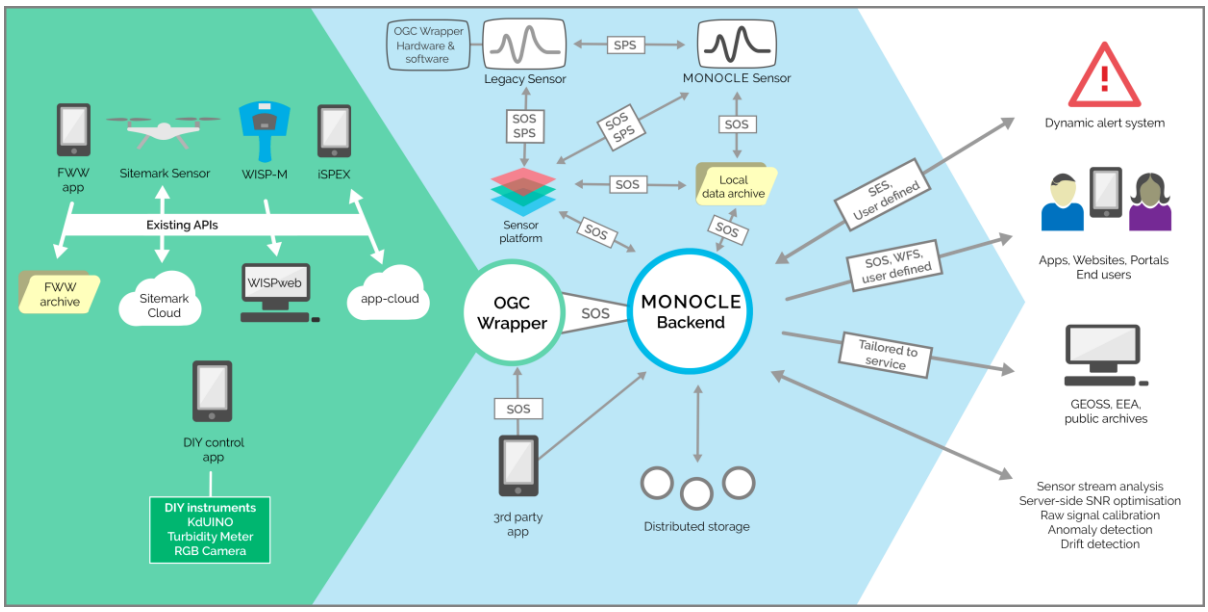


Figure 1. Example data flows from individual sensors to the MONOCLE backend and onward to data consumers.

The individual systems are presented in detail in Chapter 4, including their currently available interfaces for accessibility and data exchange. In many cases, a hybrid solution was formed around a

bespoke sensor-to-server data transfer mechanism with minimal data transfer overhead, followed by a standardized data service to expose functionality for mapping, cataloguing and individual data access.

3.2. Minimum metadata requirements

MONOCLE systems (in situ sensors, platforms, databases and software) were primarily developed to optimize satellite calibration and validation for water quality monitoring. This requires that **in situ data are associated with accurate geopositioning (including time) information** with sufficient accuracy to relate observations at relevant spatial scales**.** The accuracy of geo-position information should be recorded with the data, by specifying the source from which it was derived, such as satellite positioning, internet time protocols or a manual selection based on maps.

Results of a survey held among water quality practitioners (Heard et al. 2018) revealed a wide range of stakeholders who are responsible for in situ monitoring, with varying requirements for sharing, ownership or the data collected. It is thus required that **data ownership and licensing are added to the observation metadata from the first point of distribution and doing so in an interoperable manner is the responsibility of the data creator**.

Because the in situ data should be usable in automated procedures in near real time, while observation data may be distributed to multiple data stores, all observation data should **include a unique (set of) identifiers.** It is *recommended that individual observations, sensors and platforms or deployments are given unique identifiers for optimal traceability and quality control*. For example, if a sensor malfunction or drift is detected later, all observations from a given deployment can then be marked as suspect.

To support quality control procedures for new observations, upon new calibration results or when new drift or anomaly analysis results are made available, data managers and users require traceability to calibration measurements, labs and standards with each observation. Therefore, the **processing level, sensor calibration time, calibration software version and revision history of post-processing** should be identifiable in the data backend.

The minimum metadata requirements for connecting a sensor system to the MONOCLE backend and suggestions on how to populate these fields are summarized in Table 1. We note that these recommendations follow existing standards (e.g. ISO as indicated) where these have been identified, whilst overall these requirements should be considered a *recommendation* to fulfil the *requirements* of a wide variety of downstream data uses. Wider testing of this set of metadata would be required before recommendations to arrive at new or extended standard could be made.

Table 1 Minimum metadata requirements to enter new observations into the MONOCLE system.

Category	Element	Description	Possible values, data type, conventions, units
Location & time	Latitude	Geographic location	*Float* decimal degrees, north positive
	Longitude	Geographic location	*Float* decimal degrees, east positive
	Elevation	Height above reference ellipsoid	*Float* in meters

Category	Element	Description	Possible values, data type, conventions, units
	(alternative: Altitude)	(Alternative: height above ground)	
	Reference Coordinate System		Default WGS84
	Time	Time in Coordinated Universal Time (UTC)	Character string formatted according to ISO8601
	Location_source	Source of the Geodetic information	e.g. GNSS
	Time_source	Source of the Time information	e.g. GNSS, internet time pool
Processing	Processing_level	Sensor-specific	*Integer* 0, 1, 2 … n or *String* including sublevels such as 1A, 1B, 1C. Defined by manufacturer and described in the reference documentation. Level 0 is uncalibrated sensor output and not distributed; Level 1 is calibrated data prior to any corrections or interpretation; Level 2 is interpreted data; Level 3 is aggregated or regridded data.
	Processing_procedure	Reference to protocols and algorithms describing the steps involved in data processing	URL
	Processing_version	Version of the data processing software	Free form, recommended: major.minor.build
	Processing_revision	Incremental version of the processed data	Free form, likely an integer
	Calibration_procedure	For calibrated data: documentation describing the calibration procedure. Can be the same as Processing procedure reference	URL
	Calibration_reference	Identifier of calibration information	Flexible, system-specific
	Calibration_time	Date/time stamp of applicable (uncalibrated data, if available) or applied (calibrated data) sensor calibration.	Character string formatted according to ISO8601
	Calibration_version	Version of the calibration processing software	Free form
Identifiers	Sensor_id	Unique identifiers used to prevent data duplication with data consumers	Sensor serial number (manufacturer decides format)
	Platform_id		Platform serial number or randomly assigned identifier (UUID) used with all connected sensors. May be left empty if not applicable.
	Deployment_id		Randomly assigned identifier (UUID) specific to deployment sequence (e.g. cruise, campaign, vertical profile) of **a specific** sensor. Not shared with other sensors.
	Sample_id		Randomly assigned identifier (UUID) generated with each distinct data record from any **set** of sensors belonging to a single observation.
	Observer_id		Randomly assigned identifier (UUID) repeated with each data record from this and/or other sensors when operated by a **specific** observer.
Licensing	Owner_contact	An email address where the owner of the data can be contacted now and in future	Sustained email address (e.g. data@organisation.org rather than individual@organisation.org)

Category	Element	Description	Possible values, data type, conventions, units
	Operator_contact	An email address where the current operator can be contacted	Email address to operator or group of operators
	License	A licence string or coding that is either self-explanatory or detailed in the License_reference field.	Free form
	License_reference	A reference describing the data license in detail.	URL
	Embargo_date	A date following which the data may be used according to the specified license. Used, for example, to hide the data record in NRT visualization until quality control is completed.	Character string formatted according to ISO8601

3.3. Minimum system requirements

The operating principles and minimum data requirements can be translated into requirements for the MONOCLE data ecosystem. In particular, the following components are required:

- A **data store** consisting of one or more distributed servers that can ingest and serve data through standardized interfaces. The data store should meet the following requirements:
  - Data should be stored securely to avoid unauthorized changes
  - Updates (e.g. calibrations) and corrections to the data should be traceable
  - Data should be archived to back up locations
  - Data should be accessible, honouring data licenses and embargoes
- **Visualization tools** with GIS capabilities to inspect and interact with multi-source data
- The system should be **scalable** either within the current context, or set up so that it can be migrated to cloud-based servers and storage in future
- A layer of distributed **data consumer processes, demonstrating trigger and alert mechanisms;** e.g. report generation or alerting in situ sensor operators.

MONOCLE has invested large effort into improving the interoperability between elements of the system (from sensors to data stores and server-based analysis) through the use of standardized data interfaces, where feasible. These standards are curated by the Open Geospatial Consortium (OGC) and listed in the next section. In chapter 4 the choice of standards for each system component is explained in more detail. Chapter 5 then details how these interfaces are used in practise while Chapter 0 details how they were implemented.

3.4. Terminology

The following overview is included to avoid confusion between observation disciplines and the standard terminology used in the OGC standards.

Feature Abstraction of a real-world phenomenon.

Measurement A set of operations having the object of determining the value of a quantity

Observed Property Facet or attribute of an object referenced by a name which is observed by a *procedure*.

Observation Act of observing a property.

Observation Offering An Observation Offering groups collections of observations produced by one procedure, e.g., a sensor system, and lists the basic metadata for the associated observations including the observed properties of the observations.

Procedure Method, algorithm, instrument, sensor, or system of these which may be used in making an observation

Sensor Entity that provides information about an observed property as its output. A sensor uses a combination of physical, chemical or biological means in order to estimate the underlying observed property. At the end of the measuring chain electronic devices produce signals to be processed.

Sensor System System whose components are sensors. A sensor system as a whole may itself be referred to as a sensor with an own management and sensor output interface. In addition, the components of a sensor system are individually addressable. In MONOCLE these can also be referred to as 'platforms'.

The OGC service standards referenced in this report are listed in Table 2.

Table 2 Overview of OGC standards

Standard	Scope
SOS	HTTP based standard for the querying and addition of sensor observations and metadata using a variety of possible bindings and formats. See https://www.ogc.org/standards/sos
WFS	HTTP based standard for the access and manipulation of geographic feature data . See https://www.ogc.org/standards/wms
WMS	HTTP based standard for requesting geo-referenced map images from a server See https://www.ogc.org/standards/wms
WCS	HTTP based standard receiving of geospatial information as 'coverages': digital geospatial information representing space-varying phenomena. The results of a WCS can be used for complex modelling and analysis, and allows complex querying – users can extract just the portion of the coverage that they need. See https://www.ogc.org/standards/wcs

4. System overview

The MONOCLE system demonstrates how in situ sensors can be connected in near real time so that data are efficiently shared between data producers and data consumers.

4.1. Distributed sensors

This section describes the types of sensors included in MONOCLE, which are connected to the data back-end through the various interfaces described further below. These sensors include:

So-Rad The solar-tracking radiometry platform (So-Rad) by PML provides support for three (ir)radiance spectrometers to observe the remote-sensing reflectance (water colour) from moving platforms. It integrates automatic pointing of the radiometers to avoid sun glint. The software and hardware are fully open-source. More information

WISPStation The Water Insight SPectrometer Station measures water-leaving reflectance and derives key water quality parameters fully autonomously at high frequency, with results available in near real-time through the WISPCloud dashboard. More information

HSP-1 The HyperSpectral Pyranometer by Peak Design measures the spectrum of downwelling solar radiation and how this is partitioned between Direct, Diffuse and Global Irradiance. This sensor provides a reference for the colour or spectral distribution of sunlight near the water surface. More information

RPAS Collective name for Remotely Piloted Aircraft Systems or 'drones' equipped with native RGB cameras or additional payload to collect imagery of the water surface in high detail. More information

iSPEX 2 The iSPEX 2 developed by University of Leiden with app support by DDQ is a smartphone attachment to gather spectropolarimetric data by taking a picture of the water, the sky and a grey card for calibration. Data on the intensity and polarisation of light at different wavelengths is further analysed to derive water quality information. More information

KdUINO The original KdUINO, a moored instrument to measures the diffuse attenuation coefficient (K_d), has evolved into a KdUSTICK and a KdUMOD for low and medium-cost water transparency observations, respectively, in a portable package that can be deployed for the duration of a battery charge. More information

An overview of the data-generating properties of the sensors is given in **Error! Not a valid bookmark self-reference.** and their interfaces to collect configuration information, time and positioning information are given in **Error! Reference source not found.**.

Table 3 Overview of sensors and their typical data characteristics

Sensor	Sample rate	Data type	Data volume	Buffering
So-Rad	0.1 Hz when equipped with TriOS Ramses	3 x (ir)radiance spectrum of 193 x 32bit	Approx. 3 kB per sample including metadata	Local SQLite database

WISPstation	0.0055 Hz	6 x (ir)radiance and/or 1 x Rrs Spectra of 501 * 32bit	Approx 15kB per sample including metadata	Local SQLite database (only for unsent data)
HSP-1	1Hz max	2 x irradiance spectrum 700 x 32bit float	20MB for basic data or ~250MB for 1-min intervals with diagnostics	All data stored to local file storage
RPAS	0.3-1Hz	DJI RGB: 3x 8bit MicaSense: 5x16bit	8-10GB/flight	Local SD card
iSPEX	User triggered	Raw images (spectrum or RGB)	20MB/sample	Device Store/ Parse mobile backend
KdUINO (KdUSTICK)	1 Hz	Kd RGB	< 1 Kb per day	Local SD card
KdUINO (KdUMOD)	1 Hz	Kd multispectral, temperature	< 1 Mb per day	Local SD Card

Table 4 Overview of sensors and their interfaces for remote management, time and location data.

Sensor	Remote management	Remote triggers	Time source	Position source
So-Rad	SSH over reverse tunnel	Through periodic config update	Observations: GNSS System/logs: Internet Time Pool	GNSS
KdUINO (KdUSTICK)	Wifi – Bluetooth over Mobile App / Uplink over IoT	n/a	GNSS / Wifi	GNSS
KdUINO (KdUMOD)	Wifi – Bluetooth over Mobile App / Uplink over IoT	n/a	GNSS / Wifi	GNSS
iSpex	Via mobile backend (ssh, JSON/CURL)	Push/cron via mobile backend	GNSS/Internal clock	GNSS
HSP-1	Login via VNC or RemoteUtilities	n/a	GNSS and internet time pool	GNSS
Wisp-M / WISPstation	SSH over reverse tunnel	Configurable in crontab (also remote)	Integrated high-accuracy RTC	Google maps
RPAS	Timer mode, overlap mode, external trigger mode (PWM, GPIO, serial, and Ethernet options), manual capture mode.		GPS	GNSS

4.2. Distributed data stores

Several sensors and sensor systems (platforms) feed into intermediary data stores. MONOCLE also connects to a number of data stores that are fed by manual uploads of data records from research

missions and citizen science initiatives. An overview of these distributed data stores in given in Table 5. The majority use geospatial databases to facilitate geographical and temporal filtering of the data.

Table 5 Overview of data services and databases

Data source	Type	Interfaces	Provider and URL
WISPcloud	PostgreSQL	Proprietary API, SOS connector	Water Insight https://wispcloud.waterinsight.nl
FreshWaterWatch	Geoserver Database, version 2.17.1	WFS connector	Earthwatch (geo.earthwatch.org.uk)
LIMNADES	Geoserver	WFS connector	University of Stirling (limnades.stir.ac.uk)
iSPEX backend	MongoDB JSON / based database	JSON/SSH/CURL/Online Dashboard	DDQ
MapEO-Water	Geoserver	WMS, WCS	VITO https://maps.vito.be/geoserver/web/

4.3. MONOCLE central data backend

The sensor systems and services hosted around the data stores described in the previous sections feed into a 'backend' system. Other than for demonstration purposes, the backend need not be a single hosted entity, it can be distributed across several nodes all hosting compatible data services, and hosted either privately or public, or both. For demonstration purposes, PML hosts instances of an SOS and THREDDS server to receive and distribute data from sensor systems and middleware. In addition, data analysis and visualisation nodes are set up to connect to any of the distributed data stores. The backend services and the standards set up for this system are:

- SOS backend: receiving SOS input
- SOS Proxy: adding a security authentication layer between external sources and backend
- THREDDS data server: hosting gridded data offered, inter alia, as WMS and WCS

Details on these system components are given in Chapter 6.

4.4. MONOCLE front-ends

There are many possibilities to offer front-end (user facing) system components once data have been ingested into interoperable data services. For example, open-source desktop software such as QGis can connect to any publicly available WMS offering to rapidly deploy mapping tools. In the MONOCLE ecosystem we identify several defined front-ends either connecting directly to middleware or the backend:

- LIMNADES – frontend offering user registration and data statistics as well as mapping tools against a *Geoserver* (middleware) instance.
- PML WebGIS – an open-source WebGIS compatible with WCS and WMS (WFS being implemented) exposing all MONOCLE streams connected to the backend.

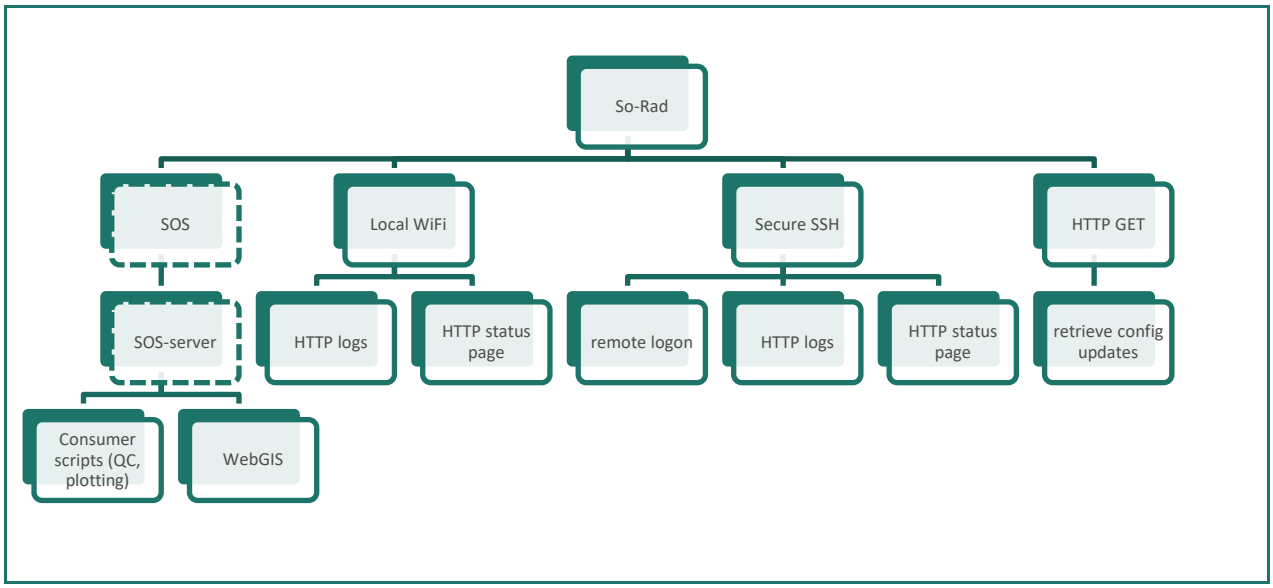
- PML SOS – including the Helgoland front-end for operators to verify data input into the SOS backend.
- Water Insight WISPcloud API + Lizard GIS: Cloud based data warehouse and analytics platform communicating with the Water Insight API (middleware).
- Earthwatch FreshWater Watch Waterhub - Data explorer with mapping and data set analytics built against the FWW middleware.

5. User guide

This section describes the exposed functionalities (the flow of data from sensor to frontend) and the interface components, for each of the major MONOCLE data flows.

5.1. So-Rad

So-Rad: Solar Tracking Radiometry platform
Location: Sensor platform (multiple sensors) PML **Usage**: Remote operation on ships, buoys
Purpose
The So-Rad controls the operation of up to three spectroradiometers, GPS, heading and tilt/pitch/roll sensors. Its main use is to optimize and record the viewing angles of spectroradiometric measurements with respect to the solar azimuth, on moving platforms. It is built to provide a low-cost automation solution for in situ reflectance spectroradiometry including optimized power consumption and bandwidth. It is an open source hardware/software platform which can be extended with updated sensors and components.
Data flow
The So-Rad is configured to send uncalibrated sensor data to the backend. A local sqlite database provides for data buffering and can be downloaded manually or synchronized with file-sharing services as an alternative to using per-sample data transmission. SOS is the preferred standard but alternatives are also considered (see below).
Functional system components
The main So-Rad data interfaces are as shown in the diagram below.



There are three primary interfaces to communicate with the So-Rad. In this description, 'local' refers to the sensor system and 'remote' is any other component or user, such as the MONOCLE back-end.

1. **SOS** is intended to connect to a remote SOS-server in the MONOCLE back-end, or any other configured SOS server. On first deployment an authentication is requested and stored in a local configuration file. The So-Rad will periodically interrogate the remote data store to identify the latest successful data upload. New data are then submitted to the remote data store. All configuration options are set in a local config file.
2. **WiFi** is used to allow on-site operators to connect to the So-Rad over SSH for maintenance and configuartion. Additionally, a web service exposes recent log files and a status page, showing the last 10 database entries. The WiFi network is password protected and upon connecting the operator can access the So-Rad and HTTP pages through its IP address.
3. **SSH** is required to allow remote configuration, monitoring and updates and requires an LTE modem to be included in the So-Rad. Then, depending on the mobile network, either a static IP address is used or a reverse-SSH tunnel should be set up. In the latter case, a service such as Dataplicity can be used to connect to the So-Rad in the same manner as described for local WiFi access. Dataplicity includes a port forwarding service ('wormhole) to expose the status pages through a randomly generated URL.
4. **HTTP GET** is used to optionally retrieve updates to the So-Rad configuration from a secure (read-only) URL. This allows the operator to configure the URL on the So-Rad which will then periodically attempt to download updates. The range of settings that can be altered through the remote 'dead-drop' configuration update is limited to sampling rates and schedules, for added security**.**

With regard to SOS, we note that defining xml templates for sensors which offer multiple data types in a single observation is not straightforward. Alternatives to using SOS for data transfer include the use of bundling data into discrete files and using a file synchronization protocol. Alternatively, a more flexible data store such as Parse server can be used, either defined with MongoDB to accept any JSON

formatted input, or with a PostGres database backend for pre-defined schemas. All information that would be offered through SOS templates can be wrapped into JSON to be offered through a Parse (or equivalent) server, however the data will then need to be validated after transmission. Both options are currently being tested.

Example usage

The operator of a So-Rad platform will periodically deploy a calibrated sensor set on a platform of choice, such as buoy or ship. At the installation site, they will connect to the WiFi that is broadcast by the So-Rad to configure the system for the specific location: they will update the deployment identifier and operator contact details which are included in the metadata, and determine suitable sampling intervals and permissible sensor viewing angles.

Following deployment, the So-Rad will connect via an LTE modem whenever a network is available. Data are always first buffered in a local database, and uploaded to the SOS backend whenever possible. In addition, database files can be retrieved over SSH.

Should the operator wish to make changes to the sampling configuration, they can point the system at a URL of their choosing where they host the configuration file, for example at a file synchronization service which provides a public (read-only) sharing link. This configuration will then overwrite the one that is set on the local system.

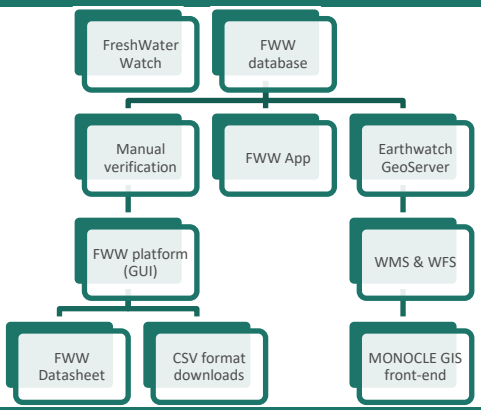
Data that have been uploaded to the backend will then be processed to Remote-sensing reflectance, triggered when new data are available. The uncalibrated observation records can be viewed through the Helgoland data visualisation front-end, which is primarily useful for the operator. Data users can browse and calibrate the calibrated data records through the WebGIS frontend (at the time of writing this interface is under development).

5.2. FreshWater Watch

FreshWater Watch Citizen Science Platform	
Location: Citizen Science Community and Platform Usage: Remote in-the-field measurements of river and catchment areas	Earthwatch
Purpose	
FreshWater Watch is a citizen science platform allowing members of the public to collect and submit river water quality data to an online data repository. Participants use either a smartphone app or downloadable datasheet to record and upload data. Measurements including colour, nitrate levels, phosphate levels and turbidity are taken using Secchi tubes, chemical testing kits and colour comparison charts. Registered users of the online platform can also download aggregated data in CSV format; selectable by geographic area, whilst curated WMS and WFS datasets are made available through a GeoServer instance.	
Connectivity	
Information and registration: **https://freshwaterwatch.thewaterhub.org/** Citizen scientist data upload through FreshWater Watch app: **https://play.google.com/store/apps/details?id=com.wk.android.fww&hl=en** or **https://apps.apple.com/gb/app/freshwater-watch/id882890751**	

Data upload also possible through datasheet:
https://freshwaterwatch.thewaterhub.org/sites/default/files/datasheet-blitz-en.pdf
Aggregated datasets (raw) available in CSV format:
https://freshwaterwatch.thewaterhub.org/our-data/explore-our-data
Aggregated datasets (validated) available in WMS and WFS formats from GeoServer:
https://geo.earthwatch.org.uk/geoserver/web/

Functional system components



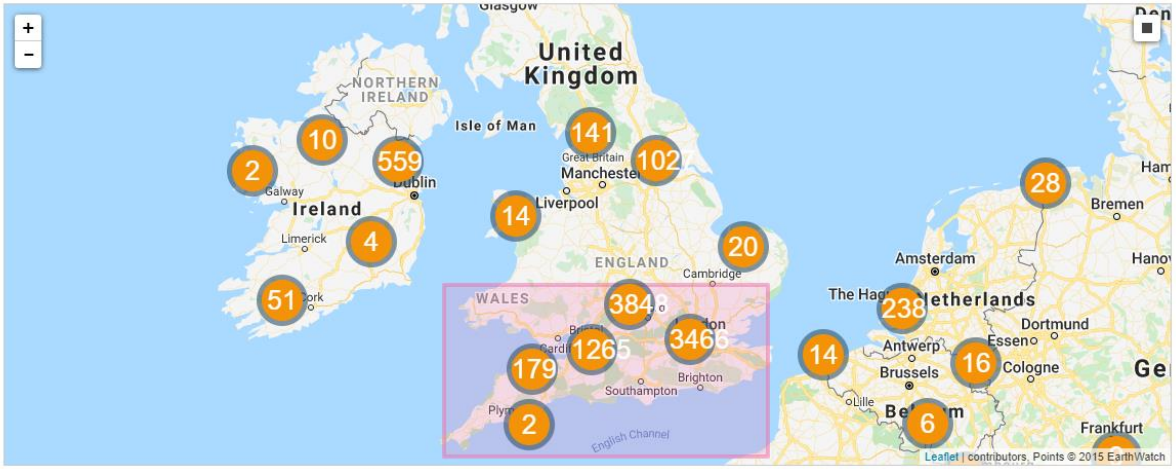
The Freshwater Watch system exposes the following functional components:

1. **FreshWater Watch App:** Citizen Scientist community use app to record and upload river water quality data to the FreshWater Watch back-end database.
2. **FreshWater Watch Datasheet:** Citizen Scientist community record data manually and upload onto the FreshWater Watch Platform. This data is then manually verified and added to the FreshWater Watch Database.
3. **FreshWater Watch Platform:** Exposes water quality datasets to registered users in a CSV format selectable by geographical extent.
4. **EarthWatch GeoServer:** Datasets from the Freshwater Watch Dataset are validated and curated for exposure on the EarthWatch GeoServer. This then provides open-access to WMS and WFS formats of the data, selectable as a geospatial layer.

Example usage

Requesting water quality data through the Freshwater Watch Platform (in CSV format)

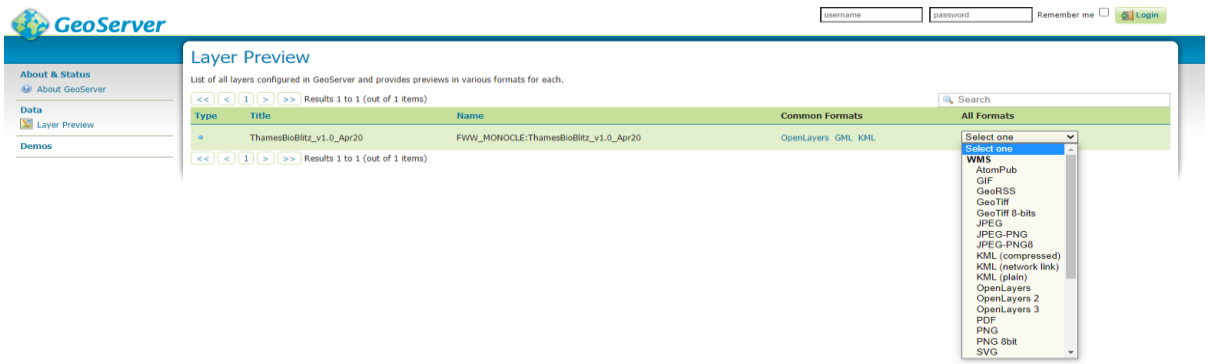
- Visit the FreshWater Watch Platform data page:
  https://freshwaterwatch.thewaterhub.org/our-data/explore-our-data
- Select the geographical area that you are interested in, by drawing a rectangle:



- Click on 'Export data' to download a CSV file to your local machine.

Requesting water quality data through the Earthwatch GeoServer WMS and WFS interfaces

- Visit the Earthwatch GeoServer page: http://geo.earthwatch.org.uk/geoserver/web/
- Select 'Layer Preview' from left hand menu.
- Select required format from drop-down menu:



- Data file will then be downloaded to your local machine in the requested format.

Data layers and their features can also be directly accessed by a GIS through GeoServer WFS and WMS services: https://docs.geoserver.org/latest/en/user/services/wfs/reference.html https://docs.geoserver.org/stable/en/user/services/wms/reference.html

5.3. KdUINO (KdUSTICK and KdUMOD)

KdUINO sensor / sensor system	
Location in system: KdUSTICK is a Sensor	CSIC

KdUMOD is a Sensor System
Usage: Buoy system (low-cost) equipped with light sensors

Purpose

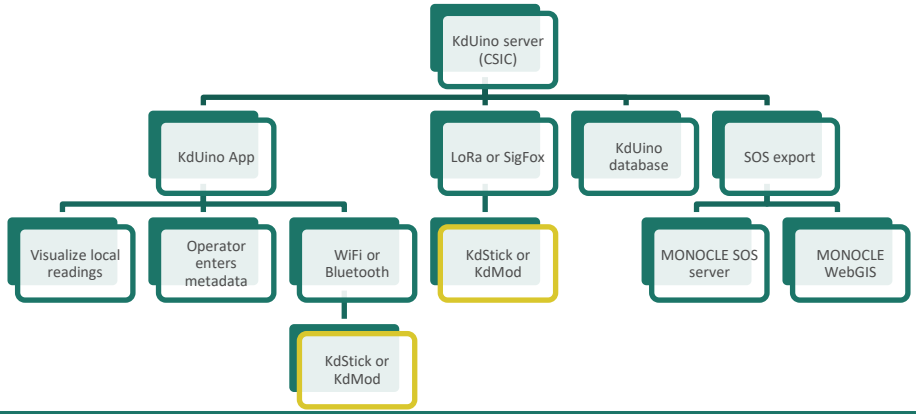
KdUSTICK obtain RGB light measurements at different depths (near the surface) to provide the broad-spectrum diffuse attenuation coefficient (Kd).

KdUMOD it is built as a versatile modular system. Each module placed at different depth has sensors for (a) multispectral light measurements and (b) temperature.

Connectivity

Both KdUINO lines can connect by WiFi/Bluetooth to the corresponding mobile App to be configured by the user. They are able to transmit data in real-time using Internet of Things (IoT) networks (LoRaWan and Sigfox). The data are first transmitted to CSIC servers, and from there to a SOS backend. The KdUINO products also store data on a local SD drive.

Functional system components



Once KdUSTICK is switched on it will repeat the following cycle of functionality until switched off:

- find GPS satellites to update position and time
- Collect RGB light measurements
- Calculate diffuse attenuation coefficient from available readings
- Send data to CSIC server using Internet of Things (IoT) protocol if available
- Possibility to send data from CSIC server to KdUSTICK devices whenever they ask for an update (downlink message)
- Save all data:
    - Light measurement from each sensor
    - Diffuse attenuation coefficient (Kd)
    - Temperature (in case of KdUMOD)

  and metadata to local SD storage, following the Ocean Sites specifications. This is an example of how it is structured: https://zenodo.org/record/3906019#.X9dBkNhKi70

- Enter deep-sleep mode for a specified time to preserve battery power

An operator can interact with the KdUSTICK using the mobile App through wifi/Bluetooth as long as the user is close and while the device is not in deep-sleep mode. The operator will then enter the metadata related to the deployment, configure the sampling frequency or load and visualize the data.

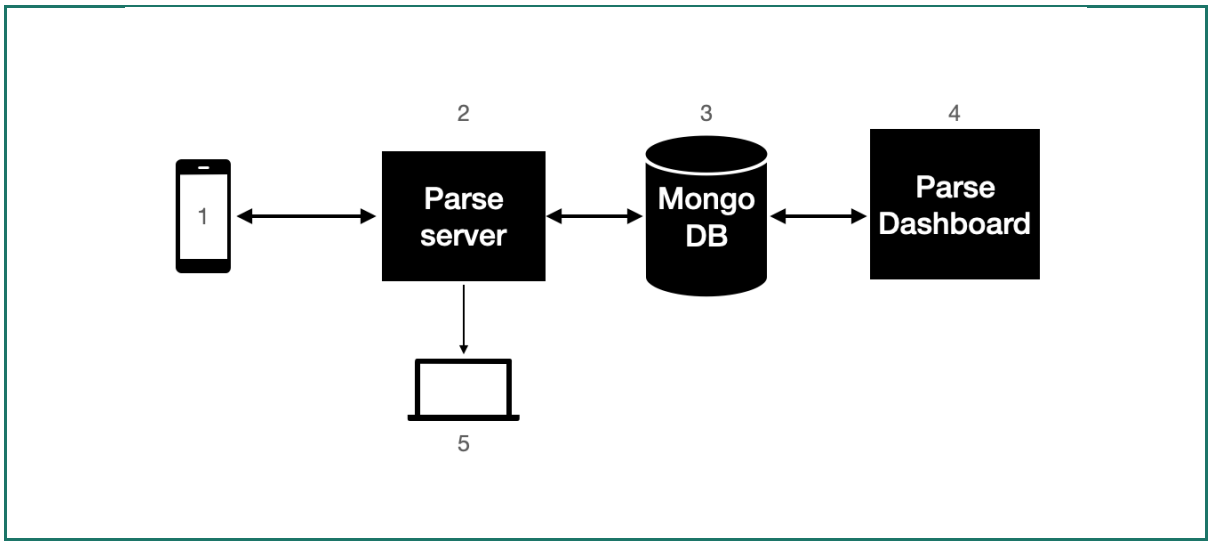
Example usage

Typically, the user/operator will deploy a KdUSTICK by switching it on, configuring it using the mobile app, casting it into the water body (1.5 - 2 m depth required) until several measurement cycles have completed, then recover the KdUSTICK and stop the measurements. The App guides the user through the specification of metadata and starts/stops the measurement cycle (to avoid measuring in air).

KdUSTICK will attempt to broadcast data to CSIC servers and from there to the SOS backend in real-time. If a connection is not possible, the user can retrieve data using the mobile app and send it to the servers later. A passive mode of exposing the data to the MONOCLE back-end is also being explored. In this case, a WMS/WFS capable server would be configured against the CSIC data store to handle requests from external front-ends such as the MONOCLE WebGIS.

5.4. ISPEX Mobile app with backend

ISPEX Mobile app with Parse server	
Location in system: Frontend/Backend	DDQ
Purpose	
The mobile app interacts with the operator/user and the sensor (phone camera). The middleware/backend is responsible for data storage, push notifications, offline synchronisation and user identification and credential management.	
Connectivity	
The iSPEX mobile app connects via JSON/API request to the parse back end service listening on https://parse.ddq.nl/parse. We also use Firebase (Google) for push notification integration. Required by Android devices.	
Functional system components	
1. Mobile app for collecting and sending sensor/user data. 2. Parse server (**www.parseserver.org**) is a nodeJS middleware package for MongoDB. 3. MongoDB, the default nosql storage solution for Parse server. 4. Parse dashboard (not a requirement, but a very versatile tool for looking at live datasets coming in from the smartphones) and scheduling of push notifications. 5. Operators can download the datasets by accessing the parse server using curl/json requests.	



Example usage

Mobile phones connected to the Parse server backend send their (meta-)data. An example of how these occur in the backend database is provided below. Sensor data and metadata (images, location, orientation) are collected, along with system information such as mobile device type. The operator can access this dashboard to inspect data coming in and to send push notifications and see log output (mobile based and server based).

CLASS
android_spectacledata 118 objects • Public Read and Write enabled

Add Row | Refresh | Filter | Security | Ed

objectId String	heading Number	ispex_on String	water_sky_selection	exposure Number	raw_pitch Number	model String
XXQMpyG6eU	44	false	water	971817	19.063481894637015	SM-G950F
Wy9CITiazo	208	false	water	500000000	95.89172509285305	SM-G920F
PPP758IAJb	201	false	sky	27688707	97.10050899656102	Pixel 2 XL
5zHP8COsqW	308	false	sky	27688707	141.14441957301486	Pixel 2 XL
UQj0E9Kdaq	98	false	sky	10369770	136.96320888905188	FP3
3lHDwCU1Xd	134	false	water	7462686	3.2133857685430067	SM-G920F
8n3NQ7dGaN	44	false	water	11627906	64.55081355258658	SM-G920F
VqdKRX9CEZ	1	false	water	3401360	26.177710796198586	SM-G920F

5.5. HSP-1

Name and type of component

Location in system: Sensor positioned on a buoy, ship or a fixed platform either on land or water.

Peak Design

Usage: Continuous recording of global & diffuse hyperspectral downwelling solar irradiance.

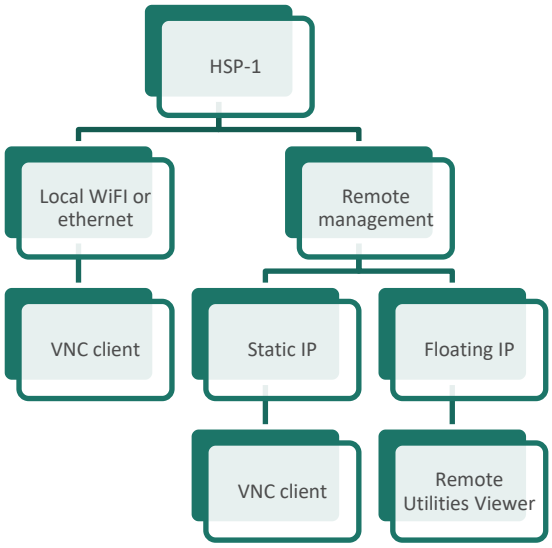
Purpose

HSP-1 measures the Global and Diffuse partition of downwelling solar radiation over the range 350nm – 1050nm. This can be used on its own or as a reference for other irradiance or reflectance measurements in a larger sensor system. Auxiliary readings of power supply voltages, internal temperature and humidity, GPS time and position, and dynamic orientation (Yaw, Pitch, Roll) can also be included.

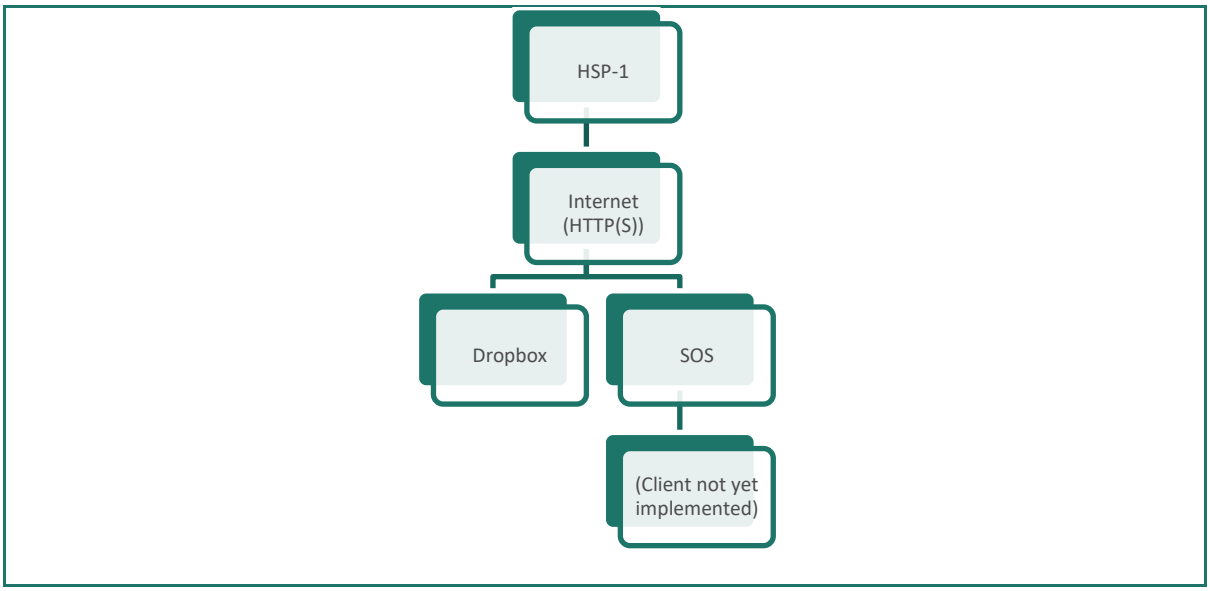
Data flow and connectivity

The HSP-1 needs a network connection for setup and data upload. A choice of direct ethernet connection or modem is available. WiFi can be used as an alternative, in suitable settings. HSP-1 runs on a Windows 10 miniature computer. If a static IP address is available a VNC client can be used for remote monitoring. With floating IPs, the Remote Utilities Viewer can be used. VNC can also be used by connecting to the sensor locally.

-



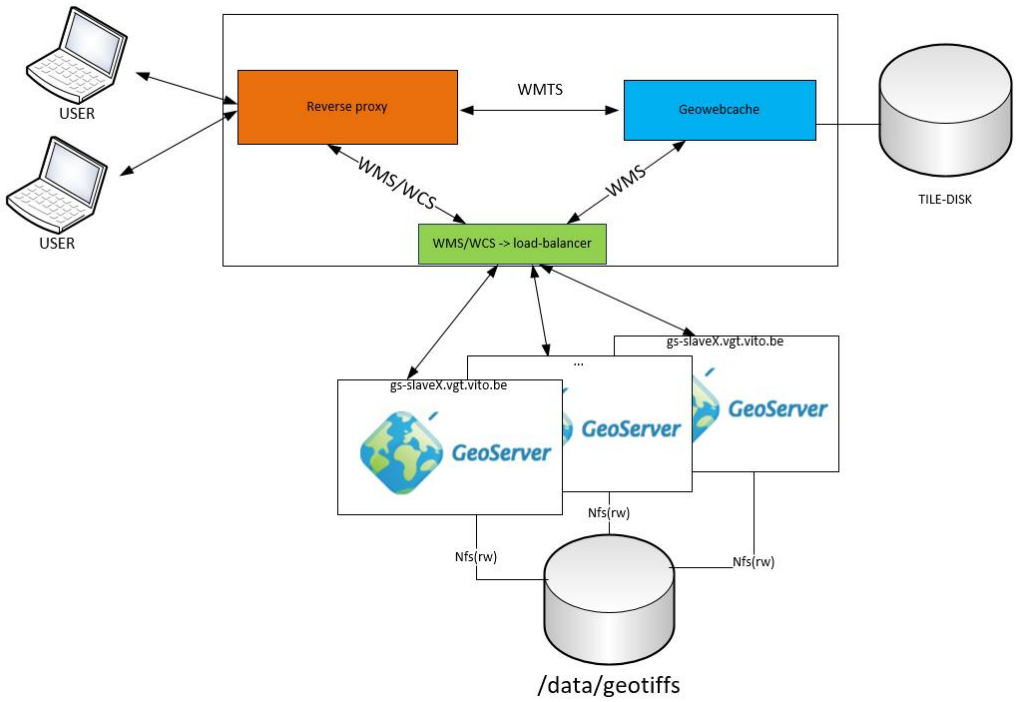
When the HSP is connected to the internet, two options for data synchronization are available. The first is through a Dropbox account using measurement files as data packets, which can then be processed by a client signed into the same Dropbox account. This procedure is more flexible than FTP because the file synchronization service takes care of incomplete uploads and resuming transfers. The second option uses SOS to make the HSP fully compliant with the MONOCLE specifications, wrapping each individual sample into a separate observation offering. The client for this data transfer is still being implemented (see chapter Error! Reference source not found. for discussion and details).



5.6. Remotely Piloted Aircraft Systems

RPAS – Remotely Piloted Aircraft System	
Location in system: Sensor platform (multiple sensors)	VITO
Purpose	
The purpose of data collection with RPAS is to construct maps of waterbodies from which water quality parameters can be derived. The RPAS systems may also serve as direct reference to satellite data, with the added advantage of detailing fine spatial features which can explain aberrations in processed satellite data, where fine features are not directly visible due to a large pixel size. The raw image data are too large (and not useful) to be disseminated beyond the data processing centres, where they are archived on suitable storage media (e.g. tape drives). Processed parameter-specific maps will be disseminated through the MONOCLE data back-end using machine interfaces (WMS/WCS).	
Connectivity	
In most cases, RPAS will not be directly connected to the backend. Instead, the operator will download the raw data to a laptop. Using a desktop application (called “fieldsoftware”), the operator can select the data, add required metadata and upload thethe data to the processing service, using provided credentials. The uploaded data will be processed through the MAPEO-Water workflow into water leaving reflectance and water quality parameters. The end products become available in a Geoserver and can be accessed and visualised by users in a frontend user interface or web application through WMS or WCS.	

Exposed functionality



Services: Web Map Service and Web Coverage Service

GeoServer is used to provide the WMS and WCS. It has been set up to scaled easily with growing data volumes and requests. On top of the GeoServer cluster a reverse proxy handles all incoming requests and will forward each single request to one of all available GeoServer instances. All the GeoServer instances share a storage volume for all geographical data. There is one master GeoServer, used for configuration, and several slave GeoServer instances to handle the incoming WMS/WCS requests.

Example usage

The OpenGIS® Web Map Service Interface Standard (WMS) provides a simple HTTP interface for requesting geo-registered map images from one or more distributed geospatial databases. A WMS request defines the geographic layer(s) and area of interest to be processed. The request response is one or more geo-registered map images (returned as JPEG, PNG, etc.) that can be displayed in a browser application. The interface also supports the ability to specify whether the returned images should be transparent, so that layers from multiple servers can be combined or not. It is possible to access the WMS directly by using a simple web browser or desktop tools such as QGIS.

The MONOCLE geoserver (authentication required):
http://dev.mapeo.be/geoserver/MONOCLE/wms?service=WMS&version=1.3.0&request=Getcapabilities

Example request:

(To make this work in the browser, first authenticate to the service using Getcapabilities request)

https://dev.mapeo.be/geoserver/MONOCLE/wms?service=WMS&version=1.3.0&request=GetMap&layers=MONOCLE:20190703_BalatonRGB-Test-20210218-224007_F01_RGB_TUR_sub&styles=&bbox=720257.0,5199643.0,720398.0,5199788.0&width=746&height=768&srs=EPSG:32633&format=application/openlayers

User manuals:

- WCS User manual
- WMS User manual

5.7. WISPstation / WISP-M

WISPstation / WISP-M: Water Insight Spectrometer station (semi-portable version)

Location in system: sensor platform — Water Insight
Usage: autonomously operating sensor at fixed position or buoy

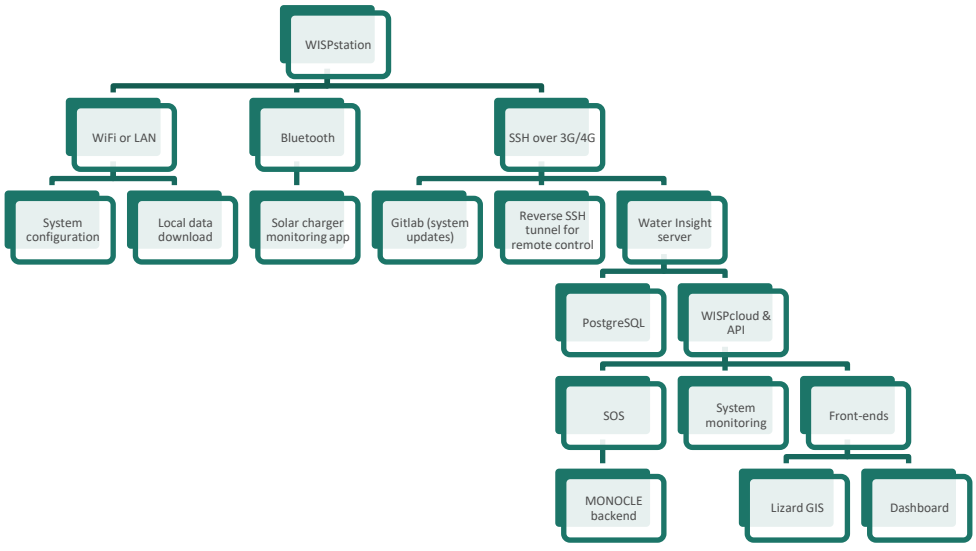
Purpose

The WISPstation/WISP-M provides a high quality measurement of R_{rs} at L2 (and constituting measurements of (2 sets of) L_{up}, L_d and E_d, at L1). The objective of the measurements within the context of the MONOCLE project is to provide reference R_{rs} measurements to compare to other sensors and drone or satellite images.

Connectivity

The WISPstation transmits data automatically to WISPcloud where the data can be retrieved from the **API.** WISPcloud is a PostgreSQL database hosted in Google Cloud. The instrument buffers locally using a SQLite database. A further interface is available to push WISPcloud data to a SOS backend. Log files for monitoring battery status, internal temperature and humidity, power consumption etc. are transmitted separately to monitor the instrument status.

Exposed functionality



There are a number of interfaces to communicate with the WISPstation:

- **Local WiFi or LAN** is used in the lab to configure the instrument and can be used upon request in the field to connect, download data and e.g. change the measurement schedule. Normally this functionality is turned off before installation to save power but

can be turned on again for short periods in 'Maintenance mode' when a reverse tunnel is open.

- **Bluetooth and the VictronConnect-app** can be used on-site to connect to the power manager to see the status of the battery and solar panel and monitor the charging cycle.
- **SSH (over 3g/4g)** is used by WI to collect the data and ancillary information into a PostgreSQL server. Upon turning on a "maintenance mode" flag on the server by Water Insight, the instrument will, after reading the flag in its next cycle, open a reverse SSH tunnel over its LTE modem that will stay open for a limited amount of time to enable remote configuration of e.g. the measurement schedule. During a maintenance mode cycle the instrument can be (if necessary) updated by cloning updates from a private GIT repository. After each measurement cycle the instrument pushes the data over the LTE modem to the WISPcloud database and receives a confirmation. If there is no connectivity, or the transfer is incomplete, the instrument saves the data package in a local SQLite database. In the next cycle a new attempt will be made. If, by the end of the day there are still unsent packages, they are stored permanently on the instrument. Any successfully received data package is deleted from the instrument to save space.
- **WISPcloud and a dedicated API** provide access to calibrated observations. The API manual can be retrieved by pointing a browser to the following URL: https://wispcloud.waterinsight.nl/api/query?SERVICE=Data&VERSION=1.0&REQUEST=GetDocumentation
  using user=demo and passwd=demo as credentials.
  Additionally, instructions how to use the API are available at: https://gitlab.com/waterinsight-public/wispcloud-api-tutorial
- A **dashboard** shows the status of instruments every 15 minutes by querying the API. The dashboard can be ported to clients on request.
- WISPcloud data are further transmitted to the **MONOCLE SOS server** by (in-house) middleware. Functionality has been designed and implemented to set up a secure connection to first register a sensor and secondly to retrieve a dataset from the WISPcloud API, reformat the data into OGC compliant XML format which is subsequently pushed to the SOS server.

Example usage

The WISPstation will be installed at a fixed location or a stable buoy. There will be intensive contact between Water Insight and the user prior to the installation to select the optimal measurement site and to define the installation parameters. The site parameters will be included in the instrument configuration file which resides in WISPcloud and on the instrument. If necessary and feasible Water Insight personnel can help to perform the installation on site. There is a detailed manual describing all the steps of the physical installation process. It is sent together with the instrument and all required tools and parts.

The operator is requested to open the lid covering the connections panel a day before the actual installation and turn the instrument on with the ignition key. From that moment, the instrument is

sending data and log files over an LTE modem to the cloud database (WISPcloud) and the performance of the instrument can be monitored in terms of internal humidity, temperature and power consumption and the (ir)radiance measurements. While the instrument can connect over LAN and WLAN in the laboratory, these connections are normally shut down for field operation because of power consumption considerations. After each completed measurement the WISPstation sends the data to the cloud database and, after receiving confirmation, the data is deleted from the system. When there is no confirmation (or no connection at all) the data is stored on the system until the next cycle, to make a new attempt. Data that were not successfully send during one day are permanently stored on the system. Just before installation, the user removes the caps that cover the (ir)radiance sensors.

Before, during and after the installation the operator can consult the VictronConnect-app to monitor the power flow from the solar panel to the instrument and the battery load status. Thus, it can be checked if the connection to the solar panel is properly done and working.

After the installation, the orientation of the instrument should be accurately measured since the observation azimuths should be precisely known. In principle, the side that carries the radiance sensors should be facing North on the Northern hemisphere. This will ensure that the instrument observes in the NNE and NNW directions and a choice can be made which orientation results in the least sun glint. Any deviation from this optimal orientation will be included in the configuration file. If a user changes the orientation, this will have to be reported to Water Insight. In concert with the user the sampling times and intervals can be set in the instrument's crontab. During the period of operation Water insight can change sampling times, frequency and perform software maintenance by SSH over a secure reverse tunnel. A dashboard at Water Insight allows to monitor the main performance parameters of an operational WISPstation. WISPcloud handles the processing of raw observations (counts) to calibrated remote sensing reflectance and some standard water quality parameters based on published robust water quality algorithms (mostly suitable for case 2 waters). The user can, at all times, connect to the output API and collect reflectance spectra and WQ parameters. Users can also collect the data at L1, constituting of calibrated (ir)radiances per single measurement (normally sets of 10 are measured per channel per cycle). The API can be queried per instrument, area and time window. Login credentials are supplied to each new user.

Support is available through support@waterinsight.nl.

6. System developer guide

This chapter provides additional technical detail on selected system components to help system developers to configure a data environment, or individual components similar to the MONOCLE data ecosystem. This information is provided without guarantee and is not updated to reflect software updates. We advise its use only by experienced developers on non-critical systems.

6.1. SOS server

SOS

Location in system: Backend — PML

Requirements

- Tomcat server installed
  - version 6 or higher (But not version v8.0.8, 7.0.54, or 6.0.41)
  - with Java 8.0 or higher but not 9
- PostgreSQL Server installed
  - version 9 or higher
  - PostGIS 2.0 or higher

Installation

1. **Locate the WAR file**
   - Download the desired SOS release: https://github.com/52North/SOS/releases
   - Unzip the bundle
   - The WAR file we need are found in ~/bin/target
2. **Create the database**
   - Create a new PostgreSQL database using the PostGIS template created during the PostGIS installation.
   - Verify that PostGIS is installed and functioning:
     `SELECT PostGIS_full_version();`
   - Create a user, or allow access to the new database for an existing user
3. **Install the WAR file**
   - Either upload through the tomcat manager: ~/manager/html
     or copy the WAR file to the appropriate webapps directory
4. **Configure your SOS instance**
   - Go to the service page (by default /52n-sos-webapp/)
   - Configure the webapp to point to the PostgreSQL database
   - Change the password for the admin user
   - Restrict Transactional Security to be local IPs only
     (Admin > Setttings > Transactional Security)
   - tick the box at the bottom of the Transaction Security page marked Delete observation or procedure physically
   - Configure any remaining parameters, as per the documentation below

Documentation

- **SOS Standard Documentation**
- **SOS Server Documentation**
- Examples: see `/<install-path/client` in your installation

6.2. SOS Proxy

SOS

Location in system: Backend — PML

Requirements

- Docker server
  - version 19 or higher
- Postgres Database
  - Version 9 or higher

Installation

1. **Download source**
   https://github.com/pml-snee/MONOCLE-SOS-proxy
2. **Build docker**
   `docker build . --tag sos-proxy`
3. **Upload docker to docker server**
4. **Start service**
   `docker run -d -p 80:8080 --name sos-proxy-container sos-proxy`

Configuration

- SOS
  The direct SOS URL should be defined in the sos_service variable in the ~/auth_token_gen_app/app.js file
- Postgres
  Your Postgres details should be added into the file ~/auth_token_gen_app/db/index.js in the Pool object parameters
  Create users in your database with
  `INSERT INTO users (username, password) VALUES ('YOUR_USERNAME', 'YOUR_PASSWORD');`

Integration Examples

Authenticate with the server
http://yourserver:yourport/api/get_token/YOUR_SENSOR_NAME/YOUR_USER_NAME/YOUR_PASSWORD

```
Send your SOS xml through the passthrough endpoint with the token from the
previous step
http://yourserver:yourport/api/sos_proxy/YOUR_TOKEN/xml/submit
```

Documentation

Github
https://github.com/pml-snee/MONOCLE-SOS-proxy/blob/main/README.md

6.3. KdUINO: KdUSTICK and KdUMOD

KdUSTICK

Location in system: Sensor — CSIC

Requirements

Required hardware:

- **LoPy4 MicroPython Dev. Board (Espressif Esp32 Chipset)**
- **Pytrack sensor shield (accelerometer, GPS)**
- **LoRa & Sigfox Antenna**

- **MicroSD card**
- **TCS34725 light sensors**
- **LiPo Battery**

Software environment:

- **Visual Studio Code with PyMakr Extension to upload code**

Installation

Sources:

- **https://git.csic.es/KdUINO/kdustick**

List firmware versions:

- **Pycom Firmware v1.20.2.r1**

Integration Examples

Data analysis code: https://git.csic.es/36579996Z/KdUINO-data-analysis

Documentation

- **https://git.csic.es/KdUINO/kdustick**

KdUMOD

Location in system: Sensor (multiple sensors) CSIC

Requirements

Required hardware:

For each independent light sensor module:

- **Esp8266 Dev. Board (Adafruit) + RTC and SD module**
- **MicroSD card**
- **AS7262 multispectral light sensors**
- **Temperature sensor**

For the surface buoy transmitting data:

- **LoPy4 MicroPython Dev. Board (Espressif Esp32 Chipset)**
- **Pytrack sensor shield (accelerometer, GPS)**
- **LoRa & Sigfox Antenna**
- **MicroSD card**
- **TCS34725 light sensors**
- **LiPo Battery**

Software environment:

- **Visual Studio Code with PyMakr Extension to upload code**

Installation

Sources:

- **https://git.csic.es/KdUINO/kdumod**

Firmware versions:

- **Pycom Firmware v1.20.2.r1**
- **MicroPython 1.13**

Integration Examples

Data analysis code: https://git.csic.es/36579996Z/KdUINO-data-analysis
Documentation
https://git.csic.es/KdUINO/kdumod

6.4. iSPEX

iSPEX/Spectacle backend	
Location in system Middleware/Backend	DDQ

Requirements

Hardware

- Smartphone with optional iSPEX add on.

Software

- Parse server

The parse server can be deployed using docker or stand-alone on a linux system. In the present system we used Ubuntu 11 for development and hosting.

Installation

Backend:
The installation (Parse server instance) javascript file is downloadable from:
https://github.com/monocle-h2020/ispex_snippets/blob/master/spectacle_dev.js

The apps will generate the 'table/document' structure in the MongoDB server (part of the default Parse configuration) if no previous data exist.

Spectacle data collection app for smartphone:
IOS app
Android app

Configuration

The parse server is set up following the manual on their website (parseserver.org). For creating listeners/instances use the provided javascript (see installation above). For starting these services up on system boot we use the pm2 javascript process manager, but this is not a requirement.

For Push Notifications on Android we use firebase cloud messaging (which is required).
https://firebase.google.com/docs/cloud-messaging

For Push Notifications on iOS/Apple we use the parse push notification service, this is configured in the parse spectacle_dev.js startup file.

Integration Examples

The apps are connected via their respective software libraries (Parse_ios and Parse_android).

For example:

```
Parse.setLogLevel(Parse.LOG_LEVEL_DEBUG);

    // initialize parse with strings from strings.xml
```

```
        Parse.initialize(new Parse.Configuration.Builder(this)
                .applicationId(getString(R.string.app_id))
                // if defined
                .clientKey(getString(R.string.client_key))
                .server(getString(R.string.server_url))
                .build()
        );
```

Documentation

The apps are documented in-line (see the MONOCLE GitHub)
https://zenodo.org/record/3967124#.X0-NQi2w1QJ#

7. Exploitation and Dissemination

This guide may be re-used freely for non-commercial purposes to inform future development and usage of sensor to backend and user interfaces, while acknowledging the source document. The MONOCLE data ecosystem will continue to be further developed as testing is ramped up to include multiple live sensors and users.

8. Future activities/recommendations

The information contained herein is expected to become obsolete over time, as new software versions and solutions are made available. For demonstration purposes at the limited scale of the MONOCLE network, the network will not be grown into a fully scaled (e.g. cloud-based) solution. However, if the popularity of the network described here were to increase, this should be considered as a useful follow-on, saving individual sensor developers time in setting up dynamic data flows.

9. References

Heard J, Simis S, Ceccaroni L, & Clymans W. (2018, November). Water Quality Survey of the Multiscale Observation Networks for Optical monitoring of Coastal waters, Lakes and Estuaries (MONOCLE) project. Zenodo. http://doi.org/10.5281/zenodo.1625594

10. Appendix

10.1. SOS example xml templates (for So-Rad)

The following templates provide an example of using the SOS backend for storing radiometric data and data about the sensor platform. This example of a 'complex' data offering can be used to derive templates for other sensors. Note that this template does not cover all metadata requirements listed in this guide and is solely intended to guide developers on the inclusion of multiple data types in a SOS result. The following sections describe, respectively, the template XML documents for the procedures to (1) Insert a new sensor instance into the SOS backend, (2) insert a result template into the SOS backend for that sensor, and (3) to insert a single result.

10.1.1 InsertSensor procedure template

```
<?xml version="1.0" encoding="UTF-8"?>
<swes:InsertSensor
    xmlns:swes="http://www.opengis.net/swes/2.0"
    xmlns:sos="http://www.opengis.net/sos/2.0"
    xmlns:swe="http://www.opengis.net/swe/2.0"
    xmlns:sml="http://www.opengis.net/sensorml/2.0"
    xmlns:gml="http://www.opengis.net/gml/3.2"
    xmlns:xlink="http://www.w3.org/1999/xlink"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:gco="http://www.isotc211.org/2005/gco"
    xmlns:gmd="http://www.isotc211.org/2005/gmd" service="SOS" version="2.0.0" xsi:schemaLocation="http://www.opengis.net/sos/2.0
http://schemas.opengis.net/sos/2.0/sosInsertSensor.xsd  http://www.opengis.net/swes/2.0 http://schemas.opengis.net/swes/2.0/swes.xsd">
    <swes:procedureDescriptionFormat>http://www.opengis.net/sensorml/2.0</swes:procedureDescriptionFormat>
    <swes:procedureDescription>
        <sml:PhysicalSystem gml:id="sensor10">
            <!--Unique identifier -->
            <gml:identifier codeSpace="uniqueID">urn:sos:h2020:monocle:pml:procedure:example:1</gml:identifier>
            <sml:identification>
                <sml:IdentifierList>
                    <sml:identifier>
                        <sml:Term definition="urn:ogc:def:identifier:OGC:1.0:longName">
                            <sml:label>longName</sml:label>
                            <sml:value>So-rad</sml:value>
                        </sml:Term>
                    </sml:identifier>
                    <sml:identifier>
                        <sml:Term definition="urn:ogc:def:identifier:OGC:1.0:shortName">
                            <sml:label>shortName</sml:label>
                            <sml:value>So-rad</sml:value>
                        </sml:Term>
                    </sml:identifier>
                </sml:IdentifierList>
            </sml:identification>
            <sml:capabilities name="offerings">
                <sml:CapabilityList>
                    <!-- Special capabilities used to specify offerings. -->
                    <!-- Parsed and removed during InsertSensor/UpdateSensorDescription, added during DescribeSensor. -->
                    <!-- Offering is generated if not specified. -->
                    <sml:capability name="offeringID">
                        <swe:Text definition="urn:ogc:def:identifier:OGC:offeringID">
                            <swe:label>So-rad Instruments</swe:label>
                            <swe:value>urn:sos:h2020:monocle:pml:so-rad:offering:1</swe:value>
                        </swe:Text>
                    </sml:capability>
                </sml:CapabilityList>
            </sml:capabilities>
```

```
<sml:capabilities name="metadata">
    <sml:CapabilityList>
        <!-- status indicates, whether sensor is insitu (true)
            or remote (false) -->
        <sml:capability name="insitu">
            <swe:Boolean definition="insitu">
                <swe:value>false</swe:value>
            </swe:Boolean>
        </sml:capability>
        <!-- status indicates, whether sensor is mobile (true)
            or fixed/stationary (false) -->
        <sml:capability name="mobile">
            <swe:Boolean definition="mobile">
                <swe:value>true</swe:value>
            </swe:Boolean>
        </sml:capability>
    </sml:CapabilityList>
</sml:capabilities>
<sml:featuresOfInterest>
    <sml:FeatureList definition="http://www.opengis.net/def/featureOfInterest/identifier">
        <swe:label>featuresOfInterest</swe:label>
        <sml:feature xlink:href="urn:sos:h2020:monocle:pml:feature-of-interest:so-rad:example:1"/>
    </sml:FeatureList>
</sml:featuresOfInterest>
<sml:inputs>
    <sml:InputList>
        <sml:input name="test_observable_property_10">
            <sml:ObservableProperty definition="urn:sos:h2020:monocle:pml:observable-property:example:1"/>
        </sml:input>
    </sml:InputList>
</sml:inputs>
<sml:outputs>
    <sml:OutputList>
        <sml:output name="test_observable_property_10_1">
            <swe:Quantity definition="urn:sos:h2020:monocle:pml:observable-property:example:1:1">
                <swe:uom code="NOT_DEFINED"/>
            </swe:Quantity>
        </sml:output>
        <sml:output name="h2020_monocle_pml_so-rad_measurements">
          <swe:DataRecord>
            <swe:field name="h2020_monocle_pml_so-rad_measurements_pc_time">
              <swe:Text definition="urn:sos:h2020:monocle:pml:observable-property:so-rad:measurements:pc_time"></swe:Text>
            </swe:field>
            <swe:field name="h2020_monocle_pml_so-rad_measurements_gps_time">
              <swe:Text definition="urn:sos:h2020:monocle:pml:observable-property:so-rad:measurements:gps_time"></swe:Text>
            </swe:field>
            <swe:field name="h2020_monocle_pml_so-rad_measurements_gps_fix">
              <swe:Quantity definition="urn:sos:h2020:monocle:pml:observable-property:so-rad:measurements:gps_fix">
```

```
    <swe:uom code="h2020_monocle_pml_so-rad_gps-fix-units"/>
  </swe:Quantity>
</swe:field>
<swe:field name="h2020_monocle_pml_so-rad_measurements_gps_speed">
  <swe:Quantity definition="urn:sos:h2020:monocle:pml:observable-property:so-rad:measurements:gps_speed">
    <swe:uom code="h2020_monocle_pml_so-rad_speed-units"/>
  </swe:Quantity>
</swe:field>
<swe:field name="h2020_monocle_pml_so-rad_measurements_platform_bearing">
  <swe:Quantity definition="urn:sos:h2020:monocle:pml:observable-property:so-rad:measurements:platform_bearing">
    <swe:uom code="h2020_monocle_pml_so-rad_platform-bearing-units"/>
  </swe:Quantity>
</swe:field>
<swe:field name="h2020_monocle_pml_so-rad_measurements_sun_azimuth">
  <swe:Quantity definition="urn:sos:h2020:monocle:pml:observable-property:so-rad:measurements:sun_azimuth">
    <swe:uom code="h2020_monocle_pml_so-rad_sun-azimuth-units"/>
  </swe:Quantity>
</swe:field>
<swe:field name="h2020_monocle_pml_so-rad_measurements_sun_elevation">
  <swe:Quantity definition="urn:sos:h2020:monocle:pml:observable-property:so-rad:measurements:sun_elevation">
    <swe:uom code="h2020_monocle_pml_so-rad_sun-elevation-units"/>
  </swe:Quantity>
</swe:field>
 <swe:field name="h2020_monocle_pml_so-rad_measurements_motor_temp">
  <swe:Quantity definition="urn:sos:h2020:monocle:pml:observable-property:so-rad:measurements:motor_temp">
    <swe:uom code="h2020_monocle_pml_so-rad_motor-temp-units"/>
  </swe:Quantity>
</swe:field>
 <swe:field name="h2020_monocle_pml_so-rad_measurements_driver_temp">
   <swe:Quantity definition="urn:sos:h2020:monocle:pml:observable-property:so-rad:measurements:driver_temp">
 <swe:uom code="h2020_monocle_pml_so-rad_driver-temp-units"/>
   </swe:Quantity>
 </swe:field>
 <swe:field name="h2020_monocle_pml_so-rad_measurements_pi_cpu_temp">
  <swe:Quantity definition="urn:sos:h2020:monocle:pml:observable-property:so-rad:measurements:pi-cpu-temp">
    <swe:uom code="h2020_monocle_pml_so-rad_pi-cpu-temp-units"/>
  </swe:Quantity>
</swe:field>
 <swe:field name="h2020_monocle_pml_so-rad_measurements_tilt_avg">
  <swe:Quantity definition="urn:sos:h2020:monocle:pml:observable-property:so-rad:measurements:tilt_avg">
    <swe:uom code="h2020_monocle_pml_so-rad_tilt_avg-units"/>
  </swe:Quantity>
</swe:field>
 <swe:field name="h2020_monocle_pml_so-rad_measurements_tilt_std">
  <swe:Quantity definition="urn:sos:h2020:monocle:pml:observable-property:so-rad:measurements:tilt-std">
    <swe:uom code="h2020_monocle_pml_so-rad_tilt-std"/>
  </swe:Quantity>
</swe:field>
```

```
            <swe:field name="h2020_monocle_pml_so-rad_measurements_bearing_accuracy">
              <swe:Quantity definition="urn:sos:h2020:monocle:pml:observable-property:so-rad:measurements:bearing_accuracy">
                <swe:uom code="h2020_monocle_pml_so-rad_bearing_accuracy"/>
              </swe:Quantity>
            </swe:field>
            <swe:field name="h2020_monocle_pml_so-rad_measurements_sorad_version">
              <swe:Text definition="urn:sos:h2020:monocle:pml:observable-property:so-rad:measurements:sorad-version">
              </swe:Text>
            </swe:field>
            <swe:field name="h2020_monocle_pml_so-rad_measurements_batt_v">
              <swe:Quantity definition="urn:sos:h2020:monocle:pml:observable-property:so-rad:measurements:batt-v">
                <swe:uom code="h2020_monocle_pml_so-rad_batt_v"/>
              </swe:Quantity>
            </swe:field>
            <swe:field name="h2020_monocle_pml_so-rad_measurements_inside_temp">
              <swe:Quantity definition="urn:sos:h2020:monocle:pml:observable-property:so-rad:measurements:inside-temp">
                <swe:uom code="h2020_monocle_pml_so-rad_inside-temp"/>
              </swe:Quantity>
            </swe:field>
            <swe:field name="h2020_monocle_pml_so-rad_measurements_inside_rel_hum">
              <swe:Quantity definition="urn:sos:h2020:monocle:pml:observable-property:so-rad:measurements:inside_rel_hum">
                <swe:uom code="h2020_monocle_pml_so-rad_inside_rel_hum"/>
              </swe:Quantity>
            </swe:field>
            <swe:field name="h2020_monocle_pml_so-rad_measurements_n_rad_obs">
              <swe:Count definition="urn:sos:h2020:monocle:pml:observable-property:so-rad:measurements:n-rad-obs"></swe:Count>
            </swe:field>
            <swe:field name="h2020_monocle_pml_so-rad_measurements_sensor_id">
              <swe:Text definition="urn:sos:h2020:monocle:pml:observable-property:so-rad:measurements:sensor_id"/>
            </swe:field>
            <swe:field name="h2020_monocle_pml_so-rad_measurements_inttime">
              <swe:Count definition="urn:sos:h2020:monocle:pml:observable-property:so-rad:measurements:int-time"></swe:Count>
            </swe:field>
            <swe:field name="h2020_monocle_pml_so-rad_measurements_measurement_hash_1">
              <swe:Text definition="urn:sos:h2020:monocle:pml:observable-property:so-rad:measurements:measurement_hash_1"></swe:Text>
            </swe:field>
            <swe:field name="h2020_monocle_pml_so-rad_measurements_measurement_hash_2">
              <swe:Text definition="urn:sos:h2020:monocle:pml:observable-property:so-rad:measurements:measurement_hash_2"></swe:Text>
            </swe:field>
            <swe:field name="h2020_monocle_pml_so-rad_measurements_measurement_hash_3">
              <swe:Text definition="urn:sos:h2020:monocle:pml:observable-property:so-rad:measurements:measurement_hash_3"></swe:Text>
            </swe:field>
            <swe:field name="test_observable_property_example_boolean">
              <swe:Boolean definition="urn:sos:h2020:monocle:pml:observable-property:example:boolean"></swe:Boolean>
            </swe:field>
          </swe:DataRecord>
        </sml:output>
      </sml:OutputList>
```

```
            </sml:outputs>
            <sml:parameters>
                <sml:ParameterList>
                    <sml:parameter name="settings">
                        <swe:Quantity definition="urn:sos:h2020:monocle:pml:parameter:example:1" updatable="false">
                            <swe:label>Test parmeter</swe:label>
                            <swe:uom code="test"/>
                            <swe:constraint>
                                <swe:AllowedValues>
                                    <swe:interval>0.01 10.0</swe:interval>
                                </swe:AllowedValues>
                            </swe:constraint>
                        </swe:Quantity>
                    </sml:parameter>
                </sml:ParameterList>
            </sml:parameters>
        </sml:PhysicalSystem>
    </swes:procedureDescription>
    <swes:observableProperty>monocle-observable-property-pml-test-10-1</swes:observableProperty>
    <swes:observableProperty>monocle-observable-property-pml-test-10-8</swes:observableProperty>
    <swes:metadata>
        <sos:SosInsertionMetadata>
            <sos:observationType>http://www.opengis.net/def/observationType/OGC-OM/2.0/OM_Measurement</sos:observationType>
            <sos:observationType>http://www.opengis.net/def/observationType/OGC-OM/2.0/OM_ComplexObservation</sos:observationType>
            <sos:observationType>http://inspire.ec.europa.eu/featureconcept/TrajectoryObservation</sos:observationType>
            <!-- multiple values possible -->
            <sos:featureOfInterestType>http://www.opengis.net/def/samplingFeatureType/OGC-OM/2.0/SF_SamplingPoint</sos:featureOfInterestType>
            <sos:featureOfInterestType>http://www.opengis.net/def/samplingFeatureType/OGC-OM/2.0/SF_SamplingCurve</sos:featureOfInterestType>
        </sos:SosInsertionMetadata>
    </swes:metadata>
</swes:InsertSensor>
```

10.1.2 InsertResult procedure template

```
<?xml version="1.0" encoding="UTF-8"?>
<sos:InsertResultTemplate
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:swes="http://www.opengis.net/swes/2.0"
    xmlns:sos="http://www.opengis.net/sos/2.0"
    xmlns:swe="http://www.opengis.net/swe/2.0"
    xmlns:sml="http://www.opengis.net/sensorML/1.0.1"
    xmlns:gml="http://www.opengis.net/gml/3.2"
    xmlns:xlink="http://www.w3.org/1999/xlink"
    xmlns:om="http://www.opengis.net/om/2.0"
    xmlns:sams="http://www.opengis.net/samplingSpatial/2.0"
    xmlns:sf="http://www.opengis.net/sampling/2.0"
```

```
    xmlns:xs="http://www.w3.org/2001/XMLSchema" service="SOS" version="2.0.0" xsi:schemaLocation="http://www.opengis.net/sos/2.0
http://schemas.opengis.net/sos/2.0/sosInsertResultTemplate.xsd http://www.opengis.net/om/2.0 http://schemas.opengis.net/om/2.0/observation.xsd
http://www.opengis.net/samplingSpatial/2.0 http://schemas.opengis.net/samplingSpatial/2.0/spatialSamplingFeature.xsd">
    <sos:proposedTemplate>
            <sos:ResultTemplate>
            <swes:identifier>urn:sos:h2020:monocle:pml:so-rad:template:1</swes:identifier>
            <sos:offering>urn:sos:h2020:monocle:pml:so-rad:offering:1</sos:offering>
            <sos:observationTemplate>
                <om:OM_Observation gml:id="sensor2obsTemplate">
                    <om:type xlink:href="http://www.opengis.net/def/observationType/OGC-OM/2.0/OM_ComplexObservation"/>
                    <om:phenomenonTime nilReason="template"/>
                    <om:resultTime nilReason="template"/>
                    <om:procedure xlink:href="urn:sos:h2020:monocle:pml:procedure:example:1"/>
                    <om:observedProperty xlink:href="monocle-observable-property-pml-test-10-8"/>
                    <om:featureOfInterest xlink:href="urn:sos:h2020:monocle:pml:feature-of-interest:so-rad:example:1" xlink:title="So_rad_feature"/>
                    <om:result/>
                </om:OM_Observation>
            </sos:observationTemplate>
            <sos:resultStructure>
                <swe:DataRecord>
                    <swe:field name="phenomenonTime">
                        <swe:Time definition="http://www.opengis.net/def/property/OGC/0/PhenomenonTime">
                            <swe:uom xlink:href="http://www.opengis.net/def/uom/ISO-8601/0/Gregorian"/>
                        </swe:Time>
                    </swe:field>
                    <swe:field name="h2020_monocle_pml_so-rad_measurements">
                      <swe:DataRecord definition="monocle-observable-property-pml-test-10-8">
                        <swe:field name="h2020_monocle_pml_so-rad_measurements_pc_time">
                          <swe:Text definition="urn:sos:h2020:monocle:pml:observable-property:so-rad:measurements:pc_time"></swe:Text>
                        </swe:field>
                        <swe:field name="h2020_monocle_pml_so-rad_measurements_gps_time">
                          <swe:Text definition="urn:sos:h2020:monocle:pml:observable-property:so-rad:measurements:gps_time"></swe:Text>
                        </swe:field>
                        <swe:field name="h2020_monocle_pml_so-rad_measurements_gps_fix">
                          <swe:Quantity definition="urn:sos:h2020:monocle:pml:observable-property:so-rad:measurements:gps_fix">
                            <swe:uom code="h2020_monocle_pml_so-rad_gps-fix-units"/>
                          </swe:Quantity>
                        </swe:field>
                        <swe:field name="h2020_monocle_pml_so-rad_measurements_gps_speed">
                          <swe:Quantity definition="urn:sos:h2020:monocle:pml:observable-property:so-rad:measurements:gps_speed">
                            <swe:uom code="h2020_monocle_pml_so-rad_speed-units"/>
                          </swe:Quantity>
                        </swe:field>
                        <swe:field name="h2020_monocle_pml_so-rad_measurements_platform_bearing">
                          <swe:Quantity definition="urn:sos:h2020:monocle:pml:observable-property:so-rad:measurements:platform_bearing">
                            <swe:uom code="h2020_monocle_pml_so-rad_platform-bearing-units"/>
                          </swe:Quantity>
                        </swe:field>
```

```
 <swe:field name="h2020_monocle_pml_so-rad_measurements_sun_azimuth">
   <swe:Quantity definition="urn:sos:h2020:monocle:pml:observable-property:so-rad:measurements:sun_azimuth">
     <swe:uom code="h2020_monocle_pml_so-rad_sun-azimuth-units"/>
   </swe:Quantity>
 </swe:field>
 <swe:field name="h2020_monocle_pml_so-rad_measurements_sun_elevation">
   <swe:Quantity definition="urn:sos:h2020:monocle:pml:observable-property:so-rad:measurements:sun_elevation">
     <swe:uom code="h2020_monocle_pml_so-rad_sun-elevation-units"/>
   </swe:Quantity>
 </swe:field>
 <swe:field name="h2020_monocle_pml_so-rad_measurements_motor_temp">
   <swe:Quantity definition="urn:sos:h2020:monocle:pml:observable-property:so-rad:measurements:motor_temp">
     <swe:uom code="h2020_monocle_pml_so-rad_motor-temp-units"/>
   </swe:Quantity>
 </swe:field>
 <swe:field name="h2020_monocle_pml_so-rad_measurements_driver_temp">
    <swe:Quantity definition="urn:sos:h2020:monocle:pml:observable-property:so-rad:measurements:driver_temp">
       <swe:uom code="h2020_monocle_pml_so-rad_driver-temp-units"/>
    </swe:Quantity>
 </swe:field>
 <swe:field name="h2020_monocle_pml_so-rad_measurements_pi_cpu_temp">
   <swe:Quantity definition="urn:sos:h2020:monocle:pml:observable-property:so-rad:measurements:pi-cpu-temp">
     <swe:uom code="h2020_monocle_pml_so-rad_pi-cpu-temp-units"/>
   </swe:Quantity>
 </swe:field>
 <swe:field name="h2020_monocle_pml_so-rad_measurements_tilt_avg">
   <swe:Quantity definition="urn:sos:h2020:monocle:pml:observable-property:so-rad:measurements:tilt_avg">
     <swe:uom code="h2020_monocle_pml_so-rad_tilt_avg-units"/>
   </swe:Quantity>
 </swe:field>
 <swe:field name="h2020_monocle_pml_so-rad_measurements_tilt_std">
   <swe:Quantity definition="urn:sos:h2020:monocle:pml:observable-property:so-rad:measurements:tilt-std">
     <swe:uom code="h2020_monocle_pml_so-rad_tilt-std"/>
   </swe:Quantity>
 </swe:field>
 <swe:field name="h2020_monocle_pml_so-rad_measurements_bearing_accuracy">
   <swe:Quantity definition="urn:sos:h2020:monocle:pml:observable-property:so-rad:measurements:bearing_accuracy">
     <swe:uom code="h2020_monocle_pml_so-rad_bearing_accuracy"/>
   </swe:Quantity>
 </swe:field>
 <swe:field name="h2020_monocle_pml_so-rad_measurements_sorad_version">
   <swe:Text definition="urn:sos:h2020:monocle:pml:observable-property:so-rad:measurements:sorad-version">
   </swe:Text>
 </swe:field>
 <swe:field name="h2020_monocle_pml_so-rad_measurements_batt_v">
   <swe:Quantity definition="urn:sos:h2020:monocle:pml:observable-property:so-rad:measurements:batt-v">
     <swe:uom code="h2020_monocle_pml_so-rad_batt_v"/>
   </swe:Quantity>
```

```
                        </swe:field>
                        <swe:field name="h2020_monocle_pml_so-rad_measurements_inside_temp">
                          <swe:Quantity definition="urn:sos:h2020:monocle:pml:observable-property:so-rad:measurements:inside-temp">
                            <swe:uom code="h2020_monocle_pml_so-rad_inside-temp"/>
                          </swe:Quantity>
                        </swe:field>
                        <swe:field name="h2020_monocle_pml_so-rad_measurements_inside_rel_hum">
                          <swe:Quantity definition="urn:sos:h2020:monocle:pml:observable-property:so-rad:measurements:inside_rel_hum">
                            <swe:uom code="h2020_monocle_pml_so-rad_inside_rel_hum"/>
                          </swe:Quantity>
                        </swe:field>
                        <swe:field name="h2020_monocle_pml_so-rad_measurements_n_rad_obs">
                          <swe:Count definition="urn:sos:h2020:monocle:pml:observable-property:so-rad:measurements:n-rad-obs"></swe:Count>
                        </swe:field>
                        <swe:field name="h2020_monocle_pml_so-rad_measurements_sensor_id">
                          <swe:Text definition="urn:sos:h2020:monocle:pml:observable-property:so-rad:measurements:sensor_id"/>
                        </swe:field>
                        <swe:field name="h2020_monocle_pml_so-rad_measurements_inttime">
                          <swe:Count definition="urn:sos:h2020:monocle:pml:observable-property:so-rad:measurements:int-time"></swe:Count>
                        </swe:field>
                        <swe:field name="h2020_monocle_pml_so-rad_measurements_measurement_hash_1">
                          <swe:Text definition="urn:sos:h2020:monocle:pml:observable-property:so-rad:measurements:measurement_hash_1"></swe:Text>
                        </swe:field>
                        <swe:field name="h2020_monocle_pml_so-rad_measurements_measurement_hash_2">
                          <swe:Text definition="urn:sos:h2020:monocle:pml:observable-property:so-rad:measurements:measurement_hash_2"></swe:Text>
                        </swe:field>
                        <swe:field name="h2020_monocle_pml_so-rad_measurements_measurement_hash_3">
                          <swe:Text definition="urn:sos:h2020:monocle:pml:observable-property:so-rad:measurements:measurement_hash_3"></swe:Text>
                        </swe:field>
                        <swe:field name="test_observable_property_example_boolean">
                          <swe:Boolean definition="urn:sos:h2020:monocle:pml:observable-property:example:boolean"/>
                        </swe:field>
                      </swe:DataRecord>
                    </swe:field>
                    <swe:field name="samplingGeometry">
                        <swe:Vector definition=http://www.opengis.net/def/param-name/OGC-OM/2.0/samplingGeometry
referenceFrame="http://www.opengis.net/def/crs/EPSG/0/4326">
                            <swe:coordinate name="latitude">
                                <swe:Quantity definition="latitude" axisID="lat">
                                    <swe:uom code="deg"/>
                                </swe:Quantity>
                            </swe:coordinate>
                            <swe:coordinate name="longitude">
                                <swe:Quantity definition="longitude" axisID="lon">
                                    <swe:uom code="deg"/>
                                </swe:Quantity>
                            </swe:coordinate>
                        </swe:Vector>
```

```
                    </swe:field>
                </swe:DataRecord>
            </sos:resultStructure>
            <sos:resultEncoding>
                <swe:TextEncoding tokenSeparator="#" blockSeparator="@"/>
            </sos:resultEncoding>
        </sos:ResultTemplate>
    </sos:proposedTemplate>
</sos:InsertResultTemplate>
```

10.1.3 InsertResult procedure template

```
<?xml version="1.0" encoding="UTF-8"?>
<sos:InsertResult
    xmlns:sos="http://www.opengis.net/sos/2.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" service="SOS" version="2.0.0" xsi:schemaLocation="http://www.opengis.net/sos/2.0
http://schemas.opengis.net/sos/2.0/sos.xsd">
    <sos:template>urn:sos:h2020:monocle:pml:so-rad:template:1</sos:template>
    <sos:resultValues>2@2019-12-19T13:20:30+02:00#datetime1#datetime2#12.4#159.15#45.5#10#1.1#2.1#3.1#4.1#34#3#78#0.6#12#11.4#43#42#sensor id we should not
see#1576754430#put your#measurement values#here#true#27.992421#-15.362673@2019-12-
19T13:20:33+02:00#datetime3#datetime4#12.5#159.15#45.6#20#1.1#2.1#3.1#4.1#34.1#3.1#78#0.6#12#11.4#43#42#sensor id we should see#1576754430#put your#measurement
values#here#true#27.992421#-15.362673@</sos:resultValues>
</sos:InsertResult>
```