––––

Harvester v0

Issued by:

Date:

Ref:

Official reference number service contract:

This document has been produced in the context of the Copernicus Climate Change Service (C3S).

The activities leading to these results have been contracted by the European Centre for Medium-Range Weather Forecasts, operator of C3S on behalf of the European Union (Delegation Agreement signed on 11/11/2014). All information in this document is provided "as is" and no guarantee or warranty is given that the information is fit for any particular purpose.

The user thereof uses the information at its sole risk and liability. For the avoidance of all doubts, the European Commission

and the European Centre for Medium-Range Weather Forecasts has no liability in respect of this document, which is merely representing the authors view.

Contributors

UNIVIE

F. Ambrogi

M. Blaschek

L. Haimberger

# Introduction

In this document we describe the main features of the 'harvesting tool' EAU\_data\_harvester.py. It is designed to automatically collect radiosondes data, available in several datasets and convert them to a file format compliant with the Common Data Model (CDM) requirements. This process constitutes an intermediate step towards the creation of merged station files, where data collected from different sources are combined together and merged into a single file; the merging process is object of the deliverable XXX and it is extensively described in the accompanying report. The file structure of both intermediate and final merged station files is identical, so that it is easier to read and process data from different data source without the need of different codes.

# Source Datasets

The radiosonde data are contained in seven different source datasets, that for brevity are labelled as ERA5\_1, ERA5\_1759, ERA5\_1761, ERA5\_3188, NCAR, IGRA2 and BUFR. Here we provide details for each of the dataset.

ERA5\_1 :

RA5\_1759:

ERA5\_1761:

ERA5\_3188:

NCAR:

IGRA2:

BUFR:

# Workflow Overview

The main workflow of the tool can be divided into two complementary parts. At first, the original source files are downloaded from the proper websites, or retrieved as in the case of ODB files. In the second part, the files are processed and converted to netCDF files. In particular, the structure of the files strictly follows the CDM requirements defined in the public GitHub project:

https://github.com/glamod/common\_data\_model/ .

We remind that the CDM is designed to store all the relevant information, being data or metadata, to unambiguously identify and characterize the observed data. The information is split into several tables, where each entry is defined in proper auxiliary tables that provide the definition of each entry.

The CDM defines a set of guidelines which implemented in the output files of the harvester tool, i.e. the content of the netCDF files is structures so that the data is stored according to the tables defined in the various documents obtainable in the above GitHub repository.

The workflow, common for any of the source file, is here summarized:

1. the source dataset is downloaded and stored on the user's disk;

2. the input data is read in memory;

3. the CDM tabled are downloaded from the GitHub repository and read in memory;

4. the input data is compared and matched with the definitions of variables in the CDM tables;

5. the retrieved data, now formatted according to the CDM definitios, are stored in netCDF files.

Regarding the point 1) above, we note that for the download of the NCAR dataset, the user must provide his own identification user name and password,

which can be obtained after registration on the website.

Regarding 2), each data source has different input files, so that the script contains different methods for reading the input source.

Regarding 3), the automatic download of the CDM tables is a guarantee that the version of the tables considered always corresponds to the latest available

Regarding 4), the existing CDM tables require specific extension to incorporate radiosonde data. This will be described in a dedicated paragraph.

Regarding point 5), the choice of the format netCDF was already discussed extensively during in-person meeting at the ECMWF centre, during 2019 Copernicus General Assembly, and in several o-line meetings with the staff responsible for the implementation od the database on the CDS (Copernicus Data Service). The format seems suitable for fast and reliable data retrieval from the users. The delivery and upload of the dataset on the CDS is object of the deliverable XXX and will be thereby extensively discussed. Here we simply note that the netCDF format is written according the CF xx requirements, and it is easily accessible via the *xarray* python module, which enables to easily convert the data to other format such as a plain CSV table.

Special cases for the CDM tables

In order to implement all the necessary data and metadata for radiosondes measurements, the following extensions to the CDM tables have been implemented.

Reanalyses feedback

It is not currently possible to implement reanalyses feedback (such as first guess and analyses departures) the CDM tables. The necessary extensions were proposed to the CDM governance xxx and eventually approved. The extension implies a modification in the observations\_table, with the addition of the variable advanced\_feedback, representing a flag value, where 0 is the absence of reanalyses information, and 1 is the availability for such data. The proper data are found in a dedicated netCDF group called 'era5fb' (the content of the netCDF files will be explained in details later on). Note that currently only the ERA\_1 dataset provides reanalysis data, however, with the proposed extension of the CDM tables, it will be straightforward to implement an arbitrary number of reanalyses data.

Z coordinates

In the case of radiosondes, the z coordinate is usually expressed as the pressure at a certain elevation of the sonde, being at significant or standard levels. The z\_coordinate\_type table was extended with the value '1', which stands for the value of the pressure in Pascal.

Station type

The station\_type is able to represents radiosondes (value 0) or pilot balloons (value 1).

crs

The crs table maps to value 0 to the 'wgs84' type. ### check what this is ???

1. Convertion from the original source files to netCDF format

In this section we describe the procedure to convert the original file in ODB (ERA5\_1, ERA\_1759, ERA5\_1761,ERA5\_3188) , bfr (BUFR) or text (IGRA2, NCAR) format to CDM compliant netCDF4.

Irrespectively of the specific routine used to read the different input files, the script will firt transform the data into pandas dataframe and subsequently to xarrays, that will be then written into the netCDF files. We reorder the dataframe according to the date and time information, and then the vertical coordinate (pressure level).

The variables in the xarray are then renamed and properly arranged according to the CDM tables downloaded from the GitHub (eventually extended as described above).

ERA5\_1 , ERA5\_3188, ERA5\_1759, ERA5\_1761

BUFR

The module *bufr\_to\_dataframe* converts the bufr files into xarray vectors.

The available variables are: pressure, temperature, dew point temperature, wind direction, wind speed and geopotential (not always available for all the stations).

NCAR

The module *uadb\_ascii\_to\_dataframe* converts the text files from the NCAR dataset into xarray vectors. Original documentation on the meaning of the various columns can be found in <https://rda.ucar.edu/datasets/ds370.1/docs/uadb-format-ascii.pdf> .

The documentation above describes how missing values or values that did not pass quality checks were attributed specific numerical codes. To produce uniform netCDF files, we replace the numerical code of the missing values declared in the documentation with numpy not-a-number values. This is done consistenly also for the other datasets.

However, we found a discrepancy between what is declared inside the documentation above and the content of the files regarding the numerical code attributed to missing values.

For example, in the case of pressure, the declared missing values should be given as -99999.0.

However, in the file e.g. *uadb\_trhc\_62271.txt*, we find the value 9999.9 appearing multiple times) to the pressure in the same record (i.e. the same launch of the sonde (in the case of significant levels), so it is not possible to associate unambiguously the values of the measured variables to a specific pressure level. We then also replace this values with missing values np.nan.

The variables provided in these files are: pressure, temperature, dew point temperature, wind direction, wind speed, geopotential and relative humidity.

As described in the previous paragraph, there are two separate files relative to the same station, once containing full information regarding temperature (air temperature, dew point relative humidity), and the other one containing full information regarding the wind (direction and force). The content of the files, however, might overlap (i.e. the temperature files might as well contain wind data and vice-versa). The structure of the file is identical, so a single reading routine is sufficient. For this step, the two files will be treated separately and will be both included in the merging procedure.

IGRA2

Similarly, the routine *igra2\_ascii\_to\_dataframe* reads the igra2 text files and produces xarrays, replacing the missing values with np.nans. The documentation can be consulted at <https://rda.ucar.edu/datasets/ds370.1/docs/uadb-format-ascii.pdf> .

The variables provided in these files are: pressure, temperature, dew point depression, wind direction, wind speed, geopotential and relative humidity. In addition, the sonde release time (i.e. exact date-time of the launch) is provided. This information is very useful, and will be stored in the *report\_timestamp* variable of the *header\_table*.

1. Structure of the output file and reading examples

The output files are in the 'netCDF4' format, produced used the 'h5netcdf' engine using the *'to\_netcdf'* functionalities of the *'xarray'* python module.

This allows to preserve the attributes associated to the variables in the file.

The output netCDF file contains two variables called *'recordindex'* and *'recordtimestamp'*, and a several distinct netCDF groups for each CDM table, such as the the *'observations\_table'*.

The first two variables contains auxiliary information that can be conveniently used for a fast access to the data along the time coordinate.

In fact, the *'recordtimestamp'* is a list of all the unique date-times of the observations, while the *'recordindex'* is an array containing the position, or index, of a particular date-time inside the array of the data. This means that the nth entry of the 'recordtimestamp' array maps to the nth entry of the 'recordindex' array, and the data relative to the particular date-time can be found in the data array (read for example with *‘xarray’* or *‘pandas’* dataframe) located at the 'recordindex' position.

These two variables can be efficiently used to check if the data for a particular date-time of interest are available in the file, and to extract them quickly by slicing the data at the proper position.

Below we provide a sample python script to read the basic content of an *example\_test.nc* file (note that the netCDF shell functionalities are also available, e.g. *ncdump* ).

## note: dateindex is not used anymore

*import netCDF4 as nc*

*example\_file = 'test.nc'*

*f = nc.Dataset(example\_file)*

*print(list(f.variables) )*

*['dateindex', 'recordindex', 'recordtimestamp']*

*print(list(f.groups) )*

*['crs', 'era5fb', 'header\_table', 'id\_scheme', 'observations\_table', 'observed\_variable', 'source\_configuration', 'station\_configuration', 'station\_configuration\_codes', 'station\_type', 'units', 'z\_coordinate\_type']*

# to access the recordindex and recordtimestamp variables

recordIndex = f.variables['recordindex']

print(recordIndex[:10])

[ 0 150 276 354 516 678 942 1104 1302 1524]

recordTimeStamp = f.variables['recordtimestamp']

print(recordTimeStamp)

<class 'netCDF4.\_netCDF4.Variable'>

int64 recordtimestamp(recordtimestamp)

units: hours since 1986-05-01 12:00:00

calendar: proleptic\_gregorian

unlimited dimensions:

current shape = (1273,)

filling off

# to access the observations\_table group

observations\_table = f.groups['observations\_table']

The core table of the file is certainly the *observations\_table*, which holds the measured variables, while all the others contain essentially metadata (either relative to the observing station, the sonde, the launch of the sonde, name of the original source file etc. ).

Note that the structure of the netCDF files here described will be maintained for the 'merged' files, which will be provided in the first release of the radiosonde measurements database v0 in the deliverable XXXX . We refer to the relative accompanying document for the extensive discussion regarding the merging procedure, i.e. the combination of data from different datasets into a single file, one for each observing station.

1. Station combination

Once the datasets have been converted, we perform a combination of the data relative to the same stations but contained in several files.

This is the case, for example, of the observing station xxx , identified by the WMO identification number XXXX .

In the deliverable XXX we described how we compiled a summary of the available data in the various datasets,

identifying the observing station of the data from the latitude and longitude information, and extracted a WMO identification number.

We found that multiple files contain data relative to the same station, that we combine together with the script pre-merging.py .

The script works as follows.

From the stations inventories mentioned above, we extracted the list of files containing data from the same observation station. These data must be combined together into a single file, one per each station.

The script then reads the data from all of the input netCDF files, and create a single output netCDF file. The structure of the file remains unchanged, as described above.