Metadata Standard

The goal of the CM SAF Metadata Standard is to make our products more uniform, which will not only improve the user experience but also facilitate our daily work with the data. We try to keep it in sync with the standards from C3S and obs4MIPs in order to be prepared for contributing to these projects.

The standard is mandatory for every newly generated TCDR and its associated ICDR.

A sample file can be found here.

Version

Version 2 (CDOP-3), March 2020.

File Names

All products must follow the CMSAF file naming convention.

Format

New CM SAF products shall be distributed in netCDF4 format with internal compression using zlib. The optimum compression level depends on the data and processing constraints.

Metadata

Metadata make a dataset self-describing and drastically improve its usability. The CF Conventions are our primary metadata standard. They are widely accepted in the climate and forecast community. In addition to the CF conventions we agreed to also comply with the Copernicus common data model specification and partly follow the Attribute Convention for Data Discovery as well as obs4MIPs. The resulting metadata standard is summarized in the sections below.

Global Attributes

Attribute	**Content**	**Example**	**Comment**
title	Dataset Title	CM SAF FCDR of SSM/I brightness temperatures	
summary	Dataset Summary	This dataset contains Fundamental Climate Data Records (FCDR) of Special Sensor Microwave/Imager (SSM/I) brightness temperatures compiled by the Satellite Application Facility on Climate Monitoring (CM SAF).	

Attribute	Content	Example	Comment
id	DOI	DOI:10.5676/EUM_SAF_CM/FCDR_SSMI/V001	TCDR only
product_version	Dataset version as text: Major.Minor	1.0	
creator_name	Creator name	DE/DWD	GCMD Providers if possible
creator_email	contact.cmsaf@dwd.de		Fixed
creator_url	http://www.cmsaf.eu/		Fixed
institution	EUMETSAT/CMSAF		Fixed
project	Satellite Application Facility on Climate Monitoring (CM SAF)		Fixed
references	Link to DOI resolver	http://dx.doi.org/10.5676/EUM_SAF_CM/FCDR_SSMI/V001	For ICDRs specify the WUI landing page
keywords_vocabulary	GCMD Science Keywords, Version 8.6		Minimum Version
keywords	comma separated list from GCMD Science Keywords	EARTH SCIENCE > SPECTRAL/ENGINEERING > MICROWAVE > BRIGHTNESS TEMPERATURE	
Conventions	comma separated list	CF-1.7, ACDD-1.3	Minimum Version
standard_name_vocabulary	Standard Name Table (v51, 16 May 2018)		Minimum Version
date_created	ISO 8601:2004	YYYY-MM-DDThh:mm:ss<zone>	
geospatial_lat_units	from udunits	degrees_north	
geospatial_lat_min	as double	-90.0	=leftmost lat bound
geospatial_lat_max	as double	90.0	=rightmost lat bound
geospatial_lat_resolution	as text	0.5 degree	if applicable
geospatial_lon_units	from udunits	degrees_east	
geospatial_lon_min	as double	-180.0	=leftmost lon bound
geospatial_lon_max	as double	180.0	=rightmost lon bound
geospatial_lon_resolution	as text	0.5 degree	if applicable
time_coverage_start	ISO 8601:2004	YYYY-MM-DDThh:mm:ss<zone>	=leftmost time bound
time_coverage_end	ISO 8601:2004	YYYY-MM-DDThh:mm:ss<zone>	=rightmost time bound
time_coverage_duration	ISO 8601:2004	P[YYYY]-[MM]-[DD]T[hh]:[mm]:[ss]	if applicable
time_coverage_resolution	ISO 8601:2004	P[YYYY]-[MM]-[DD]T[hh]:[mm]:[ss]	if applicable
platform	comma separated list from GCMD Platform List	DMSP 5D-3/F16 > Defense Meteorological Satellite Program-F16	if applicable
platform_vocabulary	GCMD Platforms, Version 8.6		Minimum version
instrument	comma separated list from GCMD Instrument List	SSMIS > Special Sensor Microwave Imager/Sounder	if applicable
instrument_vocabulary	GCMD Instruments, Version 8.6		Minimum version
history	as text	„Wed Jun 28 11:22:20 2017: ncatted -a myattr,global,a,c,myvalue myfile.nc“	if applicable
date_modified	ISO 8601:2004	YYYY-MM-DDThh:mm:ss<zone>	if applicable
variable_id	Comma separated list of primary variables in the file	ctp,cth,ctt	

Attribute	Content	Example	Comment
license	The CM SAF data are owned by EUMETSAT and are available to all users free of charge and with no conditions to use. If you wish to use these products, EUMETSAT's copyright credit must be shown by displaying the words "Copyright (c) ([release-year]) EUMETSAT" under/in each of these SAF Products used in a project or shown in a publication or website. Please follow the citation guidelines given at [DOI-landing-page] and also register as a user at http://cm-saf.eumetsat.int/ to receive latest information on CM SAF services and to get access to the CM SAF User Help Desk.		Replace descriptors in square brackets with dataset specific information. Use \n\n for the double line break.
source	Colon separated list of Input data	CM SAF FCDR of SSM/I brightness temperatures : ERA-5	Not finalized yet
lineage	Colon separated list of processing steps applied to input data (ISO Lineage model 19115-2)	pygac-1.2.3 : PPS-2014	Not finalized yet

Keyword/Platform/Instrument Vocabularies can be found here.

Additional Global Attributes

Of course you can add more global attributes. We recommend adding a CMSAF_ prefix in order to prevent name conflicts with the CF/ACDD Conventions. Here are some ideas to get started:

Attribute	Content	Example	Comment
CMSAF_processor	Overall (Re)processing framework	claas-v2.5.0	
CMSAF_L2_processor	Software used to generate level 2 products	SAFNWC-MSGv2012, CPPv5.1	
CMSAF_L3_processor	Software used to generate level 3 products	CMSAFMSGL3_V2.1	
CMSAF_orbits	Number of orbits contributed by each platform	NOAA-18=12, NOAA-19=11, METOP-A=10	
CMSAF_repeat_cycles	Number of repeat cycles contributed by each platform	METEOSAT-10=48, METEOSAT-11=48	

Consider adding temporally varying metadata not only as global attributes but also as variables. This facilitates merging multiple timesteps into one file without losing metadata of a particular timestep.

Variable Attributes

Attribute	Content	Example	Comment
long_name	Variable name written out	Cloud Fraction	Exception: Bounds variables
standard_name	CF Standard Name	cloud_area_fraction	If any, see Standard Name Table. You may also propose new standard names.
units	Physical units	%	If applicable, udunits compatible
cell_methods	Applied statistics	time: area: mean (interval: 15 minutes interval: 3 km)	Aggregated variables only (*7.3: Cell Methods*)
ancillary_variables	Ancillary variables	nobs, quality	Use this to reference number of observations, quality, standard deviation etc. (if any, *3.4. Ancillary Data*)
flag_values, flag_masks, flag_meanings	Flag decoding instructions	flag_values=[0, 1, 2], flag_meanings='good medium bad'	Flag type variables only (*3.5. Flags*)
bounds	Reference to corresponding bounds	lat_bnds	Coordinate variables only (*7.1. Cell Boundaries*)
add_offset, scale_factor	Unpacking parameters		If applicable (*8.1. Packed Data*)

In italics: Corresponding section in the CF conventions document.

Coordinates

- Each coordinate variable (time, lat, lon, …) must be characterised by cell boundaries as described in CF Standard chapter „7.1. Cell Boundaries“. Note: If adjacent intervals are contiguous, the shared endpoint must be represented indentically in each instance where it occurs in the boundary variable. For example, if the intervals that contain points `time(i)` and `time(i+1)` are contiguous, then `time_bnds(i+1,0) = time_bnds(i,1)`.
- Time coordinates must represent the left boundary of the covered temporal interval.
- Geographical coordinates of regular grids must represent the centre of the gridbox. `(lon, lat) = (0, 0)` must be the lower left corner of one grid cell.
- In case of static irregular grids (e.g. geostationary projection) you can save data volume by moving the twodimensional geographical coordinates to a separate auxiliary file. If doing so, please add two dimensions identifying the position in the image (row, column). This information is needed to identify the position in the original image after cutting a subdomain.

Precision

Always use 64 bit double for coordinates (lat/lon, time, pressure levels, …) and round the values to the number of significant digits in order to minimize floating point errors. Choose a time origin close to your dataset (not Julian Day for example).

See the Appendix for more details on coordinate precision.

Missing Records

If a product could not be generated for whatever reason (missing input data, processing failure, …), an „empty“ product containing only fill values has to be generated. Composite files consisting of multiple timestamps must always contain the same number of timestamps. If no data could be generated for a certain timestamp, all variables must be set to fill value at that particular timestamp.

In order to quickly indicate the overall status of each record in a file, every file must provide a `record_status` variable. Example:

```
netcdf test {
dimensions:
        time = 1234 ;
variables:
        byte record_status(time) ;
                record_status:long_name = "Record Status" ;
                record_status:comment = "Overall status of each record
(timestamp) in this file. If a record is flagged as not ok, it is
recommended not to use it." ;
                record_status:flag_values = 0B, 1B, 2B ;
                record_status:flag_meanings = "ok void bad_quality" ; }
```

The default for valid records is 0 (ok). If a record is missing, set the corresponding status to 1 (void). Quality concerns should be indicated with record status 3 (bad_quality).

How To Check Your Files

We encourage you to check your files against the standard *before* production using both the official CF Checker and the custom CM SAF Checker (CentOS servers only):

```
module load cmsaf/2019.01 python/3.7.2 cf_checker/4.0.0 cmsaf/checker/cdop3
cfchecks file1 file2 ... fileN
cmsaf_checker.py -c file1 file2 ... fileN
```

Appendix: Coordinate Precision

Although the definition of coordinate arrays is a straightforward task, the results may be unexpectedly inaccurate due to floating point errors. In the following example we present four different methods highlighting common pitfalls. In the following example we present four different methods highlighting common pitfalls.

Assume you have a global dataset on a regular 0.05×0.05 degree grid. The coordinate values represent the center of a grid cell and the cells are arranged symmetrically around zero, i.e. longitudes go from -179.975 to 179.975 in 0.05 degree steps:

```
N = int(7200)
dlon = double(0.05)
lon_min = double(-179.975)
```

Here are four different methods to compute the longitudes:

- *inc_f32*: Incrementing 32bit floats

```
lon = allocate(size=N, type=float)
lon[0] = float(lon_min)
for idx in 1...N-1:
    lon[idx] = lon[idx-1] + float(dlon)
```

- *lin_f32*: Linear „extrapolation“ using 32bit floats

```
lon = allocate(size=N, type=float)
for idx in 0...N-1:
    lon[idx] = float(lon_min) + idx*float(dlon)
```

- *rnd_f32*: Use *lin_f32* and round the result to the number of significant digits (3 in this case):

```
lon = allocate(size=N, type=float)
for idx in 0...N-1:
    lon[idx] = round(float(lon_min) + idx*float(dlon), digits=3,
type=float)
```

- *rnd_f64*: Like *rnd_f32*, but using 64bit double instead of 32bit float:

```
lon = allocate(size=N, type=double)
for idx in 0...N-1:
    lon[idx] = round(lon_min + idx*dlon, digits=3, type=double)
```

Note that rounding is equivalent to computing exact coordinates in integer space and converting them to float/double in the very end:

```
lon = allocate(size=N, type=double)

# Determine conversion factor for lossless conversion to integer
digits = 3
factor = double(10^digits)

# Compute exact coordinates in integer space
lon_i = allocate(size=N, type=long)
lon_min_i = round_int(lon_min*factor)
dlon_i = round_int(dlon*factor)
```

```
for idx in 0...N-1:
    lon_i[idx] = lon_min_i + idx*dlon_i

# Convert to double
for idx in 0...N-1:
    lon[idx] = double(lon_i[idx]) / factor
```

The function `round_int` rounds to the nearest integer.

How do the different methods perform in terms of accuracy? In order to determine accuracy, we need a reference. But even the reference is not exact, because the majority of numbers is not exactly representable by a floating point datatype. For example, the 32bit float closest to -179.975 is -179.975006103515625 and the 32bit float closest to 0.05 is 0.0500000007450580596923828125. 64 bit floats will be closer, but not exact either. As you might expect *rnd_f64* yields the most accurate results on a 64bit machine, so we choose it as our reference.

As you can see in figure 1, method *inc_f32* has the worst performance, because the representational errors accumulate in each loop cycle. After 7200 iterations the error adds up to almost 25% of the grid resolution! The linear extrapolation method *lin_f32* performs significantly better, because every coordinate is obtained by only one operation. Rounding the results to the number of significant digits (*rnd_f32*) yields an even higher accuracy, because computational errors are eliminated and only representational errors remain. At this point only a larger number of bits can further increase the accuracy. Please note that the 2nd and 3rd plot appear „filled“ because the data is oscillating with a high frequency.

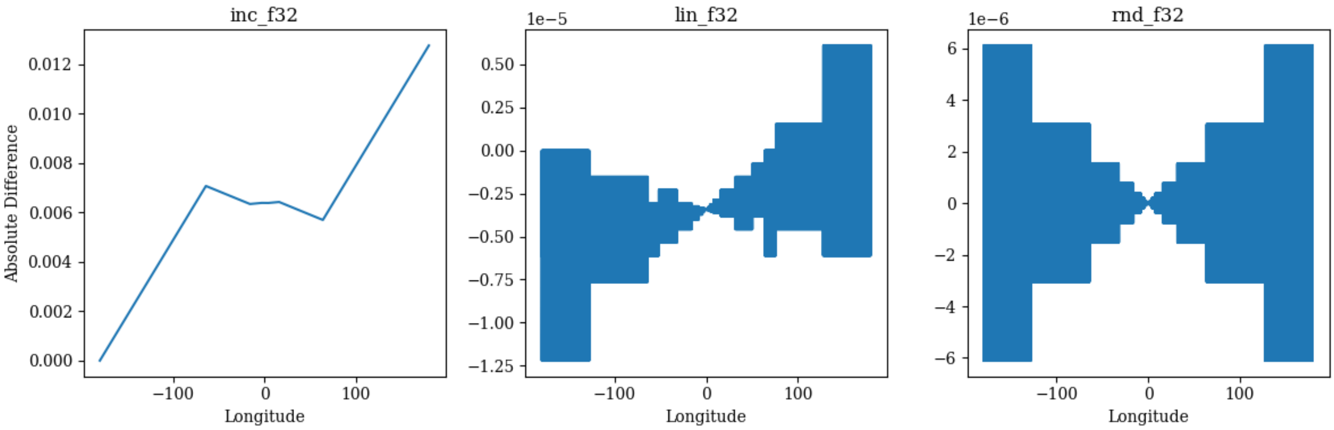

Figure 1: Absolute difference from reference rnd_f64

Another measure for accuracy is the spacing between two adjacent coordinates, which is shown in figure 2. Using coordinates of type double drastically reduces fluctuations of the grid spacing. But keep in mind that even 64bit coordinates are not exact.

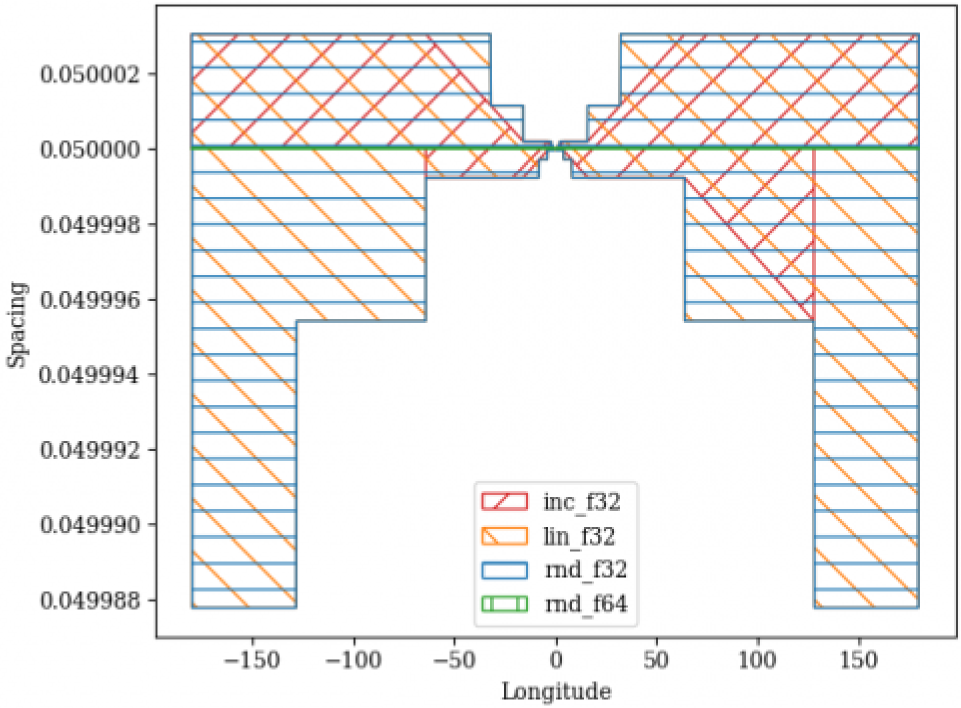

Figure 2: Spacing between adjacent coordinates. Hatched areas indicate the fluctuation range of the quantity.

The above results also show that the density of floating point numbers is largest near zero (→ smaller representational errors) and decreases exponentially towards larger values (→ larger errors). See What Every Computer Scientist Should Know About Floating-Point Arithmetic for more details than you can handle.

Conclusion

- Always use 64 bit double for coordinates (lat/lon, time, pressure levels, …) and round the values to the number of significant digits if possible.
- Double check your standard numeric library! For example `numpy.arange(start, stop, step, type='float32')` in Python is even worse than *inc_f32*!