Copernicus Climate Change Service Global Land and Marine Observations Database

metmetpy

User manual

**Issued by:** NUIM / Peter Thorne

**Date:** 31/12/2018

**Ref:**C3S\_D311a\_Lot2.3.4.4-2018\_201812\_ Marine\_code\_v1

**Official reference number service contract:** 2017/C3S\_311a\_Lot2\_NUIM/SC1

Contributors

natural environment research council, National oceanography centre (NERC noc)

1. David I. Berry

2. Elizabeth C. Kent

natIONAL UNIVERSITY OF IRELAND MAYNOOTH

1. Peter Thorne

2. Corinne Voces

National oceanic and atmospheric administration’s national centers for environmental information

1. Eric Freeman

# Table of Contents

[1. Table of Contents 3](#_Toc11310799)

[2. Introduction 5](#_Toc11310800)

[3. Tool overview 6](#_Toc11310801)

[4. Use, input, output and syntax 7](#_Toc11310802)

[4.1 Input and output 7](#_Toc11310803)

[4.2 Syntax 8](#_Toc11310804)

[4.3 Combining data from different data models 9](#_Toc11310805)

[5. How to register a new data model mapping 10](#_Toc11310806)

[6. Data model mappings 12](#_Toc11310807)

[6.1 CDM tables mapping files and descriptors 12](#_Toc11310808)

[6.2 Mapping sequence 13](#_Toc11310809)

[6.3 Defining mapping functions 13](#_Toc11310810)

[6.4 Direct code table keys mapping 15](#_Toc11310811)

[7. CDM tables definition 16](#_Toc11310812)

[7.1 Updating tables from GLAMOD Common Data Model repository 16](#_Toc11310813)

[8. Replacing mappings 16](#_Toc11310814)

Executive Summary

The C3S 311a Lot 2 (Global Land and Marine Observations Database) service is concerned with the provision of globally available land and marine surface meteorological records. The service includes inventorying of, and brokering access to, data sources, their harmonization (via conversion to a Common Data Model (CDM), merging, and quality assurance) and their provision via the Copernicus Climate Change Service (C3S) Climate Data Store (CDS).

The version history is given below:

|  |  |  |
| --- | --- | --- |
| **Version** | **Release Date** | **Release notes** |
| 1.0 |  |  |
| 2.0 |  |  |

# Introduction

The Copernicus Climate Change Service (C3S) Global Land and Marine Observations Database service provides brokered access to global historical holdings of surface meteorological observations. It builds upon existing national, regional and global efforts to create an augmented set of quality assured holdings that can be used to create datasets, products and services.

**This document will contain relevant detailed information on the software and configuration used to produce the marine data holdings for service providers to be able to access, use, modify and/or update where necessary. The current version of this document is focused on the data flow leading to the December 2018 beta release (header and observations tables) and the scripts and software tools derived from it.**

The document is ordered as follows:

* Section 2 summarizes the marine code requirements.
* Section 3 describes the marine main data flows.
* The Annex to this document provides further insight into scripts and tools.

# Tool overview

*cdm* is a python3 tool designed to map observed variables and its associated metadata from a data model or models combination (*imodel*) to the C3S CDS Common Data Model (*CDM*) where:

* **Input data** is the *imodel* data elements in a unique pandas DataFrame with its attributes (data type, native precision-number of decimal places…) available in a python dictionary.
* **Output** is a set of ascii field delimited files each with one of the *CDM* tables for which a mapping has been defined. Optionally, the intermediate mapped python object can also be accessed.
* The full transformation from *imodel* to the *CDM* is ***internally*** defined in the *imodel* subdirectory of the mappings’ library by:
  + A set of *cdm\_table*.json files, defining the mappings to each *CDM* table from the *imodel.*
  + A *imodel*.py file with the full set of transforming functions to *CDM* elements from the *imodel*.
  + A set of json files with the code table mappings between the *imodel* and the *CDM*
* With the above settings, this tool supports mapping to *CDM* elements by:
  + Direct mapping from an *imodel* element
  + Direct assignment from the *imodel* element’s attributes
  + Assignment of a *default* value
  + Transformation: *imodel* element’s transformation including any combination of the following:
    - element combination
    - simple parametrization (input keyword arguments)
    - *imodel* element’s attributes
    - *cdm* element to *cdm* element: currently not supported.
  + *imodel* or CDM coded values:
    - From *imodel* key to CDM key (mmm… see nested dictionaries)
    - From *imodel* key to CDM value: can be done via transforming functions
    - From *imodel* value to CDM key is not likely initially, but can be done via transforming functions

# Use, input, output and syntax

## Input and output

The *cdm* tool maps the data elements stored in a *python* *pandas* *DataFrame* object to the CDM.

Input to the tool is:

1. A *python pandas DataFrame* object (or *pandas.io.parsers.TextFileReader* object) with data values in columns. Columns can be simple or multiindexed, to accommodate models with data in multiple sections and/or combination of data models.
2. A *python dictionary* with attributes of the elements in the input data. Currently, used attributes are (…)

Explain here that the data model must be mapped in the tool to the CDM

The output of the mapping is a *python* dictionary with the {*cdm\_table\_name*:*cdm\_table\_object*} pairs, that can be afterwards printed to ascii files with submodule xxxx. Where:

* *cdm\_table\_name:* name of the CDM table (i.e. header, observations\_at,….)
* *cdm\_table\_object*: *python* dictionary with the {*data*:*cdm\_table\_object,atts*:*cdm\_table\_atts*} pairs, where:
  + *cdm\_table\_object* is a *python pandas DataFrame* object with the CDM elements aligned in columns. Where:
    - The columns are the CDM data elements for which a mapping from the *imodel* is defined (there are no empty or null columns for CDM data elements not defined)
    - The CDM DataFrame index preserves the indexing of the input data
  + *cdm\_table\_atts*: python dictionary with the CDM element attributes, of which ??? *column\_type* and *decimal\_places* (inherited from the *imodel* mapping) are used to print the CDM table files.

CDM tables can be afterwards printed to an ascii files ready for insertion in the database with the xxxx submodule. The resulting set of files have the following characteristics:

* One ascii file per CDM table.
* One header line with column names.
* All columns defined in the CDM table are printed even if all records are null or not defined in the *imodel* mapping.
* observations\_\* tables have records with no observed value removed.

## Syntax

The calling sequence to map a model to the CDM from a python session is:

cdm\_dict = cdm.map\_model(imodel, data, atts, cdm\_subset = None, log\_level = ’INFO’)

List of arguments:

***imodel:*** name of imodel as registered in the tool (see ). Type: string.

***data:*** input data to map. Valid data types are: *pandas* DataFrame or io.parsers.TextFileReader objects or in-memory text streams (io.StringIO object). Type: string.

***atts:*** dictionary with the {element\_name:element\_attributes} of ***data***. Type: string.

List of keyword arguments:

***cdm\_subset:*** subset of CDM model tables to map. Defaults to the full set of CDM tables defined for the *imodel*. Type: list.

***log\_level:*** Defaults to ‘DEBUG’. Type: string.

The calling sequence to print the CDM table objects to ascii files is:

cdm.cdm\_to\_ascii(cdm\_dict, delimiter = ‘|’, extension = ‘psv’, null\_label = ‘null’, out\_dir = None, suffix = None, prefix = None, log\_level = ’INFO’)

List of arguments:

***cdm\_dict:*** name of imodel as registered in the tool (see ). Type: string.

List of keyword arguments:

***delimiter:*** field delimiter. Defaults to pipe. Type: string.

***extension:*** file extension. Defaults to ‘psv’. Type: string.

***null\_label:*** label for missing values. Defaults to ‘null’. Type: string.

***out\_dir:*** output directory. Type: string.

***suffix:*** suffix to add to table name to create the output filename. Type: string.

***prefix:*** prefic to add to table name to create the output filename. Type: string.

***log\_level:*** Defaults to ‘INFO’. Type: string.

## Combining data from different data models

In some cases, it is desired that the final CDM set of tables is composed of a combination of different data models/sources. Based on the IMMA1 reprocessing experience so far, this can be the case of:

1. Using a supplemental data element instead of the corresponding element in the IMMA1 core or attachments (i.e. to improve the final product by rescuing higher quality/precision supplemental data)
2. Adding data elements from a different data source (like adding WMO PUB 47 metadata)

For the first case, it is recommended that the input DataFrame and attributes have both data sources (the main-IMMA1 and supplemental) and that a mapping set (*imodel*) is defined specifically for that main-supplemental combination. This is to avoid the burden of:

* dependencies on element substitutions (ie. latitude, location\_accuracy)
* also to avoid the uncomfortable column replacement process when the input/output DataFrames are *pandas.io.parsers.TextFileReader* objects rather than plain DataFrames

If this approach is not possible or desired, separate mapping from different data models to the CDM can be done and column replacement performed afterwards, before printing the tables to ascii files. This is possible if both initial data sources have the same indexing, since indexing is preserved in the output CDM DataFrame objects before printing to ascii files, and thus column replacement can be done directly (although iterating over chunks if input/output DataFrames are *pandas.io.parsers.TextFileReader* objects)

For the second case, where data sources are completely different in nature ( IMMA1 individual reports Vs WMO PUB 47 station metadata), it is recommended to map both things separately and then make the appropriate replacements/additions based on the corresponding CDM element matching (i.e. ‘primary\_station\_id’). There should be a tool in the *common* module of *cdm* to do this.

# Station ID correction methods

State in corresponding json file using the following key\_value pairs in the ***corrections*** key:

* ***fillna***: value
* ***map***: name\_map in metmetpy/station\_id/correction\_maps/name\_map.json
* ***function***: function name in metmetpy/station\_id/correction\_functions.py submodule
* ***replace\_patterns***: dictionary collection of pattern:replacement to apply.

Corrections are supposed to be performed sequentially as stated in the ***corrections*** key.