GAIA-CLIM deliverable D5.4

Gap Analysis for Integrated Atmospheric ECV CLImate Monitoring

WP5: Creation of a "virtual observatory" visualization and data access facility

D5.4: "Graphical User Interface".

A Horizon 2020 project;

Grant agreement: 640276

Date: 11 September 2017

Lead Beneficiary: TUT

Nature: Other

Dissemination level: PU

GAIA-CLIM deliverable D5.4

Work-package	**WP5 (Creation of a "virtual observatory" visualization and data access facility)**
Deliverable	D5.4
Nature	Other
Dissemination	PU
Lead Beneficiary	TUT
Date	11/09/2017
Status	Final
Authors	Kalev Rannat (TUT), Arndt Meier (EUMETSAT), Hannes Keernik (TUT), Merik Meriste (TUT), Neeme Loorits (TUT), Tonis Kelder (TUT), Jüri Helekivi (TUT)
Editors	
Reviewers	Peter Thorne (NUIM), Anna Mikalsen (NERSC)
Contacts	kalev.rannat@gmail.com , Arndt.Meier@eumetsat.int
URL	http://www.gaia-clim.eu

This document has been produced in the context of the GAIA-CLIM project. The research leading to these results has received funding from the European Union's Horizon 2020 Programme under grant agreement n° 640276.

Version	Author(s) /Reviewers	Date	Changes
0.1	Kalev Rannat, Arndt Meier	2017-09-04	1st draft
0.2	Kalev Rannat, Arndt Meier, Hannes Keernik, Merik Meriste, Neeme Loorits, Tonis Kelder, Jüri Helekivi	2017-09-08	2nd draft
0.3	Kalev Rannat, Arndt Meier, Hannes Keernik, Merik Meriste, Neeme Loorits, Tonis Kelder, Jüri Helekivi	2017-09-10	3rd draft. Slight restructuring and updates in sections 3.1 and 3.2.
1.0	Peter Thorne, Anna Christina Mikalsen	2017-09-12	Slight restructuring and editorial corrections

Table of Contents

[1] Introduction 5

[2] Outline of this deliverable 6

[3] Virtual Observatory (VO) and Graphical User Interface (GUI) – general aspects 6

[3.1] Architecture 7

[3.2] Data and data flow 10

[3.2.1] Correction of drift in radiosonde data 13

[3.2.2] Collocation implementation by lookup tables method 14

[3.3] Data, metadata and Data Ingestion Script (DIS) 15

[3.4] GUI (functional overview) 16

[4] Summary and recommendations 24

Glossary 25

APPENDIX A: The Virtual Observatory and Tutorials on the internet 26

APPENDIX B: Data Ingestion Scripts 27

[1] Introduction

The objectives of this work package are:

- Creation of a collocation database for satellite and reference measurements including NWP model-based (re)analyses.
- Creation of data interrogation and visualization tools building upon existing European and global infrastructure capabilities.
- Evaluation of the resulting Virtual Observatory to demonstrate its utility for scientific/statistical analysis of respective observations, performance characteristics and the monitoring of instrument and product behaviour over time.
- Provision of a transition roadmap for the Virtual Observatory (including outputs from WP1 through WP4) from research to operational status enabling operational use in Copernicus services.

Hence, the main deliverable of WP5 is the Virtual Observatory (VO) facility which will enable users to carry out comparisons of satellite data products to non-satellite reference-quality data. A range of visualisation and analysis tools have been developed to enable users to explore, analyse, and interact with the data provided within the VO and these are described herein.

Substantial contributions to the VO development have been received from many GAIA-CLIM partners in the form of discussion and support, software, and actual GAIA-CLIM deliverables. Noteworthy contributions from underlying work packages are:

- The metadata database and the CESIUM visualisation tool (WP1 lead by CNR),
- Collocation engine (EUMETSAT),
- Error traceability diagrams, measurement system questionnaires, and reference product readiness (WP2 lead by BKS),
- Look-up tables for collocation mismatch and smoothing errors derived with OSSSMOSE (WP3 lead BIRA-IASB), and
- The GRUAN processor (WP4 lead by UK MetOffice).

The Graphical User Interface (GUI) for the Virtual Observatory, as a main output of WP5 Task 5.2 is built to showcase potential methods and scientific advances from all underlying work packages (WPs). The implementation supported within the GAIA-CLIM project remit shall only serve as a proof-of-concept for an operational VO facility.

[2] Outline of this deliverable

The Virtual Observatory with its GUI and data access facility can be considered as a functional piece of software with an underlying database accessible through the internet.

Some aspects of the VO are already described in earlier deliverables of the project. Particularly, the technological platform for the collocation database and the basic architecture in D5.3 and data ingestion to the database and the tools/methods for the quantification of collocation mismatch and smoothing uncertainties in D3.5.

The present document is a short overview on the VO and its GUI focussing on the technical implementation. Deliverable 5.5 will provide a more detailed description of the tools and software provided by the VO from a user perspective (user guide).

Video tutorials (delivered in D5.5) have been created to illustrate in detail the present functionality of the VO and should help to reproduce the examples shown.

The GUI layout and its functionalities have been advanced compared to the first prototype version presented at the GAIA-CLIM 2nd user workshop in Nov 2016 and at the GAIA-CLIM General Assembly in Feb 2017. Improvements have been made in accordance with feedback from both pilot users and inhouse testing, as well as for software-technical reasons. For the end-user one of the most noticeable changes is the 3D-tool delivered by CNR under WP1 that visualises the metadata. It is highly interactive and comes with an appealing modern graphics engine. The other notable change is the new plotting engine for the collocation data. The decision has been made not to deliver satellite imagery – accordingly, this feature has been removed from the VO.

The new development framework and the Plotly graphical data plotting engine give this new VO GUI a much more professional outlook and follow clearly defined software standards. As a result, the GUI is not only more robust but also a lot sleeker and easier to maintain.

The URLs of a live demo of the VO and the video tutorials are listed in Appendix A.

[3] Virtual Observatory (VO) and Graphical User Interface (GUI) – general aspects

The design and implementation of the VO and its GUI is based on the rationale and the scope of usage of the VO.

Virtual Observatory Rationale:

- Enhance exploitation and visualisation of ground-based reference data for satellite product validation;

- Integrate ground-based reference data with existing satellite-satellite comparisons and observation feedback from NWP models and reanalysis;
- Allow users to interrogate multiple data sources in a seamless way, and permit remote data analysis;
- Increase awareness among users of satellite and non-satellite data, and on the concept of traceable uncertainty estimates;
- Provide a facility to support Copernicus Services to analyse product quality in a sustainable routine mode.

Envisioned Usage of the VO:

- Quality assessment of satellite
    - raw data (Level 1 data), and
    - retrieved products (Level 2 data),
    - both global and regional;
- Quality assessment of NWP model-based forecast/analysis and reanalysis data;
- Assessment of long-term stability of satellite data records;
- Quality assessment of other non-satellite measurements;
- Model evaluation (all aspects including processes);
- Climate analysis (variability, trend detection, etc.).

A particular challenge has been that the tools and approaches to be integrated have evolved gradually in accordance with the progress made in other WPs with development and implementation occurring almost in parallel. Further development shall occur over the coming months and be informed by feedback from the dedicated outreach activities that now replace the third user workshop.

Though the current implemantation of the VO and GUI serves as a proof-of-concept demonstrator, it is designed to be easily expandable by adding new features and data. The GUI follows principles of responsive design and is implemented in the progressive JavaScript framework VUE (https://vuejs.org/). Data visualization is based on Open Source JavaScript graphics library *plotly.js*. (https://plot.ly/javascript/).

[3.1] Architecture

Based on the rationale and possible usage of the VO, the application/implementation needs to handle diverse and massive input data originating from three main sources of data:

- ground-based observational networks (in-situ or remotely sensed + uncertainties),
- satellite-borne instruments (satellite L1 radiances, L2 retrievals + uncertainties),
- NWP simulations and reanalysis with uncertainties.

The architecture of the VO is chosen as characteristic for a client-server application. It consists of two main parts – the client and the server with a database (Figure 3.1).

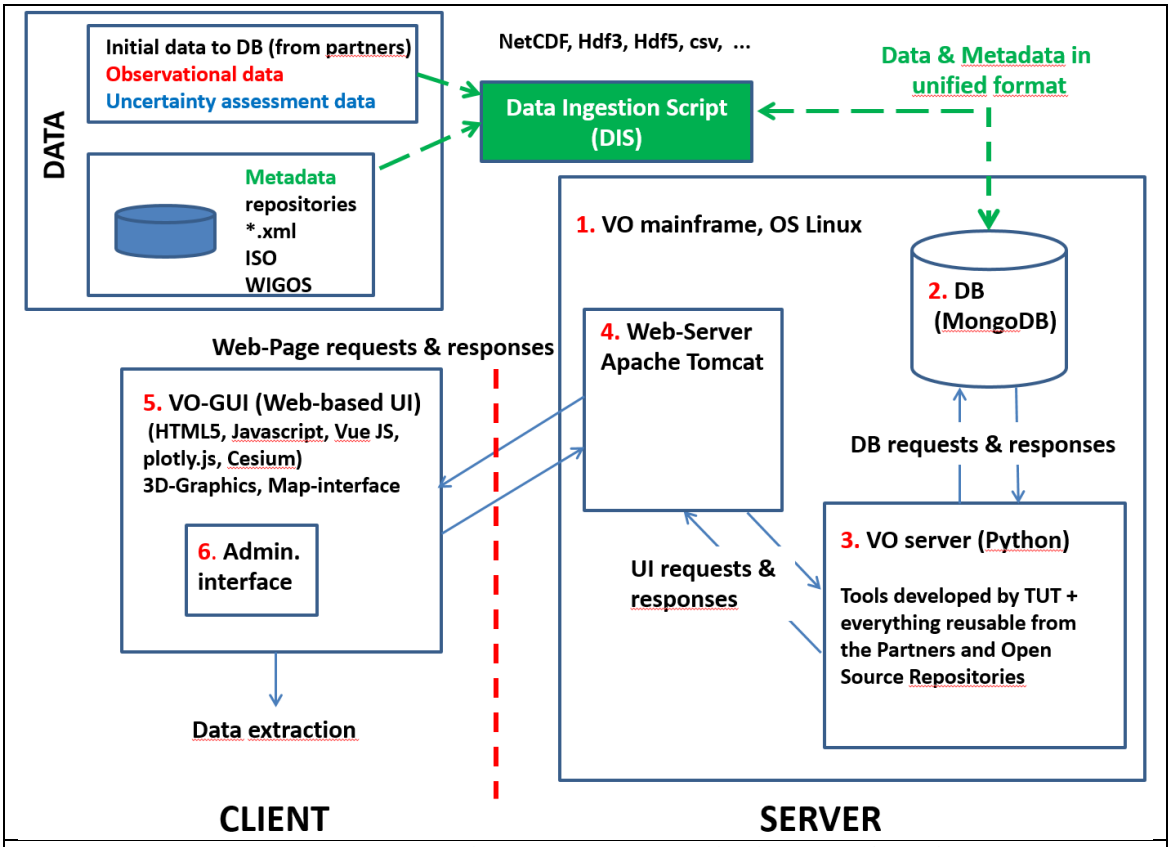

Figure 3.1 represents the architecture of the VO, consisting of a front end (client) and the back end (server).

The main data-handling paradigm in the VO is "searching data by its metadata". With metadata we mean data about what type of observations have been made at what location by what kind of instrument and network over what periods of time. For this purpose, the VO offers an integrated metadata search- and visualization tool developed by CNR. The user can easily search through the sites and networks offering data for several ECVs of interest. Additionally, the user can find information about data availability for user-defined times of interest, the location of the measurements, data and network quality (in form of the network maturity matrix described in D1.3 "Report on system of systems adopted and rationale"), etc. These features are supported by a metadata visualization tool (with its description given in D5.5). This can be helpful in collecting first-step information for further data mining or for sending a specific data request ("Data selection" in the VO main menu).

The metadata is collected and stored by CNR under WP1. The VO shares and uses this metadata as it has been ingested into the same MongoDB. However, the original metadata is kept on CNR premises in a supplementary database in case original data records need to be restored. Due to different data formats originating from different data providers, the VO uses tools for data harmonization. The necessary data harmonisation is taken care of by the data Ingestion Script(s) developed at TUT. As a result, the VO stores data internally in a unified data and metadata format.

The total data volume is expected to grow rapidly, because thus far we have been focusing on ingesting reasonably large samples of different types of data and to get the data handling

routines into place and tested. Processing and ingesting data of the same type but from different locations or years is mostly trivial. Consequently, it is expected that substantial additional data volumes shall be ingested over the next 6 months. Further, even more observations will be added as progressively more satellites commence their observation programmes (for example enhancing the Sentinel program) and adding data for different ECVs were the VO to become operational. Potentially, one could consider these operations with observational data collections as Big Data, and one clearly needs to continuously look after a growing database. The expectation of progressively adding more and different kinds of data with specifications that are not fully known in advance have been the main consideration in the decision on a modern object-oriented non-relational open source database -- MongoDB (mongodb.org) over traditional, relational databases.

Unlike for MySQL, PostgreSQL or OracleDB, the object oriented approach does not require defining the layout of the database completely in advance. The layout and data structure can be modified as needed in an extendable and seamless manner without the need to convert existing records to the new format each time. We anticipate that different future observational data will have slightly different needs, and hence these anticipated changes in data structures will not cause problems. In addition, from the application development point of view, MongoDB is a mature product with the biggest user base (68 million) amongst the non-SQL type databases; it does not entail any licensing costs, is scalable, and works seamlessly across all major operating systems.

The VO client (Figure 3.1) provides access to the GUI (running on a dedicated server) for direct interaction with the user. The GUI offers tools for data selection, data mining, processing, visualization, basic statistical analysis and export of the data (either original observational data or graphical images). The GUI offers users also the means to communicate with the database and with the server via HTTP-requests.

The server (VO-server) is a body of software for receiving and interpreting the HTTP-requests sent by the client and sending HTTP-responses that match the requests. For example, the response may contain certain observational data from the database as requested by the user, sent in JavaScript Object Notation (JSON) format. The client and the server “talk JSON”. To illustrate this in an example: To search for the available products from a GRUAN radiosonde that has been processed with the radiative transfer model RTTOV (“GRUAN-processor”/WP4) producing brightness temperature values that match the spectral characteristics of a selected satellite instrument, the request would be:

http://metadata_cci_cf/distinct/measurand_variable_list?programme_network_affiliation_name=GRUAN&measurement_observing_method=processor

and the server responds with:

```
[
 "Additional quality control information",
 "Air pressure",
 "Air temperature",
```

```
  "Air temperature at 2m",
  "Brightness temperature",
  "Correlated uncertainty of air temperature",
  "Correlated uncertainty of relative humidity",
  "Land-sea mask",
   ....
]
```

The user can also send more complicated requests, for example, requesting the calculation of some statistics over the chosen datasets. All intensive computations are performed at the server side by using either standard math libraries or other dedicated tools installed on the VO server.

Other examples of server-side software are the collocation software STAMP (Space Time Angle Matchup Procedure, developed at EUMETSAT) and the radiative-transfer model for radiosonde soundings, the GRUAN Processor (UK MetOffice). However, these latter tools can also be run on Linux workstations at original data centres prior to data ingestion, which is almost always preferable because of the huge data amounts given by satellite observation products (needed by STAMP) or NWP simulation data (needed by the GRUAN processor). The amount of data produced from successful collocations is tiny by comparison and easily transferred to the VO database server where the actual ingestion is carried out.

The main task for the server software (VO-server) is to respond to the client's HTTP-requests and to perform all and often heavy operations with the database.

The interaction of the user with the GUI drives the VO data flow (including filtering and data processing) and visualises (and/or exports) the results to the user's computing environment. The GUI is designed to be platform-independent and should work with all common web-browsers. Mozilla and Chrome have already been successfully tested. Other browsers, including Internet Explorer and Safari, will follow.

[3.2] Data and data flow

Data for the VO comes from numerous observational networks, particularly from reference networks like GRUAN or NDACC. Metadata is collected by CNR. The three most widely used data formats amongst network operators are ISO-19115-3, CCI-CF and WIGOS. These standards distinguish themselves through using completely different nomenclatures and internal structures, which makes them inherently difficult to harmonise. The original metadata is visualized by CNR's 3D tool (VO metadata view). Data formats must be harmonized before they can be used in the VO. Though the Data Ingestion Script (DIS) has a capability to handle all these metadata formats, the metedata format used by the VO internally is compiled from the observational data where observational data has been ingested. This avoids any potential duplicates. How this is done in detail with the help of the DIS is described in Appendix B. After

harmonization, the records of data and metadata in unified format are ingested into the MongoDB.

The database is built so that it is possible to extract metadata and also observational data (either collocated or non-collocated) in its original netCDF format. In the case of collocated data, the STAMP tool takes care of reading all kinds of different input data formats such as hdf4, hdf5, netCDF, IASI native, BUFR, etc. and writes its output in a standardised netCDF data format. The metadata the user can investigate and look at with the Metadata Visualization Tool is stored on CNR premises and cannot be extracted from there at this stage. A potential data export feature is in the hands of WP1 to decide and implement. However, if the user wanted to export metadata, the same data could also be reached via the VO MongoDB (a decision whether to implement this feature has not been implemented yet). There is no direct data flow between the VO and the Metadata DB at CNR. As far as the ingestion via the DIS is concerned, the Metadata DB works off-line.

The DIS can be used for single data sets and also for collocated datasets. When talking about collocated data in the VO, we are referring to pre-collocated data with a maximum time difference of 6 hours and a maximum distance of 500 km. Collocations for the VO are created with the collocation engine STAMP (EUMETSAT). Collocation mismatch and smoothing errors are calculated using Lookup tables (LUTs) (see detailed description in D3.5). As illustrated in Figure 3.2, the DIS prepares both types of datasets A and B with metadata C and then ingests them into the MongoDB.

The VO GUI gives users the possibility to refine the collocation criteria during the data selection and filtering process by specifying smaller values for max. distance and max. time difference. The result will be a subset of the total collocations (temporary user-collocated data in Figure 3.2). On the GUI the user should notice that the number of datasets is getting smaller. After the new (user-specific) collocation criteria are given, it is possible to continue working with this user-collocated dataset (plotting, statistics, data export, etc.).

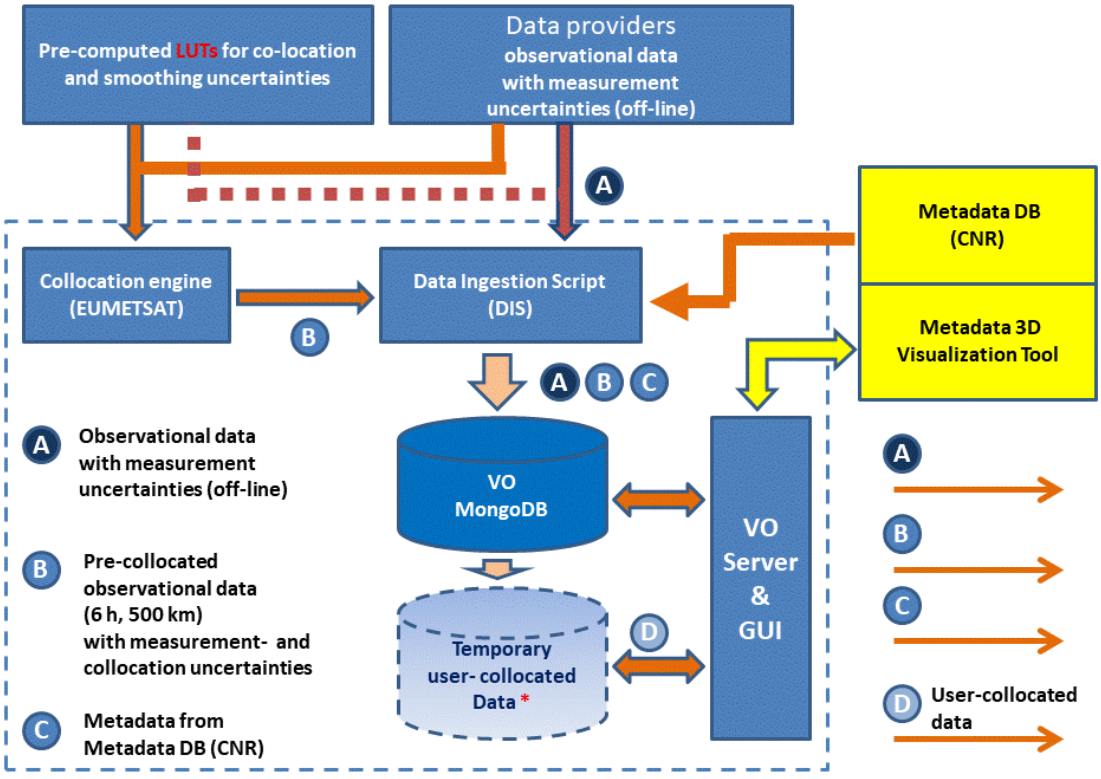

Figure 3.2. VO GUI Data Flow diagram

Collocations are performed with the collocation engine (Figure 3.2), searching over all datasets available for a certain ECV (for example, brightness temperature from processed GRUAN radiosondes are matched with the HIRS-4 instruments from 3 different satellites, namely Metop-A, Metop-B and NOAA-19). One radiosonde sounding results in one file per satellite resulting for this example in a maximum of 3 files, one each for Metop-A, Metop-B and NOAA-19, if those satellites happen to pass by near enough in time and space. Collocated data for a certain sounding (and satellite) are written into one file, including its metadata, global attributes, and observational data with uncertainties from all datasets matching the collocation criteria.

During a satellite overpass for a specific sounding more than one pixel typically fullfills the collocation criteria. In this case, up to 48 such pixels are kept, ensuring that the closest in time and space is amongst them. For the vast majority of collocations, this procedure preserves all matching pixels. The observational data of these pixels are ordered by (effective) distance and then written into the collocation file. In the database, one collocation file forms one collocation even if it contains multiple pixels. The GUI presently displays the values corresponding to the closest cloudfree pixel, but future versions of the GUI may allow a user to choose specific pixels or pool them in certain ways. For a validation scientist, this additional dimension of collocation distance may be of interest to derive correlation lengths for a variable

under investigation such as water vapour or ozone. The data export feature preserves this information; allowing a user to analyse the data offline with more specialised tools.

For the GRUAN radiosonde data its augmentation with NWP simulations introduces additional complexities. Two such NWP models are presently in use by the VO: ECMWF and UK MetOffice. The NWP data is needed to fill in reasonable values above the ceiling altitude of the radiosonde that lies typically between 25 and 33km altitude. This is done with either model before ingestion, thus duplicating the GRUAN data set with only very minor yet not negligible differences. Additionally, there are simulations available from either of the NWP models interpolated to the actual flight path of the radiosonde using the one timestep prognostic NWP data (typ. 6 hours since the last ingestion of observational data into the NWP). This results in two more data sets per original sounding resulting in a total of 4 datasets a user may choose from. The NWP simulations are of high interest to the NWP community (and WP4) to validate the NWP and achieve reference quality.

In addition to the original metadata from either observation, the collocation file also stores the version of the software used to produce this collocation. This makes also the collocation step traceable and searchable for selective re-processing in the future.

[3.2.1] Correction of drift in radiosonde data

The collocation tool STAMP uses a pragmatic approach to correct for some of the systematic sampling mismatch errors caused by the drift and very long ascent times (approx.. 2 hours) in radiosonde soundings. This is a simple and quite certainly imperfect approach that nonetheless improves the quality of resulting collocations significantly.

- An analysis of existing Level 1 radiosonde data that used the launch site co-ordinates showed the best agreement with satellite data taken approx. 30 min after the radiosonde launch, which corresponded to a pressure level (altitude) of the sonde of 300 hPa. This is to be expected since a radiosonde takes typically between 90 and 180 min to reach ceiling altitude. Instead of site location we use the location of the sonde when it crosses the 300 hPa layer to find the nearest collocation of the satellite observation.

- We use the resulting differences in latitude/longitude from the sonde drift to derive an airmass motion vector and estimate the movement of the air mass sampled by the satellite to account for the time difference between the 2 observations of sonde and satellite. This provides an “effective distance”.

  Sonde Drift: $\Delta T = T_1$ (300hPa) - T_0(1000hPa) (approx. 30min). For this time difference we extract the respective difference in latitude and longitude of the sonde ΔLat, ΔLon. Hence, the best match for HIRS observations can be expected for (T_1, Lat_1, Lon_1).

- In a first order approximation we assume that air masses near the sonde are moving with a speed vector of $\Delta Lat/\Delta T$ and $\Delta Long/\Delta T$.

- Hence, with $T_{diff}=T_1-T_S$ an air mass sampled by satellite at (T_S,Lat_S,Lon_S) is approx. found at (T_1, Lat_{Seff} =Lat_S*T_{diff}* ΔLat/ΔT, Lon_{Seff} =Lon_S*T_{diff} * ΔLon/ΔT) from which we calculate the effective distance D_{eff}= (Lat_1- Lat_{Seff}, Lon_1- Lon_{Seff}) [km] (Fig. 3.2.1).

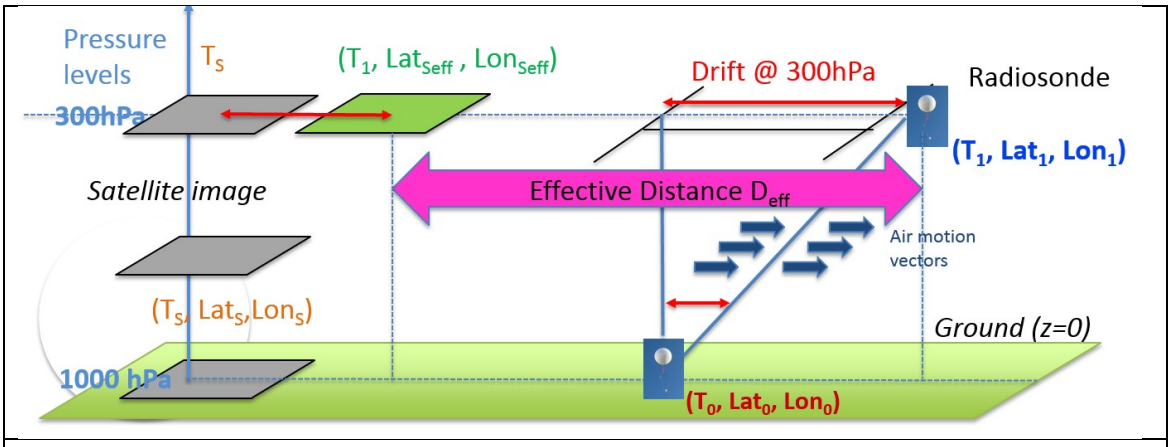

Figure 3.2.1 Correction of radiosonde drift

Example of a metadata excerpt from a processed GRUAN radiosonde collocation file can be found from Appendix B, Figure B3.

In fact this example in Appendix B is a metadata file (collocation file) generated by DIS that the VO data filter uses for quick search of collocations for user requests. Initially, the collocated data itself comes all in one netCDF file as a product of the collocation engine. Use of this collocation-file for getting the real measured values is illustrated in Figures B2, B3 and B4 in Appendix B.

[3.2.2] Collocation implementation by lookup tables method

The VO offers in addition to the measurement uncertainties also uncertainties originating from collocation mismatch and smoothing errors. This complicated task is simplified by using Look-up Tables (LUT). LUTs are the most cost-effective implementation of the complex developments and CPU-intensive computing. For example, integration of the Observing System of Systems Simulator for Multi-mission Synergies Exploration (OSSSMOSE) simulations would have been completely impossible at this time within the VO due to the technical development status and computational demand requirements (ref. D3.5).

LUT tables for Total Ozone Column (TOC) are a contribution of BIRA. The aim is to include estimates of smoothing and collocation uncertainties in the VO in the most efficient way. The current proposed solution for total ozone columns uses LUTs based on full OSSSMOSE output and provided as hdf5 files. This provides realistic collocation uncertainties that are typical of a given location and season but may not be truly representative of the instantaneous uncertainty depending upon the variance in the fields.

Smoothing uncertainties and mean errors can be calculated in the VO by interpolation of these LUTs. This can be done and added to the data at the time of data ingestion in the VO. Collocation uncertainties can be calculated by interpolation at the moment the user is setting his/her collocation criteria.

LUTs of smoothing uncertainties (spread on errors) and mean errors (a.k.a. bias) have been computed by simulating 1,000,000 smoothing errors across the globe over 6 years of IFS-MOZART data and gridding those according to meaningful coordinates:

- **ZSL-DOAS sunrise measurements**: zonal monthly means for 1 climatological year -> f(lat, month)
- **ZSL-DOAS sunset measurements**: zonal monthly means for 1 climatological year -> f(lat, month)
- **Direct sun (Brewer & Dobson, FTIR) daily mean** measurements: zonal monthly means for 1 climatological year -> f(lat, month)
- **Direct sun (Brewer & Dobson, FTIR) individual** measurements : zonal monthly means for 1 climatological year and 7 SZA ranges -> f(lat,month, SZA)

In a similar solution, LUTs for IASI-RAOB for collocation uncertainty and vertical smoothing come as a contribution of UNIBG/CNR. A detailed description of the derivation and usage of LUTs is given in deliverables D3.5 and D3.6.

The LUTs belong to the "uncertainty assessment data", Figure 3.1. They are stored in the VO database in JSON (originally Hdf5) format and used by the Data Ingestion Script and by the collocation engine during the data pre-processing stage (Figure 3.2).

[3.3] Data, metadata and Data Ingestion Script (DIS)

Both the data and metadata for the VO are given off-line. This means that the VO does not have any direct access to on-line data repositories for this VO software version. The data is offered by the project partners for demonstration only and can be used in accordance with the policies and restrictions given by the original data providers (e.g. GRUAN or NDACC networks, EUMETSAT).

The original data comes in diverse formats. Using it all in one application and one database has been challenging, but has been realized with data harmonisation and ingestion software (DIS) (see Appendix B for a detailed description), which forms one of the key components of the VO.

The user can also export the data. This feature is a key identified user requirement. Currently, the user has the option to export any graphical image visible on the graphs plot pane using *plotly.js* standard features, by clicking the "save image" icon at the upper part of the graph

area. Whatever is visible on the screen can be saved to the user's local computing environment.

The user can also extract any datasets currently shown (by clicking "export data" in the upper part of the Data selection pane) in their original format, either single or collocated, as zipped netCDF files. Potentially, it is possible to offer access to the VO (and its database) for on-line use driven with third party scripts (and RESTful protocol). This means, interacting with the VO without its GUI. This feature is not realised in this early version of the VO, but it is a viable possibility for an operational VO under the Copernicus service in future.

[3.4] GUI (functional overview)

The technical design of the VO and GUI is based on user requirements derived from the user survey and two user workshops. The user survey confirmed many assumptions made in the proposal, in particular, it reinforced the import of core functionalities such as comparison of data sets, radiative transfer capability, selection tools for data and data formats.

A new requirement from the survey is the usage of the VO for comparison of reference measurements with climate model data, which will not be implemented within GAIA-CLIM, due to resource limitations, but will be kept as a potential extension towards a climate service usage of the VO under Copernicus if taken forwards.

The GUI offers the means for accessing and filtering Earth observational data recorded in the VO database (including an option for tightening the collocation criteria), visualization of the results (plotting different statistics, uncertainties, collocation mismatch, and smoothing errors), extracting graphical images and different collocated datasets by user-defined criteria. By default, all observations in the VO database are pre-collocated (within 6 hours and 500 kilometres). These collocations are calculated with the collocation tool developed by EUMETSAT.

The main part of the VO GUI has been developed as a one-page application. It has a main menu with sub-menus "data selection", "tutorials" and "metadata". Clicking on the GAIA-CLIM logo or the welcome message will open the GAIA-CLIM home page www.gaia-clim.eu in a new tab where the user can read about the project and access its public documentation (public deliverables).

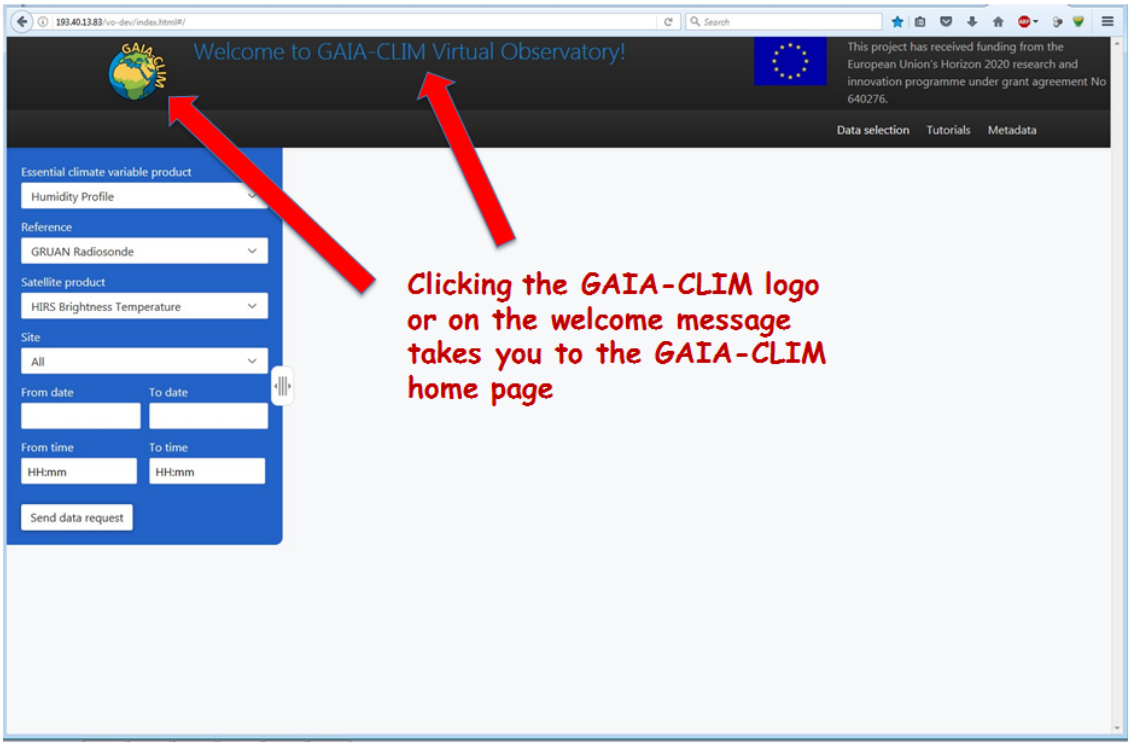

Figure 3.4.1. General view (VO main page):

The "tutorials" sub-menu provides links to uploaded audio-visual materials that serve as a general overview (introduction to the VO) and provide guided examples of how to use the VO with metadata and observational data.

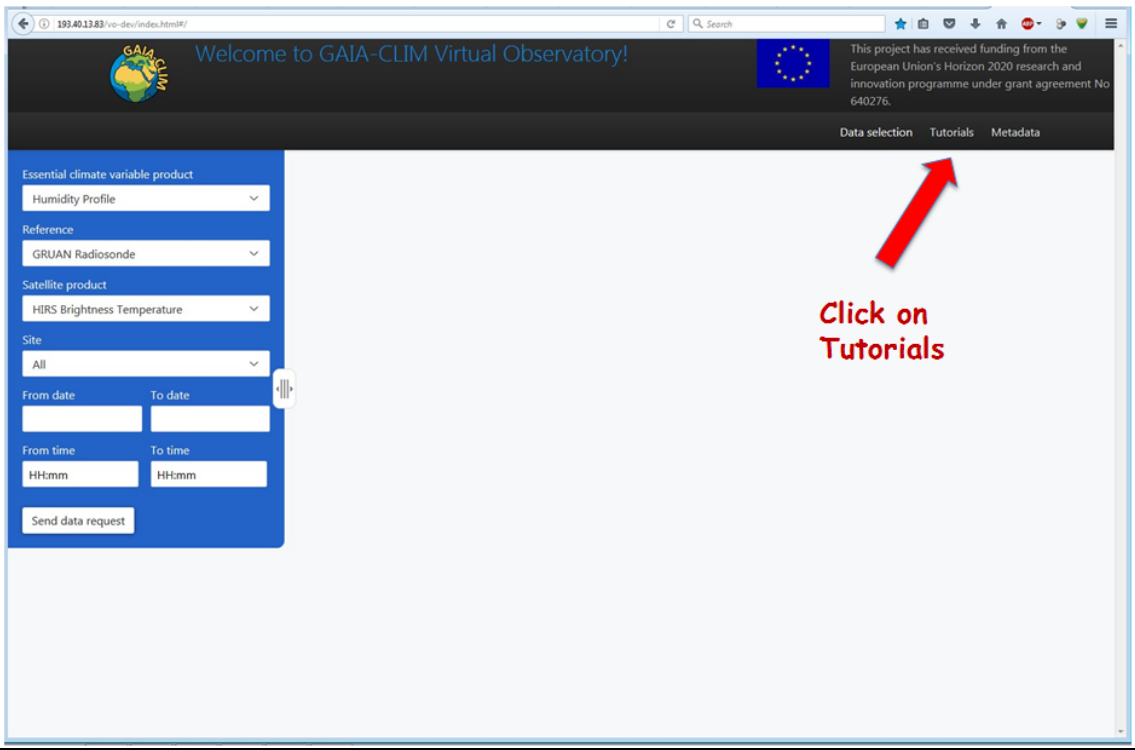

Figure 3.4.2. Get on-line tutorials:

Selecting “metadata”, the user is directed to the metadata visualization tool developed by CNR. Here the user can investigate what kind of data is available, where is it available, for which time period and many details about the observations at the site. Additionally, the user can check the quality of the networks presented in form of the maturity matrix as detailed in the deliverables of WP1[1].

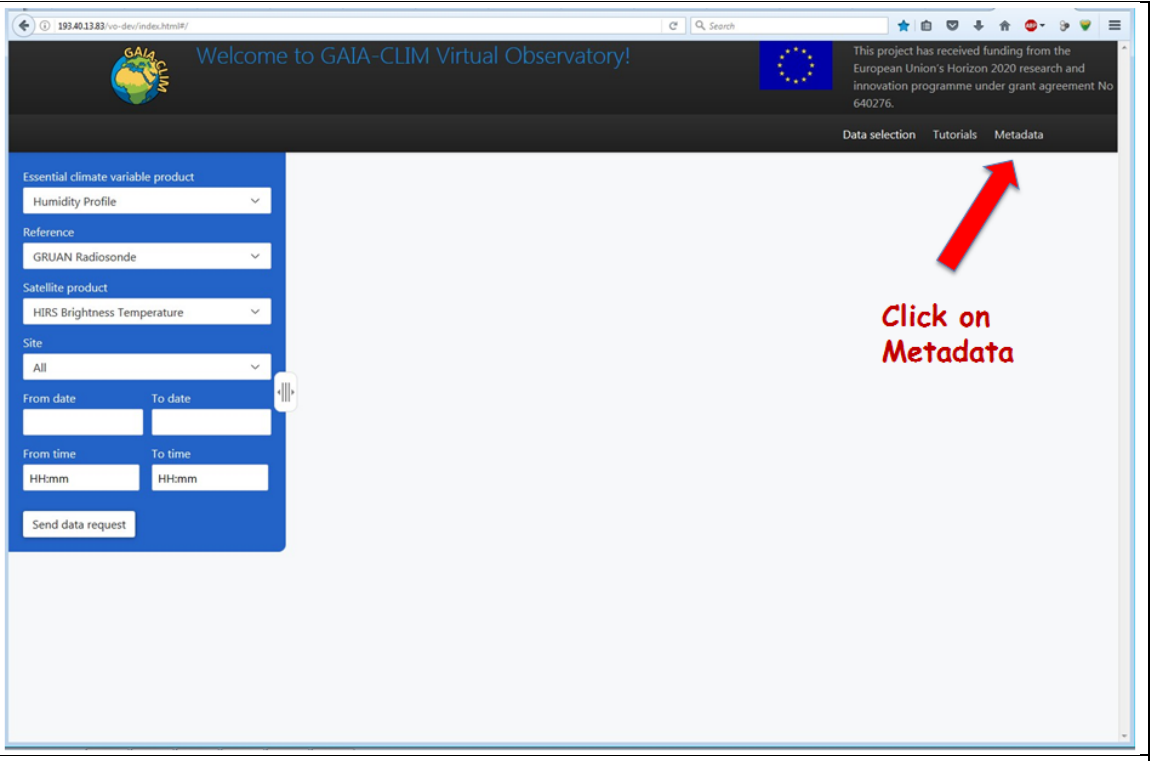

Figure 3.4.3. How to reach the Metadata view

A detailed description of the metadata tool is provided by CNR (see D1.9). This short overview is restricted to screenshots with explanatory comments overlaid in red.

[1] The Maturity Matrix Assessment is available online: http://www.gaia-clim.eu/page/maturity-matrix-assessment

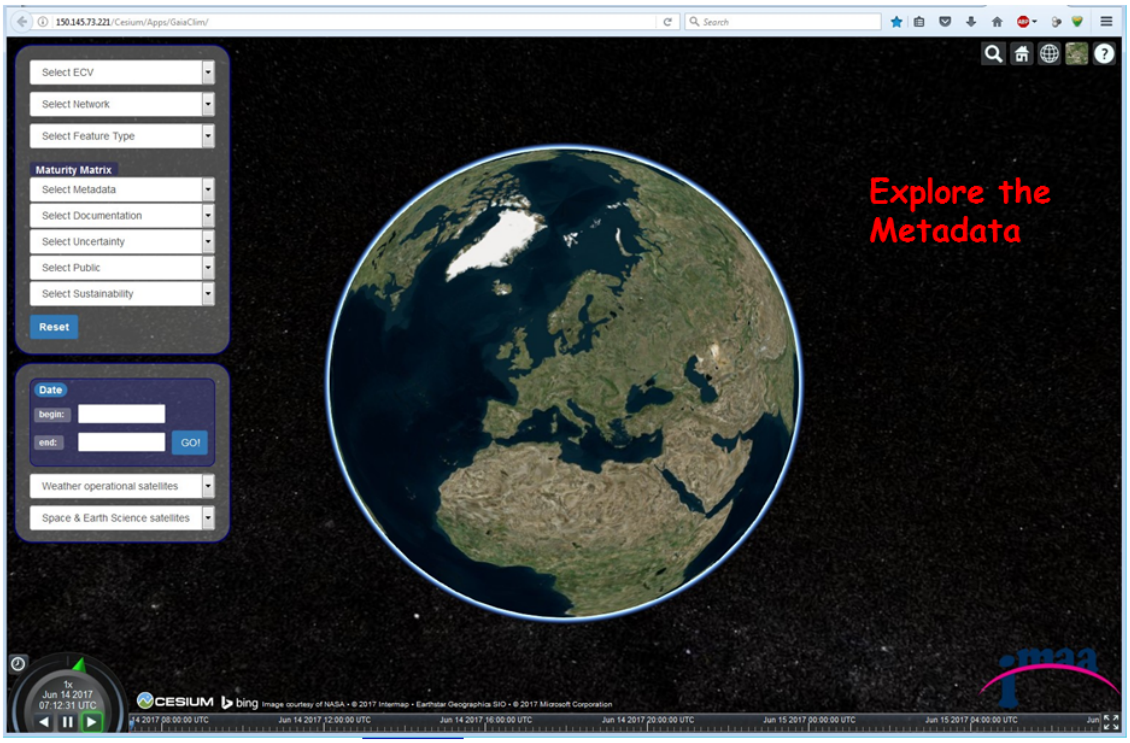

Figure 3.4.4. Metadata main menu

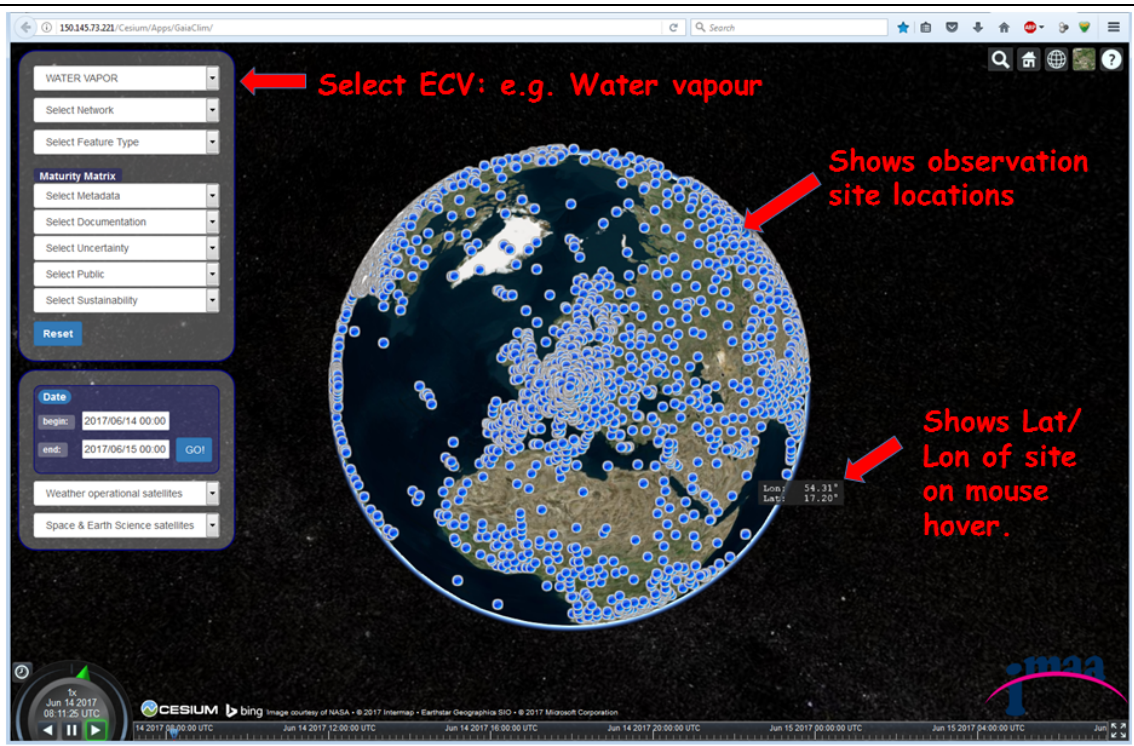

Figure 3.4.5. Metadata main menu

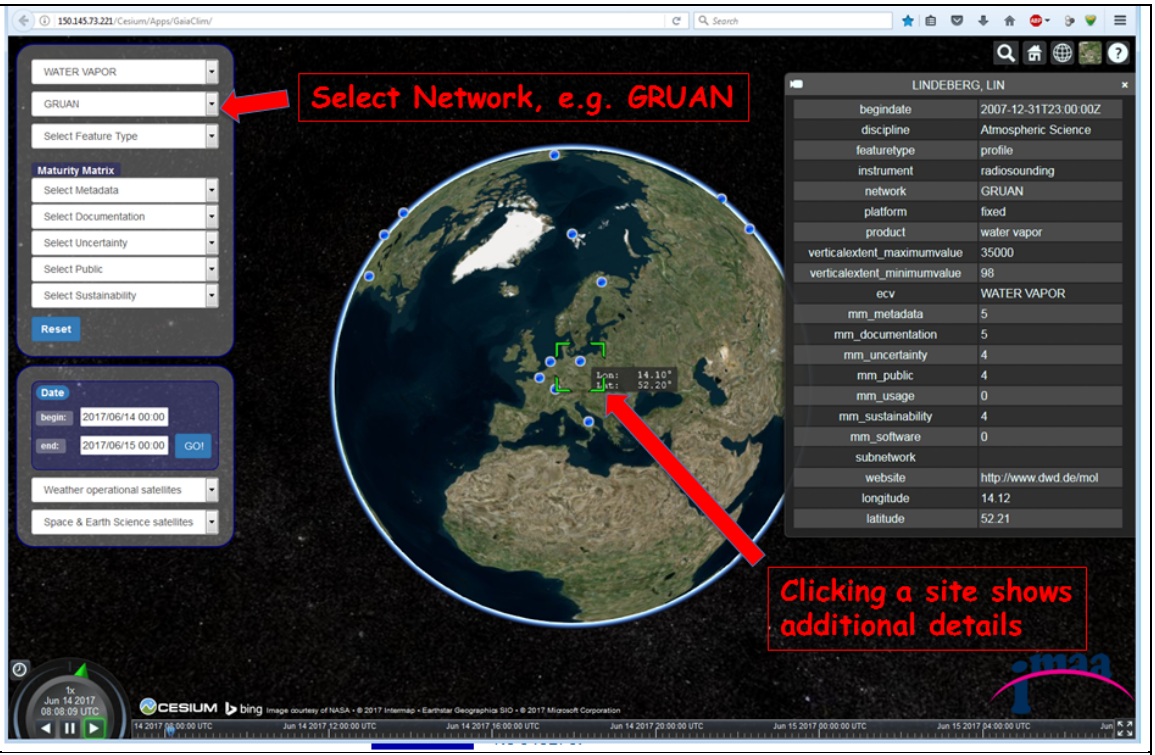

Figure 3.4.6. Metadata main menu

The Metadata visualization tool has a capability to animate satellite overpasses and footprints. Needed parameters can be selected from the lower panel at the left side of the web-page.

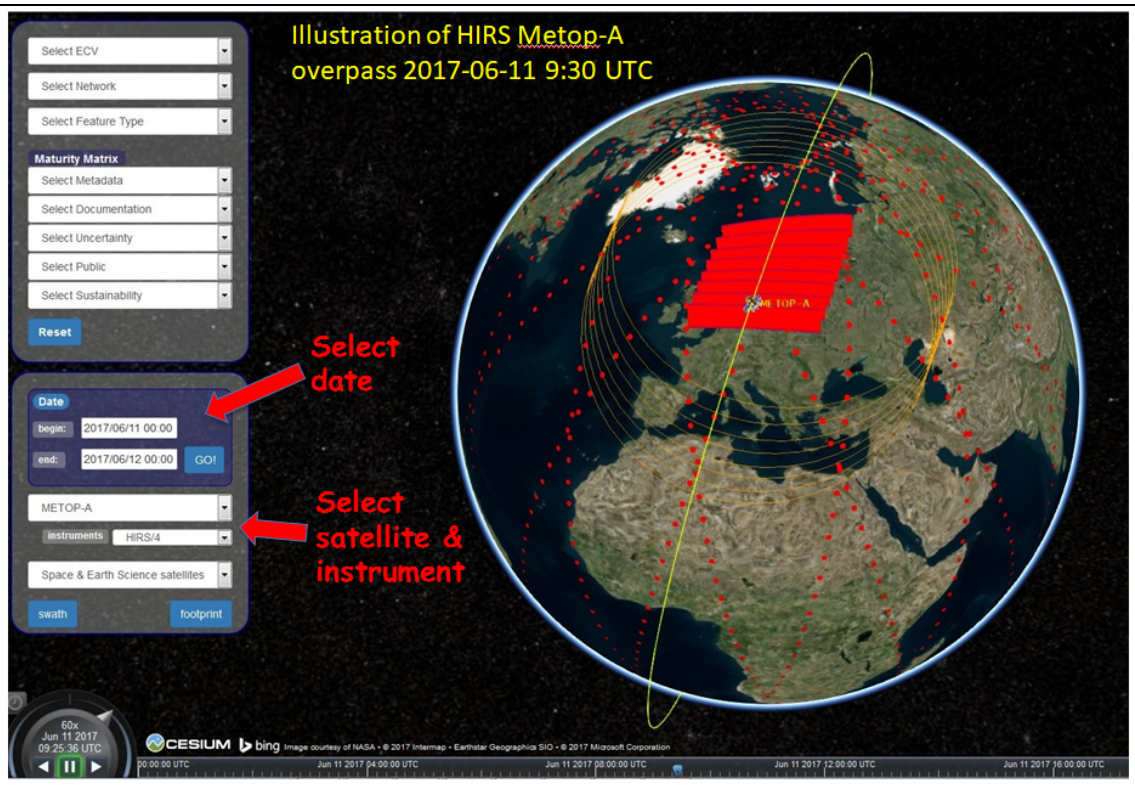

Figure 3.4.7. Metadata main menu. Animation of overpasses

Assuming that the user now has some idea of what data he or she is interested in and for what observation types, periods and locations these may be available facilitated by the metadata tool, the user may then proceed to interact with the main observational data base featuring collocation data.

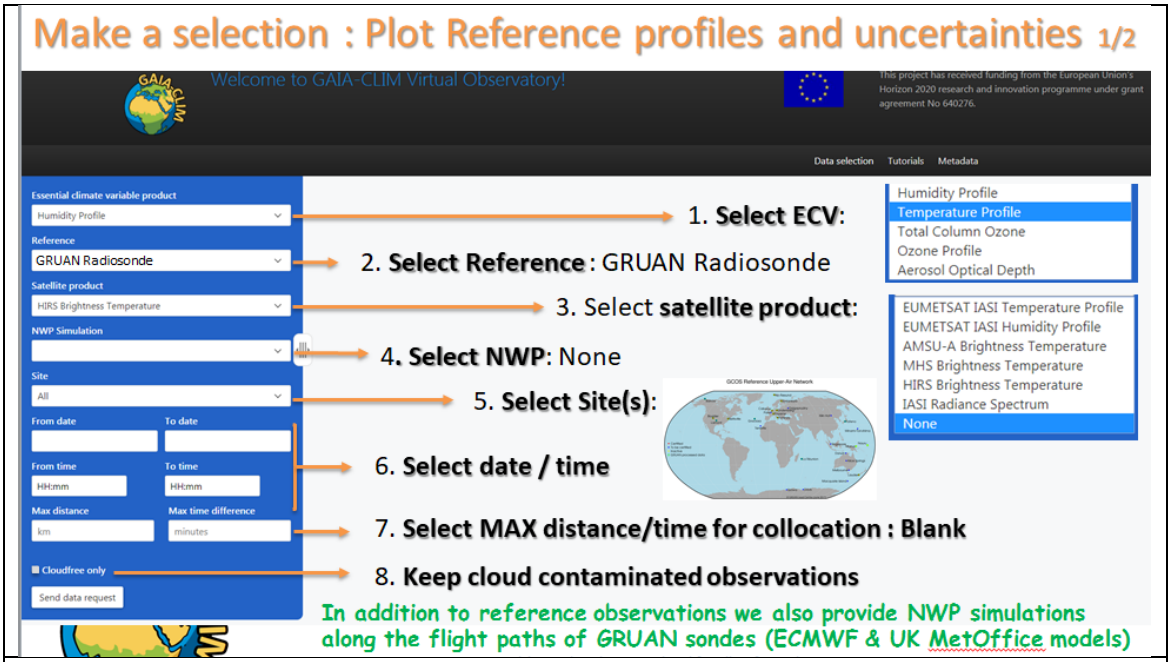

Figure 3.4.8. Choosing "data selection" from VO main menu and refining the search

For working with observational data, a user starts from the "essential climate variable product" sub-menu. It is not a requirement to query the metadata first. If the user knows already what is available and what is needed for a certain investigation he or she can start directly with selecting ECVs with different filtering options as provided in the "data selection" pane.

The user is given the option not only to look at collocation data, but also to explore the original data from a single type of observation. For example, the pressure, temperature, humidity or wind profiles from any particular radiosonde can be plotted. However, the most advanced feature and likely the most used application for the VO is to study collocation data, for example the HIRS-4 brightness temperatures from Metop-A, Metop-B and NOAA-19 satellites versus GRUAN radiosondes after deriving the corresponding brightness temperature for the latter. In the first example the user must choose "none" in the "satellite product" section.

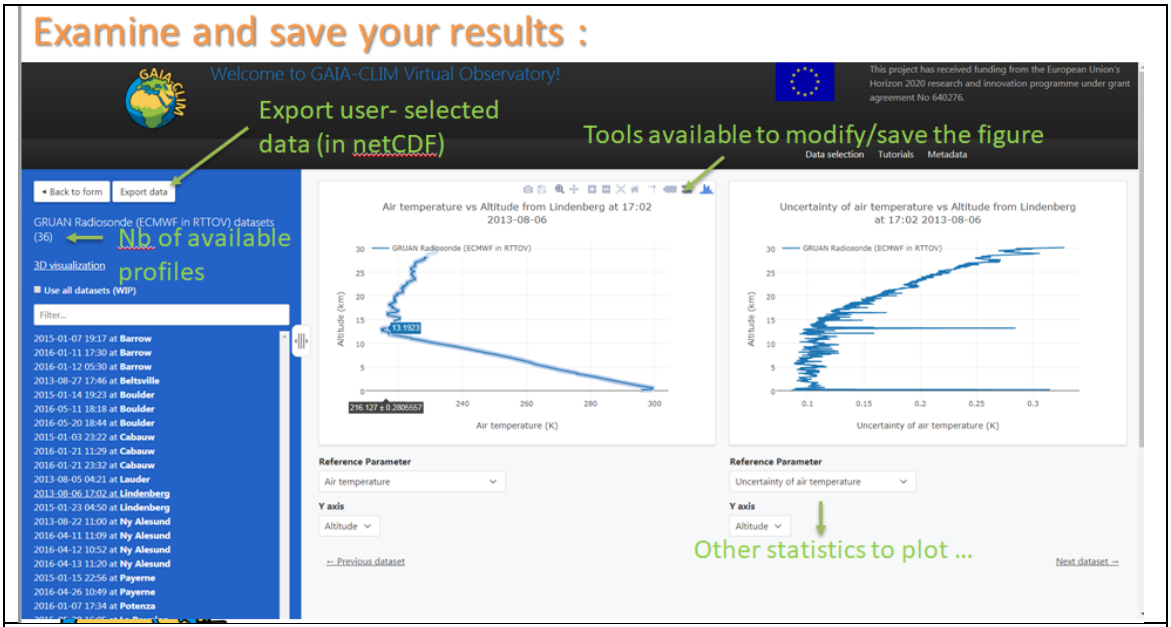

Figure 3.4.9. Plotting the results

After “send data request”, the user will see a number of available profiles according to the filtering criteria. Actual data plots can be seen on the right side (plot area). The user can choose between different parameters (available from this profile) and axis.

Zooming and saving of the figures can be chosen from the upper menus above each plot.

Between “data selection” and plot area the user can find a switch to show the plots only. This option is used to give more space for working with graphical images.

Working with collocated data:

Filtering collocated data in “data selection” area does not differ from works with single profiles, but the user must choose a “satellite product” instead of “none”.

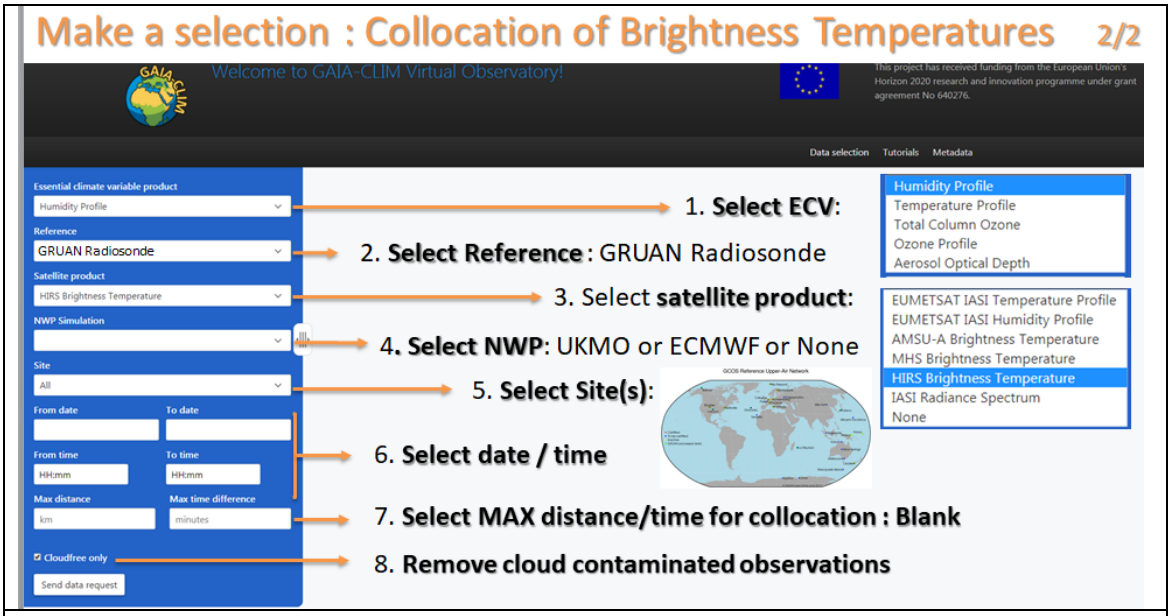

Figure 3.4.10. Working with collocated data, example for brightness temperatures

The results will be plotted in a similar manner.

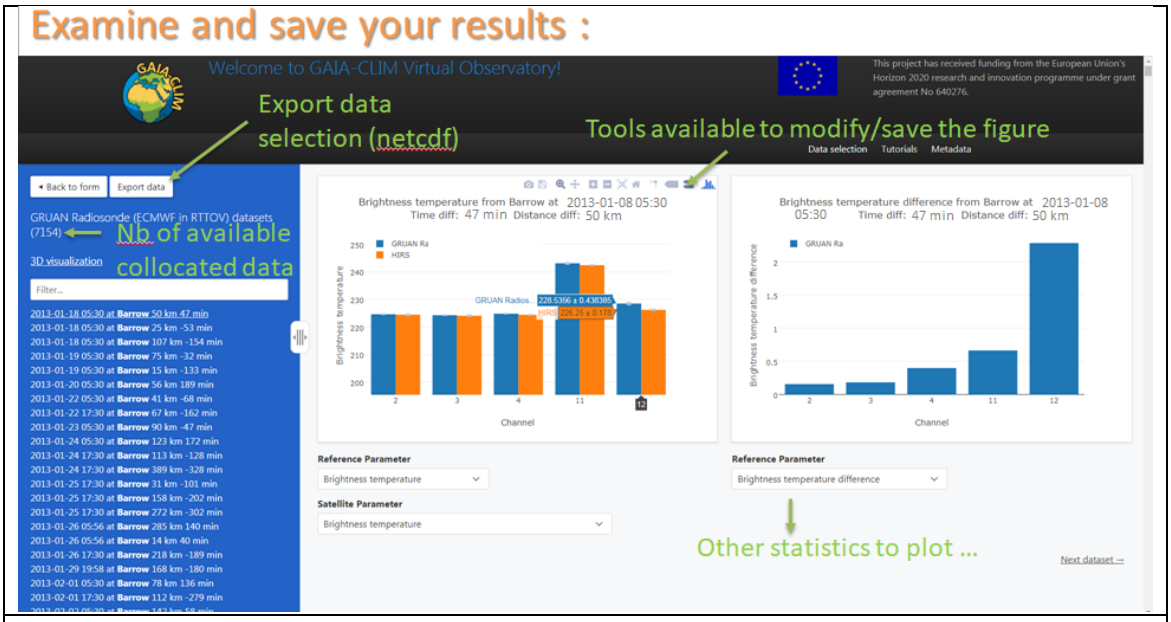

Figure 3.4.11. Working with collocated data

The user can plot different statistics either for a single dataset (by clicking on one of the datasets) or for all datasets chosen (by clicking “use all datasets” button) on the data filtering and selection pane. In the latter case the GUI plots statistics for the whole dataset chosen and for the parameters chosen in the “statistics to plot” and “satellite parameter” drop-down menus.

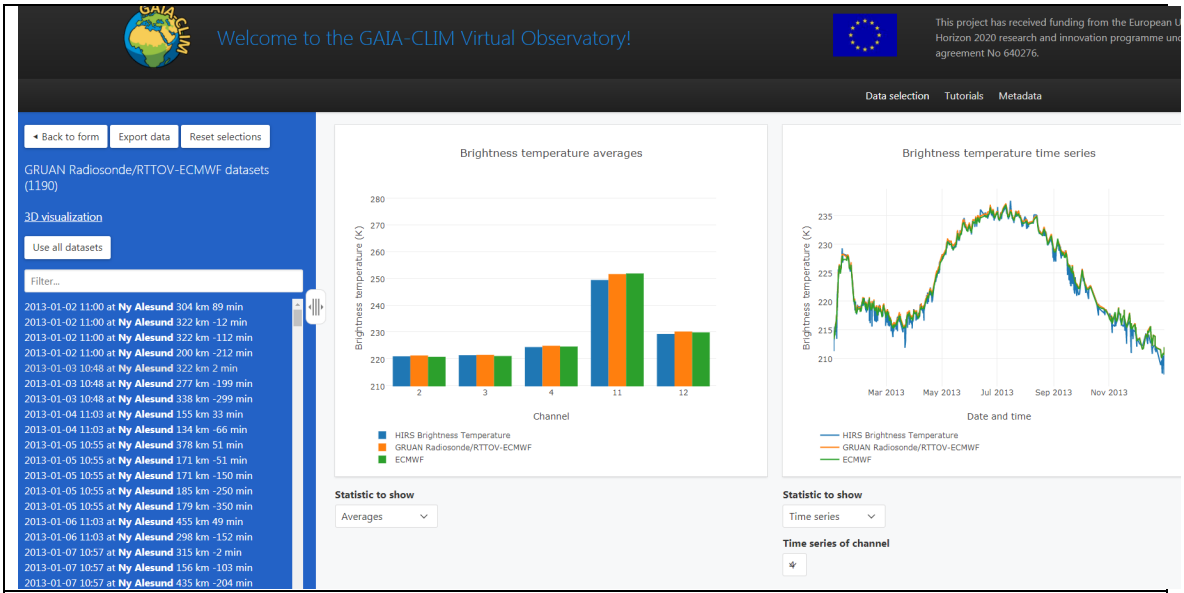

Figure 3.4.12. “Use all dataset” and plot statistics functionality: average (on the left) and time series (on the right) of brightness temperatures

[4] Summary and recommendations

The WP5 subtask of creating the GUI was led by TUT with significant contributions from EUMETSAT. Metadata visualization is fully developed by CNR and implemented in CESIUM. A more tight integration with the VO is technically possible, but will only be pursued if it serves the interests of the user community and if sufficient resources are available to do so. Technically, the CESIUM engine could add many nice additional features to the VO (related to satellite overpasses).

Detailed descriptions of how to use and further develop the VO belong naturally to the forthcoming User Guide (D5.5) and the video tutorials will then also be updated in accordance with the current status of the VO.

Glossary

CESIUM	Cross-platform virtual globe for dynamic-data visualisation in the space and defense industries
DOAS	Differential Optical Absorption Spectroscopy
ECV	Environmental Climate Variable
EO	Earth Observation
FTIR	Fourier Transform Infrared Spectroscopy
GRUAN	GCOS Reference Upper-Air Network
GUI	Graphical User Interface
ISO	International Standard Office
NetCDF	Network Common Data Form
NDACC	Network for Detection of Atmospheric Composition Change,
VO	The Virtual Observatory of GAIA-CLIM
WMO	World Meteorological Organization

APPENDIX A: The Virtual Observatory and Tutorials on the internet

The VO server is presently located on a server at TUT:

http://193.40.13.83/vo/index.html#/

After clicking on the central puzzle piece labelled “VO” one enters the interactive area of the VO:

http://193.40.13.83/vo/index.html

https://youtu.be/Qt4edW3A8hc GAIA-CLIM.eu tutorial part 1: Introduction

Tutorial part 1 (Introduction): The Virtual Observatory of the GAIA-CLIM project
http://www.gaia-clim.eu

https://youtu.be/OeshL9IVTKc GAIA-CLIM.eu tutorial part 2: Metadata

Tutorial part 3 (Metadata): The Virtual Observatory of the GAIA-CLIM project
http://www.gaia-clim.eu

https://youtu.be/MKj0Y00KqMY GAIA-CLIM.eu tutorial part 3: Observational Data

Tags: Earth Observation, Climate Monitoring, Satellite Validation, Meteorology, GAIA-CLIM

APPENDIX B: Data Ingestion Scripts

The Data Ingestion Scripts (DIS) implemented in the Python language have been developed for ingesting the information from source files into the DB of the VO. The DIS can handle observational data files represented in the netCDF-format. In addition, the scripts can be used for ingesting Look-Up Tables (LUT) in Hdf-format and XML-files represented in the WIGOS and ISO-19115 formats. However, information from source files is not directly ingested as it is, but is converted into a unified format. The unification is applied for field names as well as for values (e.g. conversion to SI-units). Moreover, by using DIS it is assured that all documents of the same kind share similar structure in the database.

Altogether, six scripts are used for the data ingestion (scripts added below):

1) ingestion_main.py – after ensuring the data is not already ingested, the main script calls other scripts to detect the format, read and homogenise the initial data, and ingest the data into the database based on type and format;
2) config.py – all environment variables as well as paths to conversion dictionaries and output files containing information about the ingestion results are defined here;
3) general_input.py – a script for detecting the initial data format and reading pre-defined dictionaries including field names, measurand variables and units for homogenisation purposes (format-specific txt-files for ISO-19115, WIGOS and CCI-CF);
4) metadata_conversion.py – all that have a part in reading and homogenising metadata from initial data in ISO-19115, WIGOS and CCI-CF formats is included here;
5) measurement_conversion.py – for reading and homogenising measurand variable values and units included in initial netCDF files. Any restrictions based on quality information are also considered, such as the *qcflag* and *qcinfo* values in case of the GRUAN processor;
6) LUTs_conversion.py – for reading and homogenising LUTs in Hdf-format.

During the conversion carried out using metadata_conversion.py and measurement_conversion.py, when needed, any original field from the source files is split into several different fields. Also, in order to make data querying more convenient at a later stage, several other fields may be added, such as the field in a metadata document containing the list of all the variables, the values of which have been estimated during the observations, links between co-located observations, uncertainties calculated based on LUTs, the coordinate field together with the 2dsphere index for geospatial queries, etc. In order to retrieve the original data file from the server, a field including the filename and path in the server is added. During the ingestion, initial files within a subfolder are moved to a folder consisting all previously ingested files. All rejected files are stored separately to ease the manual inspection. Output file containing information about the ingestion results is written. The user will be notified about any failures or new undefined data fields identified during the ingestion.

DIS ingests metadata and observational data into separate Mongo DB collections (Figure B1). Regarding observational data, one document per observational variable is created. In the VO

database, each piece of observational data is linked with its metadata using the Mongo ObjectId value. Co-located observations are linked in the same way (Figure B2). However, in the case of co-located data, an additional collection has been created. The purpose of this collection is to group together co-located measurements by location and time, each document including links to their metadata documents (Figure B3, Figure B4). This method significantly simplifies and speeds up the queries for retrieving collocations by station, start date and time, or by collocation criteria – time window and spatial difference. It must be stressed that collocations in the database are not necessary stored in pairs. If there is a previously ingested collocation data that matches with the new collocation by location and time and contains information for the same ECV, the latter is added to the group (three methods included in an example in Figure B3).

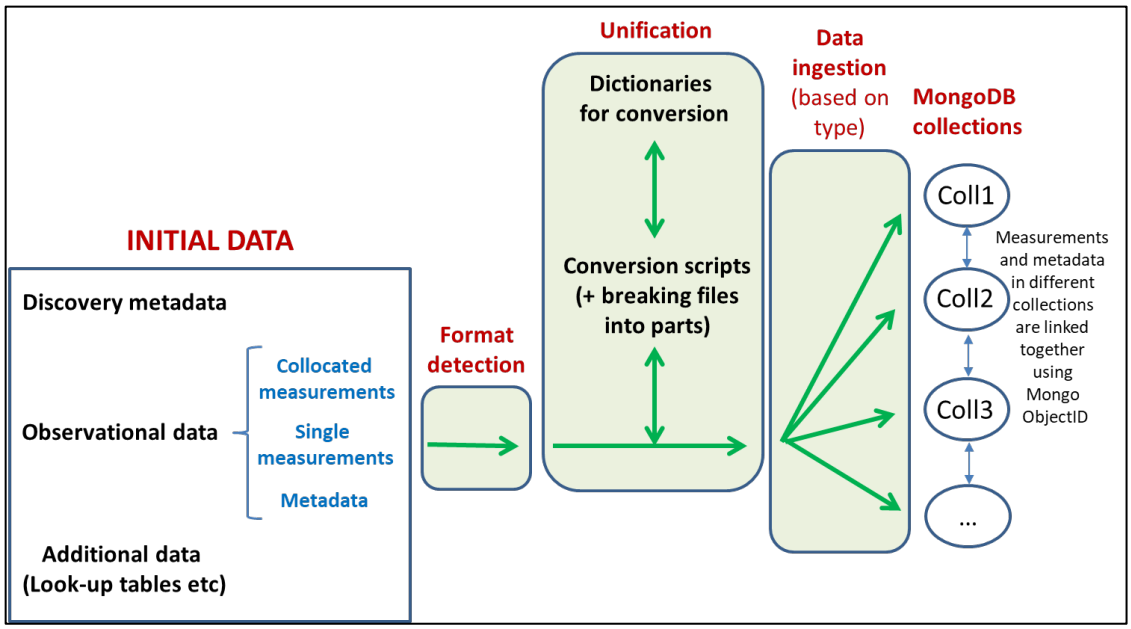

Figure B1. An overall principle of data ingestion.

```
{"_id" : ObjectId("59a7b109ddb59344e45f8450"),
  "measurand_variable" : "Brightness temperature",
  "metadata_CCI_CF_ID" : ObjectId("59a7b108ddb59344e45f8447"),
  "start_date" : "2013-01-06",
  "start_time" : "17:30:05",
  "dimensions" : ["nRefCollocations", "Channel"],
  "y_axis_variables" : ["Channel"],
  "name" : "ref_bt",
  "units" : "K",
  "long_name" : "ref_brightness_temperature",
  "scale_factor" : "1.0",
  "valid_min" : "0.0",
  "valid_max" : "10000.0",
  "fill_value" : "99999.0",
  "values" : [
    [
      232.776580810547,
      229.678085327148,
      228.27848815918,
      227.566696166992,
      234.177001953125,
      239.776153564453,
      245.209426879883,
      250.572357177734,
      241.832504272461,
      250.560012817383,
      247.555313110352,
      234.758178710938,
      245.95703125,
      239.742584228516,
      233.455383300781,
      231.04150390625,
      249.326599121094,
      251.030227661133,
      251.24462890625
```


Figure B2. An example document in the "collocations" collection (brightness temperature values from ECMWF simulation). Metadata ObjectId marked with red is used in Figure B3 and Figure B4.

```
{ "_id" : ObjectId("59a7b109ddb59344e45f8464"),
  "simulations" : {
    "ECMWF" : {
      "instrument" : "NWP radiosonde simulation for HIRSP",
      "product" : "RTTOV Brightness Temperature from NWP",
      "NWP_model" : "ECMWF Atmospheric Model",
      "station_platform_name" : "Barrow",
      "start_date" : "2013-01-06",
      "start_time" : "17:30:05",
      "collocation_time_diff_sec" : -8778,
      "collocation_space_diff_km" : 60.5577201843262,
      "metadata_ID" : ObjectId("59a7b108ddb59344e45f8447"),
      "monitored_metadata_ID" : ObjectId("59a7b109ddb59344e45f8452")}},
  "satellite_products" : {
    "HIRS Brightness Temperature" : {
      "instrument" : "HIRS",
      "product" : "Level_1c Brightness Temperature",
      "start_date" : "2013-01-06",
      "start_time" : "20:18:45",
      "cloud_flag" : [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1]}},
  "references" : {
    "GRUAN Radiosonde/RTTOV-ECMWF" : {
      "instrument" : "Radiosonde converted to BT by RTTOV",
      "product" : "RTTOV Brightness Temperature",
      "NWP_model" : "ECMWF Atmospheric Model",
      "station_platform_name" : "Barrow",
      "start_date" : "2013-01-06",
      "start_time" : "17:30:05",
      "collocation_time_diff_sec" : -8773,
      "collocation_space_diff_km" : 59.9454574584961,
      "metadata_ID" : ObjectId("59a7b7fcddb59344e462ebe6"),
      "monitored_metadata_ID" : ObjectId("59a7b7fcddb59344e462ebf1")}}}
```


Figure B3. An example document in the "collocation_IDs" collection, containing groups of collocations. Metadata ObjectId marked with red is used in Figure B4 (ECMWF simulation).

```
{"_id" : ObjectId("59a7b108ddb59344e45f8447"),
   "monitored_or_reference" : "reference",
   "collocation_IDs" : [
      ObjectId("59a7b109ddb59344e45f8452")],
   "metadata_origin_format" : "CCI-CF",
   "measurand_variable_list" : [ "Column", "Row", "Latitude", "Longitude", "Distance difference", "Time", "Time difference", "Cos ratio",
"Brightness temperature", "Uncertainty of brightness temperature"],
   "collocation_software" : "STAMP, version: 1.0.6-r106",
   "ascent_duration" : 99999.0,
   "corresponding_level" : 108.0,
   "satellite" : "RTTOV METOP2 NWP simulation",
   "instrument" : "NWP radiosonde simulation for HIRSP",
   "product" : {"name" : "RTTOV Brightness Temperature from NWP"},
   "start_date" : "2013-01-06",
   "start_time" : "17:30:05",
   "input_file" : "GProc-EC-metop_2_hirs_BAR-2013010617.nc",
   "measuring_system_altitude" : 99999.0,
   "measuring_system_latitude" : 71.3215484619141,
   "measuring_system_longitude" : -156.622512817383,
   "pressure_level_hPa" : 300.0,
   "processing_software" : "Unknown NWP version",
   "station_platform_name" : "Barrow",
   "surface_obs_pressure" : 99999.0,
   "time_offset_s" : 1794.0,
   "general" : {"site_code" : "BAR"},
   "NWP" : {"model" : "ECMWF Atmospheric Model"},
   "wavenumbers" : [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
   "max_time_diff" : 21600.0,
   "max_diff_VZA" : 80.0,
   "upper_latitude" : 80.0,
   "lower_latitude" : -80.0,
   "east_longitude" : 170.0,
   "west_longitude" : -170.0,
   "grid_latitude_step" : 2.0,
   "grid_longitude_step" : 2.5,
   "store_radiometric_info" : 1.0,
   "Channel" : [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19],
   "collocation_space_diff_km" : 60.5577201843262,
   "collocation_time_diff_sec" : -8778,
   "collocation_space_diff_km_noneff" : 57.9043159484863,
   "latitude_noneff" : 71.2220077514648,
   "longitude_noneff" : -156.695678710938,
   "original_filename":"gruan-300hPa-metopa-hirs-ec/GProc-EC-
metop_2_hirs_BAR2013010617+COLLOC_Ref_300hPa+MetOpA_HIRS+GRUAN_NWP-ECMWF_C_EUMS_20130106201845_20130106173005.nc"}
```


Figure B4. An example document in the "metadata_collocations" collection (ECMWF simulation).

ingestion_main.py

```
import xml.etree.cElementTree as ET
import netCDF4
import metadata_conversion as metadata_conversion
import measurement_conversion
import LUTs_conversion
import config
import collections
import general_input
import os
import time
import shutil
import logging
from bson.objectid import ObjectId


def ingest_data():

    db, coll_wigos, coll_iso, coll_cci, coll_LUTs, coll_meas, coll_colloc, coll_cci_colloc, coll_cci_colloc_IDs, mainpath, output_dir,
wigos_fields_filepath, iso_fields_filepath, netcdf_global_attr_fields_filepath, \
    netcdf_variable_attr_fields_filepath, variable_names_units_filepath, to_be_ingested_dir, ingested_dir, rejected_dir =
config.paths()

    result, resultpath, warnings, warningspath = config.logfile_setup(output_dir, 'w')
    result.info(' Ingestion started at ' + time.strftime('%d/%m/%Y %H:%M:%S'))

    ## Read input files with unified fieldnames, variables, attributes and units
    iso_field_names, iso_xpaths, iso_field_IDs = general_input.xml_fields(iso_fields_filepath)
    wigos_field_names, wigos_xpaths, wigos_field_IDs = general_input.xml_fields(wigos_fields_filepath)
    netcdf_global_attr_field_names, netcdf_global_attr_field_IDs = general_input.netcdf_fields(netcdf_global_attr_fields_filepath)
    netcdf_variable_attr_field_names, netcdf_variable_attr_field_IDs =
general_input.netcdf_fields(netcdf_variable_attr_fields_filepath)
    variable_names, units, units_100 = general_input.variable_names(variable_names_units_filepath)


    files_ingested = general_input.files_already_in(ingested_dir)
    CCI_CF_ID = ''
    ISO_19115_3_ID = ''
    LUTs_ID = ''
    files_ingested_before = files_ingested[:]
    count_files_ingested_before = len(files_ingested_before)
    count_files_to_be_ingested = 0
    cloudfree_percent = 0

    for subdir, dirs, files in os.walk(to_be_ingested_dir):
        for f in files:
            origFile = os.path.join(subdir, f)
            subdir_base = os.path.basename(os.path.normpath(subdir))
            write_dir = ingested_dir + subdir_base
            count_files_to_be_ingested = len(files)
            allow_ingestion = True


            if f not in files_ingested:
                dataformat, coll, coll_meas, xml_root, logFile, field_names, xpaths, field_IDs = general_input.detect_format(origFile,
iso_field_names, iso_xpaths, iso_field_IDs, wigos_field_names, wigos_xpaths, wigos_field_IDs, netcdf_global_attr_field_names,
netcdf_global_attr_field_IDs, cloudfree_percent, allow_ingestion, subdir_base)

                if dataformat == 'ISO 19115-3':
                    coll_iso = coll
                    f_in = open(mainpath + 'ISO_19115_3_files_ingested.txt', 'w', encoding='utf-8')
                    f_in.write(time.strftime('%d/%m/%Y %H:%M:%S ') + f + '\n')
                    f_in.close()
                    tree = ET.ElementTree(file=origFile)
                    root = tree.getroot()

                    with open(origFile, 'r') as xml:
```

```
                xml_text=xml.read().replace('\n', '')
                xml_text = ' '.join(xml_text.split())

            field_values_found, field_names_found, xpaths_found, field_IDs_found, nb_of_param =
metadata_conversion.find_fields_iso_wigos(dataformat, root, origFile, logFile, field_names, xpaths, field_IDs, variable_names)
            for nb in range(nb_of_param):
                dic_iso, stn_name, network_name = metadata_conversion.iso_wigos_to_dict(dataformat, coll_iso, origFile,
xml_text, field_values_found, field_names_found, xpaths_found, field_IDs_found, nb, xml_root, f, subdir_base)
                ISO_19115_3_ID = general_input.insert_data_to_mongo(dic_iso, coll_iso, f)
            general_input.make_dir(write_dir)
            shutil.move(origFile, os.path.join(write_dir, f))
            general_input.del_empty_dir(subdir)
            files_ingested.append(f)

        if dataformat == 'CCI-CF':
            coll_cci = coll
            colloc_IDs = []
            meas_types = []
            drop_down_choices = []
            start_dates = []
            start_times = []
            station_platform_names = []


            if coll_meas == coll_colloc:
                c_list = ['reference', 'monitored']

                coll_cci = coll_cci_colloc
                dic_cci_short = collections.OrderedDict()
            if coll_meas != coll_colloc:
                c_list = ['single_measurement']
            for c in c_list:
##                print (c)
                field_values_found, field_names_found, field_IDs_found, stn_name, start_date, start_time,
measurement_observing_method, ind_by_eff_dist, cloudfree_percent, allow_ingestion =
metadata_conversion.find_fields_cci(dataformat, origFile, logFile, field_names, variable_names, field_IDs, c, cloudfree_percent,
allow_ingestion, subdir_base)

                dataformat2 = False
                for subdir2, dirs2, files2 in os.walk(to_be_ingested_dir):
                    for f2 in files2:
                        if f[:-7] in f2 and '.xml' in f2:            # try to insert matching WIGOS and CCI-CF files together into the db
                            origFile2 = os.path.join(subdir2, f2)
                            dataformat2, coll, coll_meas, xml_root2, logFile2, field_names2, xpaths2, field_IDs2 =
general_input.detect_format(origFile2, iso_field_names, iso_xpaths, iso_field_IDs, wigos_field_names, wigos_xpaths,
wigos_field_IDs, netcdf_global_attr_field_names, netcdf_global_attr_field_IDs, cloudfree_percent, allow_ingestion, subdir_base)
                            if dataformat2 == 'WIGOS':
                                coll_wigos = coll
                                tree2 = ET.ElementTree(file=origFile2)
                                root2 = tree2.getroot()
                                with open(origFile2, 'r') as xml2:
                                    xml_text2=xml2.read().replace('\n', '')
                                    xml_text2 = ' '.join(xml_text2.split())
                                field_values_found2, field_names_found2, xpaths_found2, field_IDs_found2, nb_of_param2 =
metadata_conversion.find_fields_iso_wigos(dataformat2, root2, origFile2, logFile2, field_names2, xpaths2, field_IDs2,
variable_names)

                                dic_wigos, stn_name, network_name = metadata_conversion.iso_wigos_to_dict(dataformat2, coll_wigos,
origFile2, xml_text2, field_values_found2, field_names_found2, xpaths_found2, field_IDs_found2, nb_of_param2, xml_root2, f2,
subdir_base)
                                WIGOS_ID = general_input.insert_data_to_mongo(dic_wigos, coll_wigos, f)


                                dic_cci = metadata_conversion.cci_to_dict(origFile, dataformat, coll_cci, field_values_found,
field_names_found, field_IDs_found, stn_name, f, subdir_base, c)
                                CCI_CF_ID = general_input.insert_data_to_mongo(dic_cci, coll_cci, f)
                                colloc_IDs.append(CCI_CF_ID)
```

```
                    dics_meas, measurand_variables = measurement_conversion.meas_nc_to_dict(origFile, coll_meas,
mainpath, variable_names, WIGOS_ID, CCI_CF_ID, stn_name, start_date, start_time, measurement_observing_method,
netcdf_variable_attr_field_names, netcdf_variable_attr_field_IDs, units, units_100, c, ind_by_eff_dist, subdir_base)
                    if measurand_variables != []:
                      for dic_meas in dics_meas:
                        measurements_ID = general_input.insert_data_to_mongo(dic_meas, coll_meas, f)
                      update_variable_list = {'measurand_variable_list': measurand_variables}
                      coll_cci.update({'_id':CCI_CF_ID}, {'$set': update_variable_list}, upsert=False)
                      coll_wigos.update({'_id':WIGOS_ID}, {'$set': update_variable_list}, upsert=False)
                      update_CCI_CF_ID = {'metadata_CCI_CF_ID': CCI_CF_ID}
                      coll_wigos.update({'_id':WIGOS_ID}, {'$set': update_CCI_CF_ID}, upsert=False)

                      update_pnan = {'programme_network_affiliation_name': network_name}
                      coll_cci.update({'_id':CCI_CF_ID}, {'$set': update_pnan}, upsert=False)
                      if network_name == 'EARLINET':
                        update_mom = {'measurement_observing_method': 'aerosol lidar'}
                        coll_cci.update({'_id':CCI_CF_ID}, {'$set': update_mom}, upsert=False)

                      try:
                        for itm in coll_iso.find({'station_platform_name':stn_name,
'programme_network_affiliation_name':network_name}):
                          ISO_19115_3_ID = itm.get('_id')
                          update = {"observation_data_present": "True"}
                          coll_iso.update({'_id':ISO_19115_3_ID}, {"$set": update}, upsert=False)
                      except AttributeError:
                        warnings.warning(' Station ' + stn_name + ' not found in collection metadata_ISO_19115_3')

                    if c == c_list[-1]:
                      general_input.make_dir(write_dir)
                      shutil.move(origFile2, os.path.join(write_dir, f2))
                      shutil.move(origFile, os.path.join(write_dir, f))
                      general_input.del_empty_dir(subdir)
                      files_ingested.append(f2)
                      files_ingested.append(f)


                if dataformat2 == False:      # if matching WIGOS metadata file was not found, insert only data from the nc-file
                  dic_cci = metadata_conversion.cci_to_dict(origFile, dataformat, coll_cci, field_values_found, field_names_found,
field_IDs_found, stn_name, f, subdir_base, c)
                  if allow_ingestion == True:
                    CCI_CF_ID = general_input.insert_data_to_mongo(dic_cci, coll_cci, f)
                    colloc_IDs.append(CCI_CF_ID)
                    if coll_meas == coll_colloc:

meas_type,drop_down_choice,dic_cci_short,start_date,start_time,meas_types_all,drop_down_menu_all,station_platform_name
= metadata_conversion.dic_cci_short(dic_cci, dic_cci_short)
                      meas_types.append(meas_type)
                      drop_down_choices.append(drop_down_choice)
                      start_dates.append(start_date)
                      start_times.append(start_time)
                      station_platform_names.append(station_platform_name)

                    network_name = ''
                    try:
                      network_name = dic_cci['programme_network_affiliation_name']
                    except (KeyError):
                      pass

                    dics_meas, measurand_variables = measurement_conversion.meas_nc_to_dict(origFile, coll_meas, mainpath,
variable_names, '', CCI_CF_ID, stn_name, start_date, start_time, measurement_observing_method,
netcdf_variable_attr_field_names, netcdf_variable_attr_field_IDs, units, units_100, c, ind_by_eff_dist, subdir_base)

                    if measurand_variables != []:
                      for dic_meas in dics_meas:
                        measurements_ID = general_input.insert_data_to_mongo(dic_meas, coll_meas, f)
                      update_variable_list = {"measurand_variable_list": measurand_variables}
                      coll_cci.update({'_id':CCI_CF_ID}, {"$set": update_variable_list}, upsert=False)
                      try:
                        if stn_name != '':  # update metadata_ISO_19115_3 document which has the same value of
'station_platform_name'
```

```
                        for itm in coll_iso.find({'station_platform_name':stn_name,
'programme_network_affiliation_name':network_name}):
                            ISO_19115_3_ID = itm.get('_id')
                            update = {'observation_data_present': 'True'}
                            coll_iso.update({'_id':ISO_19115_3_ID}, {'$set': update}, upsert=False)
                    except AttributeError:
                        warnings.warning(' Station ' + stn_name + ' not found in collection metadata_ISO_19115_3')

                    if c == c_list[-1]:
                        general_input.make_dir(write_dir)
                        shutil.move(origFile, os.path.join(write_dir, f))
                        general_input.del_empty_dir(subdir)
                        files_ingested.append(f)

                else:           ## if ingestion is not allowed
                    if c == c_list[-1]:
                        general_input.make_dir(rejected_dir)
                        shutil.move(origFile, os.path.join(rejected_dir, f))
                        general_input.del_empty_dir(subdir)


                if allow_ingestion == True:
                    for colloc_ID in colloc_IDs:
                        colloc_IDs_copy = colloc_IDs[:]
                        colloc_IDs_copy.remove(colloc_ID)
                        update = {'collocation_IDs': colloc_IDs_copy}
                        coll_cci.update({'_id':colloc_ID}, {"$set": update}, upsert=False)

                ## applies only for collocations_IDs collection
                if coll_meas == coll_colloc and allow_ingestion == True:
                    coll_cci_colloc_IDs_found = False
                    for mt in range(len(meas_types_all)):
                        if coll_cci_colloc_IDs_found != True:
                            if meas_types_all[mt] != 'satellite_products':
                                for itm in coll_cci_colloc_IDs.find({meas_types[1]+'.'+drop_down_choices[1]+'.start_date' : start_dates[1],
meas_types[1]+'.'+drop_down_choices[1]+'.start_time' : start_times[1],
meas_types_all[mt]+'.'+drop_down_menu_all[mt]+'.start_date' : start_dates[0],
meas_types_all[mt]+'.'+drop_down_menu_all[mt]+'.start_time' : start_times[0],
meas_types_all[mt]+'.'+drop_down_menu_all[mt]+'.station_platform_name' : station_platform_names[0]}):
                                    colloc_ID = itm.get('_id')
                                    update1 = {meas_types[0]+'.'+drop_down_choices[0]:
dic_cci_short[meas_types[0]][drop_down_choices[0]]}
                                    update2 = {meas_types[0]+'.'+drop_down_choices[0]+'.metadata_ID' : colloc_IDs[0]}
                                    update3 = {meas_types[0]+'.'+drop_down_choices[0]+'.monitored_metadata_ID' : colloc_IDs[1]}
                                    coll_cci_colloc_IDs.update({'_id':colloc_ID}, {"$set": update1}, upsert=False)
                                    coll_cci_colloc_IDs.update({'_id':colloc_ID}, {"$set": update2}, upsert=False)
                                    coll_cci_colloc_IDs.update({'_id':colloc_ID}, {"$set": update3}, upsert=False)
                                    coll_cci_colloc_IDs_found = True
                                    continue
                    if coll_cci_colloc_IDs_found == False:
                        CCI_CF_ID_short = general_input.insert_data_to_mongo(dic_cci_short, coll_cci_colloc_IDs, f)
                        update1 = {meas_types[0]+'.'+drop_down_choices[0]+'.metadata_ID' : colloc_IDs[0]}
                        update2 = {meas_types[0]+'.'+drop_down_choices[0]+'.monitored_metadata_ID' : colloc_IDs[1]}
                        coll_cci_colloc_IDs.update({'_id':CCI_CF_ID_short}, {"$set": update1}, upsert=False)
                        coll_cci_colloc_IDs.update({'_id':CCI_CF_ID_short}, {"$set": update2}, upsert=False)


            if dataformat == 'HDF':          ## in the case of LUTs
                if coll == coll_LUTs:
                    dics = LUTs_conversion.LUTs_to_dict(f, origFile, coll, mainpath)
                    LUTs_ID = general_input.insert_data_to_mongo(dics, coll_LUTs, f)
                    general_input.make_dir(write_dir)
                    shutil.move(origFile, os.path.join(write_dir, f))
                    general_input.del_empty_dir(subdir)
                    files_ingested.append(f)

                else:
                    result.error(' Not able to read data from HDF-file: ' + subdir_base + '/' + f)
                    warnings.error(' Not able to read data from HDF-file: ' + subdir_base + '/' + f)
```

```
            else:                       ## if this file has been ingested before
                if f in files_ingested_before:
                    result.error(' File already ingested: ' + subdir_base + '/' + f)
                    warnings.error(' File already ingested: ' + subdir_base + '/' + f)
                    general_input.make_dir(rejected_dir)
                    shutil.move(origFile, os.path.join(rejected_dir, f))
                    general_input.del_empty_dir(subdir)


    if CCI_CF_ID == '' and ISO_19115_3_ID == '' and LUTs_ID == '':
        result.warning(' No new data found!')
        warnings.warning(' No new data found!')

    result.info(' Ingestion ended at ' + time.strftime('%d/%m/%Y %H:%M:%S'))
    result.info(' Number of files ingested: ' + str(len(files_ingested)-count_files_ingested_before))
    result.info(' Number of files rejected: ' + str(count_files_to_be_ingested + count_files_ingested_before - len(files_ingested)))
    result.info(' Number of documents in LUTs collection: ' + str(coll_LUTs.count()) )
    result.info(' Number of documents in measurements collection: ' + str(coll_meas.count()))
    result.info(' Number of documents in metadata_CCI_CF collection: ' + str(coll_cci.count()))
    result.info(' Number of documents in metadata_ISO_19115_3 collection: ' + str(coll_iso.count()))
    result.info(' Number of documents in metadata_WIGOS collection: ' + str(coll_wigos.count()))
    result.info(' Number of documents in collocations collection: ' + str(coll_colloc.count()))
    result.info(' To see all warnings and errors, please see ' + output_dir + 'warnings.txt')
    general_input.append_permanent_logfile(warningspath, resultpath)

ingest_data()
```

config.py

```
from pymongo import MongoClient
import logging
import os


def paths():
    client = MongoClient('localhost', 27017)
    db = client.VO

    coll_wigos = db.metadata_WIGOS
    coll_cci = db.metadata_CCI_CF
    coll_iso = db.metadata_ISO_19115_3
    coll_LUTs = db.LUTs
    coll_meas = db.measurements
    coll_colloc = db.collocations
    coll_cci_colloc = db.metadata_CCI_CF_collocations
    coll_cci_colloc_IDs = db.collocations_IDs

    mainpath = 'C:/GAIA_CLIM/VO_db/data/'
    scriptpath = os.path.dirname(os.path.abspath(__file__))

    wigos_fields_filepath = scriptpath + '/input_fields/wigos_fields.txt'
    iso_fields_filepath = scriptpath + '/input_fields/iso-19115_fields.txt'
    netcdf_global_attr_fields_filepath = scriptpath + '/input_fields/cci-cf_global_attr_fields.txt'
    netcdf_variable_attr_fields_filepath = scriptpath + '/input_fields/cci-cf_variable_attr_fields.txt'
    variable_names_units_filepath = scriptpath + '/input_fields/variable_names_units.txt'

    to_be_ingested_dir = mainpath + 'to_be_ingested'
    ingested_dir = mainpath + 'ingested/'
    rejected_dir = ingested_dir + 'rejected/'
    output_dir = scriptpath + '/output/'

    return db, coll_wigos, coll_iso, coll_cci, coll_LUTs, coll_meas, coll_colloc, coll_cci_colloc, coll_cci_colloc_IDs, mainpath,
output_dir, wigos_fields_filepath, iso_fields_filepath, netcdf_global_attr_fields_filepath, netcdf_variable_attr_fields_filepath,
variable_names_units_filepath, to_be_ingested_dir, ingested_dir, rejected_dir



def logfile_setup(output_dir, mode, level=logging.INFO):
    formatter = logging.Formatter('%(levelname)s:%(message)s')

    resultpath = output_dir + 'result.txt'
    handler1 = logging.FileHandler(resultpath)
    fileHandler1 = logging.FileHandler(resultpath, mode=mode)
    handler1.setFormatter(formatter)
    result = logging.getLogger('logfile_result')
    result.setLevel(level)
    result.addHandler(handler1)

    warningspath = output_dir + 'warnings.txt'
    handler2 = logging.FileHandler(warningspath)
    fileHandler2 = logging.FileHandler(warningspath, mode=mode)
    handler2.setFormatter(formatter)
    warnings = logging.getLogger('logfile_warnings')
    warnings.setLevel(level)
    warnings.addHandler(handler2)

    return result, resultpath, warnings, warningspath
```

general_input.py

```
import os, errno
import config
import logging
import time
from pathlib import Path
import metadata_conversion


db, coll_wigos, coll_iso, coll_cci, coll_LUTs, coll_meas, coll_colloc, coll_cci_colloc, coll_cci_colloc_IDs, mainpath, output_dir,
wigos_fields_filepath, iso_fields_filepath, netcdf_global_attr_fields_filepath, \
   netcdf_variable_attr_fields_filepath, variable_names_units_filepath, to_be_ingested_dir, ingested_dir, rejected_dir =
config.paths()
result = logging.getLogger('logfile_result')
warnings = logging.getLogger('logfile_warnings')


# reads data from *.txt file which contains field names, xpaths and fields IDs for ISO-19115 and WIGOS metadata ingestion
def xml_fields(xpath_filepath):

   field_names = []
   xpaths = []
   field_IDs = []
   f = open(xpath_filepath, 'r')
   for row in f:
      row = row.strip('\n')
      row = row.split(', ')
      field_names.append(str(row[0]))
      xpaths.append(str(row[1]))
      field_IDs.append(str(row[2]))
   f.close()
   return field_names, xpaths, field_IDs


# reads data from *.txt file which contains unified variable names and their possible names in nc-files
def variable_names(filepath):

   variable_names = []
   units = []
   units_100 = []
   f = open(filepath, 'r')
   for row in f:
      one_variable = []
      one_unit = []
      one_unit_100 = []
      row = row.strip('\n')
      row = row.split(';')
      counter = 0
      for elem in row:
         elem = elem.strip("[]")
         el = elem.split(", ")
         for e in el:
            if counter == 0:
               one_variable.append(str(e))
            if counter == 1:
               one_unit.append(str(e))
            if counter == 2:
               one_unit_100.append(str(e))
         counter += 1
      variable_names.append(one_variable)
      units.append(one_unit)
      units_100.append(one_unit_100)
   f.close()
   return variable_names, units, units_100


# reads data from *.txt file which contains field names and fields IDs for CCI-CF metadata ingestion
def netcdf_fields(filepath):
```

```
    field_names = []
    field_IDs = []
    f = open(filepath, 'r')
    for row in f:
        row = row.strip('\n')
        row = row.split(', ')
        field_names.append(str(row[0]))
        r = []
        for n in range(1, len(row)):
            r.append(str(row[n]))
        field_IDs.append(r)
    f.close()
    return field_names, field_IDs


# check which ISO, WIGOS and CCI-CF files are already ingested
def files_already_in(ingested_dir):

    files_ingested = []
    for subdir, dirs, files in os.walk(ingested_dir):
        for f in files:
            files_ingested.append(f)

    return files_ingested


# detects the format of the original file
def detect_format(origFile, iso_field_names, iso_xpaths, iso_field_IDs, wigos_field_names, wigos_xpaths, wigos_field_IDs,
netcdf_global_attr_field_names, netcdf_global_attr_field_IDs, cloudfree_percent, allow_ingestion, subdir_base):

    f = open(origFile, 'r')
    dataformat = 'Unknown'
    if origFile[-3:] == '.nc':
        xml_root = ''
        logFile = mainpath + 'logfile_cci-cf_to_db.txt'
        field_names = netcdf_global_attr_field_names
        xpaths = ''
        field_IDs = netcdf_global_attr_field_IDs
        coll = coll_cci
        dataformat = 'CCI-CF'
        field_values_found, field_names_found, field_IDs_found, stn_name, start_date, start_time,
measurement_observing_method, ind_by_eff_dist, cloudfree_percent, allow_ingestion =
metadata_conversion.find_fields_cci(dataformat, origFile, logFile, field_names, [], field_IDs, 'single_measurement',
cloudfree_percent, allow_ingestion, subdir_base)
        if 'collocation_software' in field_names_found:
            coll_measurement = coll_colloc
        else:
            coll_measurement = coll_meas


    if origFile[-3:] == '.h5':
        dataformat = 'HDF'
        xml_root = ''
        logFile = mainpath + 'logfile_hdf_to_db.txt'
        field_names = netcdf_global_attr_field_names
        xpaths = ''
        field_IDs = netcdf_global_attr_field_IDs
        coll_measurement = coll_meas
        if 'LUT_TOC_' in origFile:
            coll = coll_LUTs
        else:
            coll = coll_cci

    if origFile[-4:] == '.xml':
        for row in f:
            if 'gaiaClimMetadataType' in row:
                dataformat = 'WIGOS'
                xml_root = '/gaiaClimMetadataType/'
                logFile = mainpath + 'logfile_wigos_to_db.txt'
                field_names = wigos_field_names
                xpaths = wigos_xpaths
```

```
                field_IDs = wigos_field_IDs
                coll = coll_wigos
            if 'ISO 19115-3' in row:
                dataformat = 'ISO 19115-3'
                xml_root = '/mdb:MD_Metadata/'
                logFile = mainpath + 'logfile_iso_to_db.txt'
                field_names = iso_field_names
                xpaths = iso_xpaths
                field_IDs = iso_field_IDs
                coll = coll_iso
        coll_measurement = coll_meas
    if dataformat == 'Unknown':
        result.error(' File ' + subdir_base + '/' + origFile + ' - unknown format!')
        warnings.error(' File ' + subdir_base + '/' + origFile + ' - unknown format!')
        f.close()
        return dataformat,'','','','','',''
    f.close()
    return dataformat, coll, coll_measurement, xml_root, logFile, field_names, xpaths, field_IDs



# inserts metadata in the unified format into the database
def insert_data_to_mongo(dic, coll, f):

    mongo_ID = coll.insert(dic)
    counted = coll.count()
    return mongo_ID


## create a directory if it does not exsist
def make_dir(directory):

    try:
        os.makedirs(directory)
    except OSError as e:
        if e.errno != errno.EEXIST:
            raise


## if a directory is empty, delete this
def del_empty_dir(directory):

    if not os.listdir(directory):
        os.rmdir(directory)



def append_permanent_logfile(warningspath, resultpath):

    permanent_output_filepath = output_dir + 'permanent_output.txt'
    my_file = Path(permanent_output_filepath)
    if my_file.is_file():
        w_mode = 'a'
    else:
        w_mode = 'w'
    filenames = [warningspath, resultpath]
    with open(permanent_output_filepath, w_mode) as outfile:
        outfile.write('------------------------------------------------------------------------' + '\n')
        for fname in filenames:
            with open(fname) as infile:
                for line in infile:
                    outfile.write(line)
        outfile.write('------------------------------------------------------------------------')
```

metadata_conversion.py

```
import pymongo
import json
import xmltodict
import collections
import netCDF4
import time
from text_unidecode import unidecode
from datetime import datetime
from bson.objectid import ObjectId
from bson import json_util
import config
import logging
import numpy as np
import math

db, coll_wigos, coll_iso, coll_cci, coll_LUTs, coll_meas, coll_colloc, coll_cci_colloc, coll_cci_colloc_IDs, mainpath, output_dir,
wigos_fields_filepath, iso_fields_filepath, netcdf_global_attr_fields_filepath, \
   netcdf_variable_attr_fields_filepath, variable_names_units_filepath, to_be_ingested_dir, ingested_dir, rejected_dir =
config.paths()
result = logging.getLogger('logfile_result')
warnings = logging.getLogger('logfile_warnings')


def split_uppercase(s):
   r = []
   l = False
   count = 0
   for c in s:
      # l being: last character was not uppercase
      if l and c.isupper():
         r.append(' ')
      l = not c.isupper()
      if count == 0:
         r.append(c.upper())
      else:
         r.append(c)
      count += 1
   return ''.join(r)


#Given two dicts, merge them into a new dict as a shallow copy
def merge_two_dicts(x, y):

   z = x.copy()
   z.update(y)
   return z



# convert the field value to float
def if_number(value, formatting):

   try:
      if formatting == 'float':
         value = float(value)
      else:
         value = int(value)
   except (ValueError, TypeError):
      value = value
   return value


# search the fields determined by cci-cf_fields.txt in CCI-CF netCDF4 files
def find_fields_cci(dataformat, origFile, logFile, field_names, variable_names, field_IDs, c, cloudfree_percent, allow_ingestion,
subdir_base):


   field_values_found = []
```

```
    field_names_found = []
    field_IDs_found = []
    stn_name = ''
    measurement_observing_method = ''
    distance_diff = ''
    ind_by_eff_dist = 0
    time_diff = ''
    nb_of_param = 1
    date_format = "%Y-%m-%d"
    time_format = "%H:%M:%S"

    if c == 'reference':
      c2 = 'ref_'
      except_fields = 'monitored_'
    if c == 'monitored':
      except_fields = 'reference_'
      c2 = 'mon_'
    if c == 'single_measurement':
      except_fields = 'nothing_to_except'
      c2 = 'single_'

    if dataformat == 'CCI-CF':
      j = netCDF4.Dataset(origFile, mode='r')

      keys = list(j.variables.keys())
      attrs = list(j.ncattrs())

      attrs_val = []
      for a in j.ncattrs():
        val = getattr(j, a)
##        print (val)
        attrs_val.append(val)

      attrs_found = []
      for elem in range(len(field_IDs)):
        for nc_path in field_IDs[elem]:
##          print (nc_path)
          if nc_path in attrs and except_fields not in field_names[elem]:
            attrs_found.append(nc_path)
            ind = attrs.index(nc_path)
            field_IDs_found.append(attrs[ind])
            value = attrs_val[ind]
            if field_names[elem] == 'programme_network_affiliation_name' and ' ' in value:
              values = value.split(' ')
              for v in values:
                if v.upper() == "GRUAN" or v.upper() == "WOUDC":
                  value = v.upper()
            if field_names[elem] == 'monitored_product.name':
              if 'TemperatureMonitored' in value:
                value = value.replace('TemperatureMonitored', 'Temperature')
            if field_names[elem] == 'monitored_satellite':
              if value == 'N1919':
                value = 'N19'

            if field_names[elem] == 'station_platform_model' or field_names[elem] == 'measurement_observing_method':
              if ' ' in value:
                values = value.split(' ')
                for v in values:
                  v = v.lower()
                  if v == 'ozonsesonde' or v == 'ozonesonde':
                    value = 'ozonesonde'
                  if v == 'gruan':
                    value = 'radiosonde'
              if field_names[elem] == 'station_platform_model':
                if value == 'jpss':
                  value = 'JPSS ATMS'

            if field_names[elem] in values_split:
              if ' ' in str(value):
                values = value.split()
                value = values[0]
```

```
            if 'station_platform_name' in field_names[elem]:
               value = split_uppercase(value)
               value = ' '.join(value.split())
               if ',' in value:
                  values = value.split(',')
                  value = values[0]
               stn_name = value

            if field_names[elem] != 'product.version' and field_names[elem] != 'general.type.version':
               if field_names[elem][-3:] == ".id" or field_names[elem][-3:] == "_id":
                  value = if_number(value, 'int')
               else:
                  value = if_number(value, 'float')

            if field_names[elem] == 'measurement_observing_method' and nc_path == 'instrument':
               measurement_observing_method = value
               if 'LIN-' in origFile:
                  stn_name = 'Lindenberg'

            if field_names[elem] == 'Channel' or field_names[elem] == 'Wavenumber' or field_names[elem] ==
'reference_wavenumbers':
               value = value.tolist()

            if value != ' ' and value != '':
               field_names_found.append(field_names[elem])
               field_values_found.append(value)


      if 'channels' not in field_names_found:
         for dim in j.dimensions:
            if dim == 'ch_ref':
               channel_list = list(range(1, 20))
               field_names_found.append('Channel')
               field_values_found.append(channel_list)



      if c in ['monitored','reference']:
         keys = list(j.variables.keys())
         if 'mon_aod550' in keys:
            vals = np.asarray(j.variables['time_diff'])
            time_diff_list = vals.tolist()
            vals = np.asarray(j.variables['distance'])
            dist_list = vals.tolist()
            field_names_found.append('collocation_space_diff_km')
            field_values_found.append(dist_list[0])
            field_names_found.append('collocation_time_diff_sec')
            field_values_found.append(time_diff_list[0])

            for variable_name in variable_names:
               for key in keys:
                  if key.lower().replace(c2,'') in variable_name:
                     vals = np.asarray(j.variables[key.lower()])
                     if variable_name[0] == 'longitude':
                        field_names_found.append('longitude')
                        field_values_found.append(float(vals[0]))
                     if variable_name[0] == 'latitude':
                        field_names_found.append('latitude')
                        field_values_found.append(float(vals[0]))


         else:
            try:
               vals = np.asarray(j.variables['eff_distance'])
               eff_dist_list = vals.tolist()
               for i in range(len(eff_dist_list)):
                  if math.isnan(eff_dist_list[i]) or eff_dist_list[i] > 1000:
                     allow_ingestion = False
```

```
                warnings.warning(' Data from ' + subdir_base + '/' + origFile.split('\\')[-1] + ' was not ingested - meaningful values for "Effective distance" missing')
                break

            vals = np.asarray(j.variables['time_diff'])
            time_diff_list = vals.tolist()
            vals = np.asarray(j.variables['distance'])
            dist_list = vals.tolist()
            vals = np.asarray(j.variables['mon_eff_lat'])
            mon_eff_lat_list = vals.tolist()
            vals = np.asarray(j.variables['mon_eff_lon'])
            mon_eff_lon_list = vals.tolist()
            vals = np.asarray(j.variables['mon_bt'])
            mon_bt_list = vals.tolist()
            ind_by_eff_dist_pre_list = []
            for n in range(len(eff_dist_list)):
              if all(e >= 180 for e in mon_bt_list[n]):
                ind_by_eff_dist_pre_list.append(n)

            if ind_by_eff_dist_pre_list == []:
              allow_ingestion = False
              ind_by_eff_dist = 0
              warnings.warning(' Data from ' + subdir_base + '/' + origFile.split('\\')[-1] + ' was not ingested - meaningful values for "Brightness temperature" missing')

            ind_by_eff_dist_pre = ind_by_eff_dist_pre_list[0]
            ind_by_eff_dist = ind_by_eff_dist_pre_list[0]
            vals = np.asarray(j.variables['mon_cloud_flag']).tolist()

            if vals[ind_by_eff_dist_pre][7] == 1:      ## search for measurements made by monitored instrument with no low clouds but more far away (effective distance increasing)
              for i in ind_by_eff_dist_pre_list[1:]:
                if vals[i][7] == 0:
                  ind_by_eff_dist = i
                  break

            if ind_by_eff_dist == ind_by_eff_dist_pre:     ## in case of low clouds were present in all collocations, search the nearest pixel without high clouds
              for i in ind_by_eff_dist_pre_list:
                if vals[i][11] == 0 or vals[i][12] == 0:
                  ind_by_eff_dist = i
                  break

            field_names_found.append('collocation_space_diff_km')
            field_values_found.append(eff_dist_list[ind_by_eff_dist])
            field_names_found.append('collocation_time_diff_sec')
            field_values_found.append(time_diff_list[ind_by_eff_dist])
            field_names_found.append('collocation_space_diff_km_noneff')
            field_values_found.append(dist_list[ind_by_eff_dist])

          except:
            pass


          try:

            for variable_name in variable_names:
              for key in keys:
                if key.lower().replace(c2,'') in variable_name:
                  vals = np.asarray(j.variables[key.lower()])
                  if variable_name[0] == 'effective longitude':
                    field_names_found.append('longitude')
                    field_values_found.append(float(vals[ind_by_eff_dist]))
                  if variable_name[0] == 'effective latitude':
                    field_names_found.append('latitude')
                    if float(vals[ind_by_eff_dist]) > 90:
                      field_values_found.append(90)
                    else:
                      field_values_found.append(float(vals[ind_by_eff_dist]))
                  if variable_name[0] == 'longitude':
                    field_names_found.append('longitude_noneff')
```

```
                        if c2 == 'ref_':
                            field_values_found.append(float(vals))
                        else:
                            field_values_found.append(float(vals[ind_by_eff_dist]))
                    if variable_name[0] == 'latitude':
                        field_names_found.append('latitude_noneff')
                        if c2 == 'ref_':
                            field_values_found.append(float(vals))
                        else:
                            field_values_found.append(float(vals[ind_by_eff_dist]))


                else:
                    if key.lower().replace(c2,'') == 'cloud_flag' and 'cloud_flag' not in field_names_found:
                        vals = np.asarray(j.variables[key.lower()])
                        field_names_found.append('cloud_flag')
                        field_values_found.append(vals[ind_by_eff_dist].tolist())
                        field_names_found.append('cloud_flag_comment')
                        field_values_found.append(str(getattr(j.variables[key], 'units')))

            except:
                pass

        j.close()




    # writes a logfile
    if c == 'single_measurement':
        for a in attrs:
            if a not in attrs_found:
                warnings.warning(' New field ' + a + ' was found in file ' + subdir_base + '/' + origFile.split('\\')[-1])
        ind_by_eff_dist = ''
    return field_values_found, field_names_found, field_IDs_found, stn_name, start_date, start_time,
measurement_observing_method, ind_by_eff_dist, cloudfree_percent, allow_ingestion




# searches the fields determined by iso-19115_fields.txt and wigos_fields.txt in xml files
def find_fields_iso_wigos(dataformat, root, origFile, logFile, field_names, xpaths, field_IDs, variable_names):

    field_values_found = []
    field_names_found = []
    xpaths_found = []
    field_IDs_found = []
    nb_of_param = 1
    variable_attr = ['spatial_extent_min_value', 'spatial_extent_max_value', 'measurand_domain',
'measurand_mode_of_observation', 'measurand_subdomain', 'measurand_unit_name', 'measurand_unit_abbreviation',
'temporal_extent', 'start_date', 'start_time', 'stop_date', 'stop_time', 'representativeness_of_observation']

    if dataformat == 'WIGOS':
        for i in range(len(xpaths)):
            a = root.findall(xpaths[i])
            for field in range(len(a)):

                try:
                    value = a[field].find(field_IDs[i]).text
                except AttributeError:
                    if field_names[i] in variable_attr:
                        value = 'missing'
                        if field == 0:
                            var_values = [value]
                        if field > 0:
                            var_values.append(value)
                    continue
```

```
            # if the xpath is unique
            if len(a) == 1 and value != None:
               field_values_found.append(value)
               field_names_found.append(field_names[i])
               xpaths_found.append(xpaths[i])
               field_IDs_found.append(field_IDs[i])

            # if there are several fields with the same xpath
            if len(a) > 1 and value != None:
               if field == 0:
                  var_values = [value]
               if field > 0:
                  var_values.append(value)

               if field == len(a)-1:
                  field_values_found.append(var_values)
                  field_names_found.append(field_names[i])
                  xpaths_found.append(xpaths[i])
                  field_IDs_found.append(field_IDs[i])
                  nb_of_param = len(var_values)
   ##               break

##    print (field_values_found)
   if dataformat == 'ISO 19115-3':
      for i in range(len(xpaths)):
         a = root.findall(xpaths[i], namespaces_ISO_19115)
         for field in range(len(a)):

##             if True:
            try:
               if '@' not in field_IDs[i] and field_IDs[i] != 'codeListValue' and field_IDs[i] != 'id':
                  value = a[field].find(field_IDs[i], namespaces_ISO_19115).text

               elif '@' in field_IDs[i]:
                  field_IDs[i] = field_IDs[i][1:]
                  value = a[field].get(field_IDs[i], namespaces_ISO_19115)

               elif field_IDs[i] == 'codeListValue' or field_IDs[i] == 'id':
                  value = a[field].get(field_IDs[i], namespaces_ISO_19115)


            except AttributeError:
               continue

            if field_names[i] == 'programme_network_affiliation_name':
               value = value.split(' - ')
               value = value[0].upper()

            if field_names[i] == 'station_platform_name':
               value = value.split(', ')
               value = value[0].split(': ')
               value = value[1].replace('-', ' ')
               value = unidecode(value).title()
               if value == 'Lindeberg':
                  value = 'Lindenberg'

            if field_names[i] == 'station_altitude_asl':
               if ' Alt:' not in value:
                  continue
               else:
                  values = value.split(',')
                  for val in values:
                     v = val.split(':')
                     if v[0] == ' Alt':
                        value = v[1].split()
                        value = value[0]


            if field_names[i] == 'measurand_variable':
               value = value.title()
```

```
            if field_names[i] == 'application_area' or field_names[i] == 'use_constraints':
               value = split_uppercase(value)

            try:
               value = value.replace('\n','')     ## remove newlines
               value = ' '.join(value.split())    ## remove duplicated spaces
            except AttributeError:
               value = value

            value = if_number(value, 'float')

            # if the xpath is unique
            if len(a) == 1 and value != None:
               field_values_found.append(value)
               field_names_found.append(field_names[i])
               xpaths_found.append(xpaths[i])
               field_IDs_found.append(field_IDs[i])

            # if there are several fields with the same xpath
            if len(a) > 1 and value != None:
               if i == 10 and field == 0 or i == 11 and field == 1 or i == 22 and field == 0 or i == 30 and field == 0 or i == 31 and field == 0
or i == 32 and field == 1 or i == 33 and field == 2 or i == 34 and field == 3 or i == 35 and field == 4 or i == 36 and field == 5:
                  field_values_found.append(value)
                  field_names_found.append(field_names[i])
                  xpaths_found.append(xpaths[i])
                  field_IDs_found.append(field_IDs[i])
                  break

   return field_values_found, field_names_found, xpaths_found, field_IDs_found, nb_of_param




# finds the station name from WIGOS metadata
def find_stn_name_from_WIGOS(xml_file, field_values_found, field_names_found, xpaths_found, xml_attribs=True):

   with open(xml_file, 'rb') as f:
      for i in range(len(xpaths_found)):
         if field_names_found[i] == 'station_platform_name':
            stn_name = field_values_found[i]
            if ',' in stn_name:
               stn_name = stn_name.split(',')
               stn_name = stn_name[0]

   return (stn_name)



# converts the metadata information found in original files in ISO-19115 and WIGOS format into a dictionary
def iso_wigos_to_dict(dataformat, coll, xml_file, xml_text, field_values_found, field_names_found, xpaths_found,
field_IDs_found, nb, xml_root, orig_filename, subdir_base, xml_attribs=True):

   fields_in_degrees = ['longitude', 'latitude']
   fields_in_meters = ['station_altitude_asl', 'extent_minimum_value', 'extent_maximum_value']
##    print (field_names_found)
##    print (field_values_found)
   variable_attr = ['spatial_extent_min_value', 'spatial_extent_max_value', 'measurand_domain',
'measurand_mode_of_observation', 'measurand_subdomain', 'measurand_unit_name', 'measurand_unit_abbreviation',
'temporal_extent', 'start_date', 'start_time', 'stop_date', 'stop_time', 'representativeness_of_observation']


   with open(xml_file, 'rb') as f:
      d0 = collections.OrderedDict()

      d0['metadata_origin_format'] = dataformat
      if dataformat == 'WIGOS':
         d0['observation_data_present'] = 'True'
         d0['metadata_CCI_CF_ID'] = ''
         coll.create_index('metadata_CCI_CF_ID')
```

```
        d0['measurand_variable_list'] = ''
        d0['measurand_variables'] = collections.OrderedDict()
    if dataformat == 'ISO 19115-3':
        try:
            for e in field_names_found:
                if e == 'station_platform_name':
                    ind = field_names_found.index(e)
                    stn_name = field_values_found[ind]
                    itm_measurements = db.measurements.find_one({"station_platform_name":stn_name})
                    ISO_19115_3_ID = itm_measurements.get('_id')
                    d0['observation_data_present'] = 'True'
        except AttributeError:
            d0['observation_data_present'] = 'False'


    for i in range(len(xpaths_found)):
        if dataformat == 'WIGOS':
            if field_names_found[i] == 'measurand_variable':
                variables = field_values_found[i]
                for n in range(nb):
                    d0['measurand_variables'][variables[n]] = collections.OrderedDict()
                continue

            if field_names_found[i] in variable_attr:
                for n in range(nb):
                    d0['measurand_variables'][variables[n]][field_names_found[i]] = field_values_found[i][n]
                continue

        if isinstance(field_values_found[i], (str, float, int)):
            d0[field_names_found[i]] = field_values_found[i]
            value = field_values_found[i]

        if field_names_found[i] in fields_in_degrees:
            d0[field_names_found[i]] = value
            d0[field_names_found[i] + "_unit_name"] = 'degree'
        if field_names_found[i] in fields_in_meters:
            d0[field_names_found[i]] = value
            d0[field_names_found[i] + "_unit_name"] = 'metre'
        if field_names_found[i] not in fields_in_degrees and field_names_found[i] not in fields_in_meters:
            d0[field_names_found[i]] = value

        if field_names_found[i] == 'longitude':
            longitude = field_values_found[i]

        if field_names_found[i] == 'latitude':
            latitude = field_values_found[i]
            d0['loc'] = {'type': 'Point', 'coordinates': [longitude, latitude]}

        if field_names_found[i] == 'station_platform_name':
            stn_name = field_values_found[i]

        if field_names_found[i] == 'programme_network_affiliation_name':
            network_name = field_values_found[i].upper()

    d0['original_filename'] = subdir_base + '/' + orig_filename
    d1 = xmltodict.parse(f, xml_attribs=xml_attribs)    # original xml file transformed to dictionary
    d2 = {}

    d2['original_xml'] = xml_text
    dic = merge_two_dicts(d0, d2)
    dic_to_json = json.dumps(dic, indent=4, default=json_util.default)
  coll.create_index([('loc', pymongo.GEOSPHERE)])

  return (dic, stn_name, network_name)



# converts the metadata information found in original files in CCI-CF format into a dictionary
def cci_to_dict(origFile, dataformat, coll, field_values_found, field_names_found, field_IDs_found, stn_name, orig_filename,
subdir_base, c):
```

```
    fields_in_degrees = ['longitude', 'latitude']

    d0 = collections.OrderedDict()
    for i in range(len(field_names_found)):
      if i == 0:
        if c != 'single_measurement':
          d0['monitored_or_reference'] = c
          d0['collocation_IDs'] = ''
        d0['metadata_origin_format'] = dataformat
        try:
          if c == 'single_measurement':
            d0['station_platform_name'] = stn_name
          d0['measurand_variable_list'] = ''
        except (AttributeError):
          d0[field_names_found[i]] = field_values_found[i]

      field_names_found[i] = field_names_found[i].replace('monitored_','')
      field_names_found[i] = field_names_found[i].replace('reference_','')
      field_names_found_splitted = field_names_found[i].split('.')


      if field_names_found_splitted == [field_names_found[i]]:
        d0[field_names_found[i]] = field_values_found[i]
      if len(field_names_found_splitted) == 2:
        try:
          d0[field_names_found_splitted[0]][field_names_found_splitted[1]] = field_values_found[i]
        except KeyError:
          d0[field_names_found_splitted[0]] = collections.OrderedDict()
          d0[field_names_found_splitted[0]][field_names_found_splitted[1]] = field_values_found[i]
      if len(field_names_found_splitted) == 3:
        try:
          d0[field_names_found_splitted[0]][field_names_found_splitted[1]][field_names_found_splitted[2]] =
field_values_found[i]
        except KeyError:
          d0[field_names_found_splitted[0]] = collections.OrderedDict()
          d0[field_names_found_splitted[0]][field_names_found_splitted[1]] = collections.OrderedDict()
          d0[field_names_found_splitted[0]][field_names_found_splitted[1]][field_names_found_splitted[2]] =
field_values_found[i]

      if field_names_found[i] in fields_in_degrees:
        d0[field_names_found[i] + "_unit_name"] = 'degree'

      if field_names_found[i] == 'latitude':
        latitude = field_values_found[i]
        if 'longitude' in field_names_found:
          lon_ind = field_names_found.index('longitude')
          d0['loc'] = {'type': 'Point', 'coordinates': [field_values_found[lon_ind], latitude]}

      if field_names_found[i] == 'longitude':
        longitude = field_values_found[i]

      if field_names_found[i] == 'measurement_observing_method':
        if field_values_found[i] == 'atms' or field_values_found[i] == 'iasi':
          if 'GRUAN' in origFile:
            d0['measurement_observing_method'] = "processor"
            d0['programme_network_affiliation_name'] = "GRUAN"
            d0['comment'] = 'Sounding processed in the GRUAN processor resulting in a simulated
{0}'.format(field_values_found[i].upper()) + ' measurement.'
            d0['monitored_or_reference'] = 'reference'
          else:
            d0['measurement_observing_method'] = "NWP model"
            if field_values_found[i] == 'atms':
              d0['programme_network_affiliation_name'] = "UK MetOffice"
            if field_values_found[i] == 'iasi':
              d0['programme_network_affiliation_name'] = "ECMWF"
            d0['comment'] = 'GRUAN processor output for the NWP fields matching the sounding and resulting in a simulated
{0}'.format(field_values_found[i].upper()) + ' measurement.'
            d0['monitored_or_reference'] = 'monitored'

    if c != 'monitored':
```

```
        d0['station_platform_name'] = stn_name

    # First, if latitude and/or longitude fields were not found in nc-file, take the coordinates from ISO metadata.
    # If its not possible, take the first values from lat-lon profiles from nc-file
    if 'longitude' not in field_names_found or 'latitude' not in field_names_found:
        latitude = ''
        longitude = ''
        for itm_ISO_19115_3 in db.metadata_ISO_19115_3.find({"station_platform_name":stn_name}).limit(1):
            latitude = itm_ISO_19115_3.get('latitude')
            longitude = itm_ISO_19115_3.get('longitude')
            d0['latitude'] = latitude
            d0['longitude'] = longitude
            d0['loc'] = {'type': 'Point', 'coordinates': [longitude, latitude]}

        if latitude == '' or longitude == '':
            j = netCDF4.Dataset(origFile, 'r')
            keys = list(j.variables.keys())
            for key in keys:
                if key == 'lat':
                    latitude = round(float(j.variables[key][0]), 3)
                    d0['latitude'] = latitude
                if key == 'lon':
                    longitude = round(float(j.variables[key][0]), 3)
                    d0['longitude'] = longitude
                    d0['loc'] = {'type': 'Point', 'coordinates': [longitude, latitude]}

            j.close()
    d0['original_filename'] = subdir_base + '/' + orig_filename
    dic = d0
    dic_to_json = json.dumps(dic, indent=4)
    coll.create_index([('loc', pymongo.GEOSPHERE)])

    return dic


## for structuring documents in the collocations_IDs collection
def dic_cci_short(dic_cci, dic_cci_short):

    station_platform_name = ""
    include_fields = ['instrument', 'product', 'NWP', 'station_platform_name', 'start_date', 'start_time', 'cloud_flag',
'collocation_time_diff_sec', 'collocation_space_diff_km']
    monitored_or_reference = dic_cci['monitored_or_reference']
    meas_types = ["satellite_products", "satellite_products", "satellite_products", "satellite_products", "satellite_products",
"satellite_products", "simulations", "simulations", "references", "references", "satellite_products", "references"]
    drop_down_menu = ["EUMETSAT IASI Temperature Profile", "EUMETSAT IASI Humidity Profile", "AMSU-A Brightness
Temperature", "MHS Brightness Temperature", "HIRS Brightness Temperature", "IASI Radiance Spectrum", "UK MetOffice",
"ECMWF", "GRUAN Radiosonde/RTTOV-UKMO", "GRUAN Radiosonde/RTTOV-ECMWF", "AATSR Aerosol Optical Depth",
"AERONET Sunphotometer"]
    instrument = ["IASI", "IASI", "AMSU-A", "MHS", "HIRS", "IASI", "NWP radiosonde simulation", "NWP radiosonde simulation",
"Radiosonde converted", "Radiosonde converted", "AATSR", "AERONET Sunphotometer"]
    product = ["Temperature Profile", "Humidity Profile", "Brightness Temperature", "Brightness Temperature", "Brightness
Temperature", "Radiance Spectrum", "from NWP", "from NWP", "RTTOV", "RTTOV", "Aerosol Optical Depth", "Aerosol Optical
Depth"]
    NWP_model = ["-", "-", "-", "-", "-", "-", "MetOffice", "ECMWF", "MetOffice", "ECMWF", "-", "-"]

    for i in range(len(instrument)):
        if instrument[i] in dic_cci['instrument']:
            if product[i] in dic_cci['product']['name']:
                try:
                    if NWP_model[i] in dic_cci['NWP']['model']:
                        meas_type = meas_types[i]
                        drop_down_choice = drop_down_menu[i]
                except (KeyError):
                    meas_type = meas_types[i]
                    drop_down_choice = drop_down_menu[i]


    for i in include_fields:
        if i in dic_cci:
            if i == 'instrument':
```

```
            dic_cci_short[meas_type] = collections.OrderedDict()
            dic_cci_short[meas_type][drop_down_choice] = collections.OrderedDict()
            dic_cci_short[meas_type][drop_down_choice][i] = dic_cci[i]
        elif i in ['collocation_time_diff_sec', 'collocation_space_diff_km']:
            if i == 'collocation_time_diff_sec':
                if 'satellite' not in meas_type:
                    dic_cci_short[meas_type][drop_down_choice][i] = dic_cci[i]
            if i == 'collocation_space_diff_km':
                if 'satellite' not in meas_type:
                    dic_cci_short[meas_type][drop_down_choice][i] = dic_cci[i]
        elif i == 'product':
            dic_cci_short[meas_type][drop_down_choice][i] = dic_cci['product']['name']
        elif i == 'NWP':
            dic_cci_short[meas_type][drop_down_choice]['NWP_model'] = dic_cci['NWP']['model']
        else:
            if i == 'start_date':
                start_date = dic_cci[i]
            if i == 'start_time':
                start_time = dic_cci[i]
            if i == 'station_platform_name':
                station_platform_name = dic_cci[i]
            dic_cci_short[meas_type][drop_down_choice][i] = dic_cci[i]

    return meas_type, drop_down_choice, dic_cci_short, start_date, start_time, meas_types, drop_down_menu,
station_platform_name
```

measurement_conversion.py

```
import netCDF4
import collections
import json
import math
import os
import time
import numpy as np
from bson.objectid import ObjectId
from bson import json_util
import config
import logging

db, coll_wigos, coll_iso, coll_cci, coll_LUTs, coll_meas, coll_colloc, coll_cci_colloc, coll_cci_colloc_IDs, mainpath, output_dir,
wigos_fields_filepath, iso_fields_filepath, netcdf_global_attr_fields_filepath, \
   netcdf_variable_attr_fields_filepath, variable_names_units_filepath, to_be_ingested_dir, ingested_dir, rejected_dir =
config.paths()
result = logging.getLogger('logfile_result')
warnings = logging.getLogger('logfile_warnings')


## Check if qcinfo and qcflags values in nc-file are equal to 0
def check_quality_info(ds, measurement_observing_method):

   keys = list(ds.variables.keys())
   qcinfo_ok = ''
   qcflags_ok = ''
   if measurement_observing_method != 'atms':
      qcinfo_ok = 'Yes'
      qcflags_ok = 'Yes'
   else:
      for key in keys:
         if key == 'qcinfo':
            vals = list(ds.variables[key])
            if vals[0] == 0:
               qcinfo_ok = 'Yes'
         if key == 'qcflags':
            vals = list(ds.variables[key])
            if vals[0] == 0:
               qcflags_ok = 'Yes'

   return keys, qcinfo_ok, qcflags_ok



def unit_field(d0, ds, key, units, units_multiply, unit, variable_index, variable_name, multiply_factor, origFile, mainpath, c,
subdir_base):
   if unit == None:
      warnings.warning(' Unit missing in file ' + subdir_base + '/' + origFile.split('\\')[-1] + ' for variable "' +  d0['measurand_variable']
+ '"')
   if unit != ' ' and unit[-1] == ' ':
      unit = unit[:-1]
   if unit not in units[variable_index] and unit not in units_multiply[variable_index]:
      if key.lower() != 'time' and key.lower() != c+'time':
##          print (key.lower())
         warnings.warning(' New unit "' + unit + '" found in file ' + subdir_base + '/' + origFile.split('\\')[-1] + ' for variable "' +
d0['measurand_variable'] + '"')
      d0['units'] = unit
   if unit not in units[variable_index] and unit in units_multiply[variable_index]:
      multiply_factor = float(units_multiply[variable_index][-1].strip('multiply='))
      d0['units'] = units[variable_index][0]
   if unit in units[variable_index]:
      d0['units'] = units[variable_index][0]

   return d0, multiply_factor
```

```
def meas_nc_to_dict(origFile, coll_meas, mainpath, variable_names, WIGOS_ID, CCI_CF_ID, stn_name, start_date, start_time,
measurement_observing_method, field_names, field_IDs, units, units_multiply, c, ind_by_eff_dist, subdir_base):

    logFile_measurements = mainpath + 'logfile_measurements_to_db.txt'
    logFile_measurements_var_attr = mainpath + 'logfile_measurements_var_attr_to_db.txt'

    ds = netCDF4.Dataset(origFile, 'r')
    f = origFile.split('\\')
    measurand_variables = []
    dics = []

    if c == 'reference':
        c = 'ref_'
        except_param = 'mon_'
    if c == 'monitored':
        c = 'mon_'
        except_param = 'ref_'
    if c == 'single_measurement':
        c = 'single_'
        except_param = 'nothing_to_except'

    y_axis_dims = ['time', 'altitude', 'geopotential altitude', 'air pressure', 'channel', 'length', 'levels', 'num_obs', 'nMonCollocations',
'nRefCollocations']
    x_axis_variables = []
    y_axis_variables = []


    keys, qcinfo_ok, qcflags_ok = check_quality_info(ds, measurement_observing_method)

    if qcinfo_ok == 'Yes' and qcflags_ok == 'Yes':
        for key in keys:

            var_shape = ds.variables[key].shape
            if var_shape == ():    ## if variable is dimensionless
                vals = [float(ds.variables[key].getValue())]
            key_found = False

            for variable_name in variable_names:
                if key.lower() in variable_name or key.lower().replace(c,'') in variable_name:
                    variable_index = variable_names.index(variable_name)
                    multiply_factor = False
                    key_found = True
                    can_be_ingested = False
                    if var_shape != ():
                        vals = np.asarray(ds.variables[key])
                        vals = vals.tolist()
                        if c != 'single_':
                            if key.lower() in ['distance','time_diff','cos_ratio'] or 'mon_' in key.lower():
                                vals = [vals[ind_by_eff_dist]]

                    d0 = collections.OrderedDict()
                    d0['measurand_variable'] = variable_name[0].capitalize()

                    if WIGOS_ID != '':
                        d0['metadata_WIGOS_ID'] = ObjectId(WIGOS_ID)
                        coll_meas.create_index('metadata_WIGOS_ID')

                    d0['metadata_CCI_CF_ID'] = ObjectId(CCI_CF_ID)
                    coll_meas.create_index('metadata_CCI_CF_ID')
                    if c == 'single_measurement':
                        d0['station_platform_name'] = stn_name
                    d0['start_date'] = start_date
                    d0['start_time'] = start_time

                    dims = []
                    for dim in ds.variables[key].dimensions:
                        dim_found = ''
                        for v in variable_names:
                            if dim.lower() in v:
                                dims.append(str(v[0].capitalize()))
                                dim_found = 'Yes'
```

```
            if dim_found == '':
                dims.append(dim)
        d0['dimensions'] = dims

        dims_lower = [x.lower() for x in dims]
        for dim in y_axis_dims:
            if dim in dims_lower and variable_name[0] not in y_axis_dims:
                d0['y_axis_variables'] = []

            if dim == variable_name[0]:
                d0['x_axis_variables'] = []


        d0['name'] = key

        for attr in ds.variables[key].ncattrs():
            attr_name = ''
            for elem in range(len(field_IDs)):
                for nc_path in field_IDs[elem]:
                    if nc_path == attr:
                        attr_name = field_names[elem]
            if attr_name != '':
                if attr_name == 'fill_value':
                    d0[attr_name] = float(getattr(ds.variables[key], attr))
                    fill_value_default.append(float(getattr(ds.variables[key], attr)))

                if attr_name == 'units':
                    unit = str(getattr(ds.variables[key], attr))
                    d0, multiply_factor = unit_field(d0, ds, key, units, units_multiply, unit, variable_index, variable_name,
multiply_factor, origFile, mainpath, c, subdir_base)

                else:
                    d0[attr_name] = str(getattr(ds.variables[key], attr))
            else:
                warnings.warning(' New variable attribute ' + attr + ' found in file ' + subdir_base + '/' + origFile.split('\\')[-1])

        if 'units' not in ds.variables[key].ncattrs():
            if measurement_observing_method in ['atms','iasi']:
                unit = rttov_keys_units.get(key)
                d0, multiply_factor = unit_field(d0, ds, key, units, units_multiply, unit, variable_index, variable_name,
multiply_factor, origFile, mainpath, c, subdir_base)


        if isinstance(vals[0], list):
            for u in vals:
                for n,i in enumerate(u):
                    if i in fill_value_default or math.isnan(float(i)) == True:
                        u[n]= -999999
                    else:
                        if variable_name[0].capitalize() == 'Air temperature':
                            if 200 < u[n] < 340:
                                can_be_ingested = True
                        else:
                            can_be_ingested = True
                            if multiply_factor is not False:
                                u[n]= u[n]*multiply_factor

        if isinstance(vals[0], (int, float, str)):
            for n,i in enumerate(vals):
                if i in fill_value_default or math.isnan(float(i)) == True:
                    vals[n]= -999999
                else:
                    if variable_name[0].capitalize() == 'Air temperature':
                        if 200 < vals[n] < 340:
                            can_be_ingested = True
                    else:
                        can_be_ingested = True
                        if multiply_factor is not False:
                            vals[n]= vals[n]*multiply_factor
```

```
                d0['values'] = vals
                dic = d0
                dic_to_json = json.dumps(dic, indent=4, default=json_util.default)

                if can_be_ingested is True:
                    if c != 'single_':
                        if except_param not in str(getattr(ds.variables[key], 'long_name')):
                            dics.append(dic)
                            measurand_variables.append(variable_name[0].capitalize())
                            for dim in y_axis_dims:
                                if dim in dims_lower:
                                    if variable_name[0].capitalize() not in x_axis_variables and variable_name[0] not in y_axis_dims:
                                        x_axis_variables.append(variable_name[0].capitalize())

                                if dim == variable_name[0]:
                                    if variable_name[0].capitalize() not in y_axis_variables:
                                        y_axis_variables.append(variable_name[0].capitalize())
                    else:
                        dics.append(dic)
                        measurand_variables.append(variable_name[0].capitalize())

                        for dim in y_axis_dims:
                            if dim in dims_lower:
                                if variable_name[0].capitalize() not in x_axis_variables and variable_name[0] not in y_axis_dims:
                                    x_axis_variables.append(variable_name[0].capitalize())


                            if dim == variable_name[0]:
                                if variable_name[0].capitalize() not in y_axis_variables:
                                    y_axis_variables.append(variable_name[0].capitalize())



                if can_be_ingested is False and d0['measurand_variable'] != 'Solar zenith angle':
                    warnings.warning(' Parameter "' + d0['measurand_variable'] + '" found in ' + subdir_base + '/' + origFile.split('\\')[-1]
+' was not ingested due to quality issues')


            if key_found is False and except_param not in key and key not in ['eff_distance', 'mon_cloud_flag']:
                warnings.warning(' New parameter ' + key + ' found in ' + subdir_base + '/' + origFile.split('\\')[-1])

        for dictionary1 in dics:
            x_axis_variables_copy = x_axis_variables[:]
            y_axis_variables_copy = y_axis_variables[:]

            for key, value in dictionary1.items():
                if key == 'measurand_variable':
                    if value in x_axis_variables_copy:
                        if 'Channel' in dictionary1['dimensions']:
                            dictionary1['y_axis_variables'] = ['Channel']
                        else:
                            dictionary1['y_axis_variables'] = y_axis_variables_copy
                    if value in y_axis_variables_copy:
                        for dic2 in dics:
                            for key2, value2 in dic2.items():
                                if 'Channel' in dic2['dimensions'] and dic2['measurand_variable'] in x_axis_variables_copy:
                                    x_axis_variables_copy.remove(dic2['measurand_variable'])
                        dictionary1['x_axis_variables'] = x_axis_variables_copy

    else:
        db.metadata_CCI_CF.remove({'_id': ObjectId(CCI_CF_ID)})
        result.warning(' Data derived from ' + subdir_base + '/' + origFile.split('\\')[-1] + ' was not ingested due to quality issues')
        warnings.warning(' Data derived from ' + subdir_base + '/' + origFile.split('\\')[-1] + ' was not ingested due to quality issues')
        measurand_variables = []
    ds.close()

    return dics, measurand_variables
```

LUTs_conversion.py

```
import collections
import json
import numpy as np
from bson import json_util
import h5py
import config

db, coll_wigos, coll_iso, coll_cci, coll_LUTs, coll_meas, coll_colloc, coll_cci_colloc, coll_cci_colloc_IDs, mainpath, output_dir,
wigos_fields_filepath, iso_fields_filepath, netcdf_global_attr_fields_filepath, \
   netcdf_variable_attr_fields_filepath, variable_names_units_filepath, to_be_ingested_dir, ingested_dir, rejected_dir =
config.paths()



def LUTs_to_dict(f, origFile, coll_LUTs, mainpath):

   logFile_LUTs = mainpath + 'logfile_LUTs_to_db.txt'
   measurand_variables = []
   dics = []

   with h5py.File(origFile, "r") as hf:
      keys = list(hf.keys())
      for key in keys:
         data = hf.get(key)

         if key == 'colocUncertainty':
            i_start = 0
            for i in range(120, 480, 120):
               np_data = np.array(data[i_start:i])
               p = ('Shape of the array ' + key + ': \n', np_data.shape)
               vals = np_data.tolist()

               d0 = collections.OrderedDict()

               d0['LUT_name'] = f[:-3]
               d0['variable_name'] = key
               d0['longitude_range'] = [i_start-180, i-180]
               units = data.attrs['Units'].decode()
               d0['units'] = units
               try:
                  dims = data.attrs['Dimensions'].decode()
                  d0['dimensions'] = dims
               except KeyError:
                  u = 0
               d0['values'] = vals
               dic = d0
               dic_to_json = json.dumps(dic, indent=4, default=json_util.default)
               dics.append(dic)
               i_start = i



         else:
            np_data = np.array(data)
            p = ('Shape of the array ' + key + ': \n', np_data.shape)
            vals = np_data.tolist()

            d0 = collections.OrderedDict()
            d0['LUT_name'] = f[:-3]
            d0['variable_name'] = key
            try:
               units = data.attrs['Units'].decode()
               d0['units'] = units
            except KeyError:
               u=0
            try:
               dims = data.attrs['Dimensions'].decode()
```

```
            d0['dimensions'] = dims
        except KeyError:
            u=0
        d0['values'] = vals
        dic = d0
        dic_to_json = json.dumps(dic, indent=4, default=json_util.default)
        dics.append(dic)

return dics
```