DOE/SC-ARM-15-004

ARM Data File Standards Version: 1.3

ARM Standards Committee

September 2020

DISCLAIMER

DOE/SC-ARM-15-004

ARM Data File Standards Version: 1.3

ARM Standards Committee

September 2020

Work supported by the U.S. Department of Energy,
Office of Science, Office of Biological and Environmental Research

Standards Committee

This document was prepared by the 2020 Standards Committee consisting of:

Table 1. ARM Standards Committee

Group	Primary Member	Incoming Member*
VAP/Ingest*	Krista Gaustad (PNNL)	Brian Ermold (PNNL)
Metadata Reviewer	Maggie Davis (ORNL)	Ric Cederwall (ORNL)
ARM Data Center	Harold Shanafield (ORNL)	Robert Records (ORNL)
DQ Office	Ken Kehoe (OU) [Chair]	Alyssa Sockol (OU)
Translator/Mentor	Jenni Kyrouac (ANL)	Damao Zhang (PNNL)

The incoming member may assume the role of primary member after a term.

*The VAP Manager must be a member of the committee.

ANL = Argonne National Laboratory *ORNL = Oak Ridge National Laboratory*

OU = University of Oklahoma *PNNL = Pacific Northwest National Laboratory*

Acronyms and Abbreviations

AAF	ARM Aerial Facility
ABLE	Argonne Boundary Layer Experiment
ADC	ARM Data Center
ADI	ARM Data Integrator
AGL	above ground level
AMF	ARM Mobile Facility
AOS	aerosol observing system
ARM	Atmospheric Radiation Measurement
ASCII	American Standard Code for Information Exchange
BoM	Bureau of Meteorology (Australia)
CF	Climate and Forecast
CGA	calibration, grooming, and alignment
DOD	Data Object Design
DOE	U.S. Department of Energy
DOI	digital object identifier
DQO	Data Quality Office
DQR	Data Quality Report
EBBR	energy balance Bowen ratio station
ECOR	eddy correlation flux measurement system
ECR	Engineering Change Request
GMT	Greenwich Mean Time
hdf	Hierarchical Data Format
IATA	International Air Transport Association
ID	identification
IOP	intensive operational period
jpg	Joint Photographic Expert Group
LASSO	LES ARM Symbiotic Simulation and Observation
LES	large-eddy simulation
MAOS	mobile aerosol observing system
MFRSR	multifilter rotating shadowband radiometer
mpg	Moving Picture Expert Group
MSL	mean sea level
NASA	National Aeronautics and Space Administration
netCDF	Network Common Data Format
NOAA	National Oceanic and Atmospheric Administration

NSA	North Slope of Alaska
PCM	Process Configuration Management
pdf	portable document format
PI	Principal Investigator
PNG	Portable Network Graphics
QC	quality control
RWP	radar wind profiler
SGP	Southern Great Plains
SIRS	solar and infrared radiation station
SWATS	soil water and temperature system
TAR	Tape ARchive
TWP	Tropical Western Pacific
UAS	unmanned aerial systems
URL	Uniform Resource Locator
UTC	coordinated universal time
VAP	value-added product

Contents

Tables

1.0 Introduction

1.1 Intended Audience

This document is intended for ARM infrastructure and any data user needing further explanation of the format of ARM data file standards.

1.2 Background

The U.S. Department of Energy (DOE) Atmospheric Radiation Measurement (ARM) user facility performs routine in situ and remote-sensing observations to provide a detailed and accurate description of the Earth atmosphere in diverse climate regimes. The result is a huge archive of various data sets containing observational and derived data (http://www.archive.arm.gov/archive-usage/). Continuing the current processing while scaling this to even larger sizes is extremely important to the ARM facility and requires consistent metadata and data standards. The standards described in this document will enable development of automated analysis and discovery tools for the ever-growing data volumes. It will enable consistent analysis of the multiyear data, allow for development of automated monitoring and data health status tools, and allow future capabilities of delivering data on demand that can be tailored explicitly for user needs. This analysis ability will only be possible if the data follows a minimum set of standards. This document proposes a hierarchy of *required* and *recommended* standards.

All new baseline data sets must adhere to *required* ARM standards to be published in the ARM Data Center, unless an exception is granted. The historical data will be brought into compliance with the standards as they are reprocessed or updated.

Where feasible, the standards listed in this document follow the Climate and Forecast (CF) convention. Using the CF standards will increase the usability of the data to the broader scientific community. A full description of the CF convention can be found at cfconventions.org.

Benefits of adhering to these standards include:

- consistency across datastreams
- code reuse by using consistent formats
- simple and consistent software able to read all standardized netCDF files
- files (netCDF data files) both human and machine readable as much as possible
- self-describing data files containing metadata describing how the data were collected.

1.3 Advantages of Following Standards

Adhering to the standards put forth in this document will allow automated utilities to function with minimal updates. Overall, if data products meet a required set of standards, the software products used to assess and/or display them can be developed much more efficiently. Adherence to the standards will lead

to better quality and more readily understandable netCDF files. The standards present a consistent "look and feel" to data users who are familiar with ARM standards.

As more products adhere to the standards, fewer exceptions must be added to data product software, such as value-added products (VAPs), when ingesting various input datastreams. For developers, encountering fewer exceptions results in reduced chances to introduce software errors and quicker development time. This lowers the costs for development, and unintended costs to the ARM facility through reprocessing tasks.

1.4 Examples of Tools Using Standards

The ARM facility has many individual software tools using the standards listed in this document. Conforming to the standards enables ARM to function efficiently and accomplish significantly more with fewer resources. Some examples of software tools dependent on adherence to the standards include:

- DQ Explorer, DQ Inspector, DQ Zoom, ACT, DQ Plotbrowser
- Arm Data Integrator, Ingest and VAP processing at the ARM Data Center (ADC), DFMON
- Data Discovery, storage, custom data files at the ADC
- Process Configuration Management and Metadata Service tool at the ARM Data Center.

Links to these tools can be found at http://i.arm.gov.

2.0 The Standards Hierarchy

The standards are divided into two groups.

2.1 Required Standards

Required standards must be met to comply with the ARM standard. To reference a standards version number, all required standards must be met except those allowed to deviate by the Standards Committee. Unless indicated, all standards listed are required. If the required standards are not met, data will not be published in the ADC unless an exception is granted.

A few standards are required with conditions. The few cases are explicitly described in this document. If the conditions are not met, the standard is not required (e.g., *missing_value* attribute).

2.2 Recommended Standards

Recommended standards are encouraged in order to increase the usability of the final data products by both the ARM infrastructure and ARM data users. Following recommended standards enables automatic status monitoring, automated extraction tools, and consistency of the data. The recommended standards will be labeled as recommended in this document. Not following these standards may result in the data set not being monitored for data quality status and not discoverable through the ARM operational tools.

3.0 Optional Methods

Some netCDF variables (i.e., quality control, source, or state indicator variable) or metadata (e.g., *cell_methods*) are optional and up to the discretion of the developer/mentor/translator to implement. All instances of optional methods are labeled as optional in this document. If an optional variable or metadata is used, the required and recommended standards listed in those sections apply.

3.1 Changes from Version 1.2

- Updated to use the term “variable” only instead of “variable” or “field” interchangeably in this document
- Change from using “unitless” to udunits compliant “1” for units
- Highly Recommend not using *valid_min* and *valid_max* attributes for quality control limits. Suggests using *fail_min* and *fail_max*.
- Use of netCDF4 for compression does not need Standards Committee approval
- Allow for a variable’s source attribute to list a variable name without the full *datastream*:variable format if both are within the same file
- Allow at creator’s discretion the use of time-varying inputs to be listed under a non-time-varying source *attribute* instead of requiring a source *variable*
- If *_FillValue* and *missing_value* are both defined recommend using the same value
- Addition of a “U” facility ID for unmanned aerial system
- Recommendations for co-located facility naming
- Recommend using *standard_name “quality_flag”* for all quality control variables.

4.0 File Type/Format

RAW instrument data is typically written in ASCII, binary, or netCDF data formats. Most formats are decided by the instrument vendor, not by the ARM facility. If an option is available, use best judgment when choosing a vendor data file format.

The ARM facility preferred choice for final data format is netCDF-3 model because it supports efficient data storage and reliable/robust documentation of the data structure. More information about netCDF is available at www.unidata.ucar.edu/software/netcdf/.

High-volume data may be treated as a special case and allowed to use netCDF-4 4 classic model to take advantage of the compression option. Use of netCDF-4 classic model to take advantage of the compression abilities requires a significant reduction in file size (a minimal reduction of 50% data volume) or increase in usability of the data.

ASCII, binary, and HDF formats are used for some external data products. When using ASCII or binary data formats, a description of the file structure and its proposed documentation must be easily available to the user. HDF is the standard for most satellite data. More information about HDF is available at www.hdfgroup.org and www.hdfeos.org.

5.0 Construction of Data Filename

5.1 File Naming Conventions for Processed Data

ARM netCDF files are named according to the following naming convention. All characters are lowercase except for the facility indicator. Only “a-z”, “A-Z”, “0-9”, and “.” characters are allowed.

(sss)(inst)(qualifier)(temporal)(Fn).(dl).(yyyymmdd).(hhmmss).nc

where:

(sss) is the three-letter ARM site identifier (e.g., sgp, twp, nsa, pgh, nim, ena, mag) with the primary purpose to provide general classification and grouping of the data. The identifier is defined by a geographic reference or the International Air Transport Association (IATA) three-letter airport code to indicate approximate location. Fixed sites are named after a geographic reference, while ARM Mobile Facility (AMF) deployments use the IATA code. For remote deployments not near an airport or deployments on moving platforms, it has become practice to use a 3-letter acronym of the campaign/experiment, e.g., mag, acx, awr. Exceptions may be made for large geographic areas for satellite data (e.g., gec, nac). For ARM Aerial Facility (AAF) deployments, the site code will be chosen from the airport the aircraft departs from unless the flight path is large and choosing a geographical area is more appropriate. When the AAF is flying over a fixed or AMF site, that site name will be used. The use of “OSC” to represent the general case of “offsite campaign” site code is depreciated. A distinct new site code for each off ite campaign is required unless the use of a current/previous site code is desired.

(inst) is the ARM instrument abbreviation (e.g., mwr, met, ecor, mpl), or the name of an ARM VAP. The abbreviation is typically an acronym describing the instrument suite or VAP, and may describe the method for retrieving the measured or derived quantity. We recommend not ending the instrument abbreviation iwith a number, as this can be confused with the data temporal resolution descriptor or other optional descriptors following the instrument abbreviation, unless the number is used as an enumeration to distinguish multiple instruments or multiple versions.

(qualifier) is an optional qualifier that distinguishes these data from other data sets produced by the same instrument or VAP (e.g., avg, 1long). The optional qualifier may have one or more additional qualifiers describing a specific algorithm method or instrument specifics. This qualifier is used to describe monthly, yearly, or annual files. We recommend that the qualifier not end with a number, as this can be confused with the data temporal resolution descriptor, unless the number is used as an enumeration to distinguish different qualifiers. (i.e., aosflow1 versus aosflow2 where the number indicates different versions).

(temporal) is an optional description of data temporal resolution (e.g., 30m, 1h, 5s, 200ms, 14d). All temporal resolution descriptors require a units identifier. Accepted abbreviations include: ns=nanosecond, us=microsecond, ms=millisecond, s=second, m=minute, h=hour, d=day, mo=month, yr=year. We recommend that the primary datastream for use by the end user not have a data integration period in the name. Time integration periods are converted to the lowest unit description. When possible, default to minutes. Example: 60 seconds is labeled as “1m”, 60 minutes is labeled as “1h”.

(Fn) is the two- or three-character ARM facility designation. A facility is designated with a capital letter followed by one or two numbers (e.g., S1, C1, E13, B4, M1, I4). A facility code padded with a 0 is considered a different facility than the one not using a padded 0. For example, S01 is not the same facility as S1.

(dl) data level is the two-character descriptor consisting of one lower-case letter followed by one number, except for RAW data level that will consist of two numbers (e.g., 00, a0, b1, c1, c2). See the Data Level section for further explanation.

(yyyymmdd) is the coordinated universal time (UTC) date in year, month, day-of-month format consisting of exactly eight characters, indicating the start date of the first data point in the file. Single-digit month and day values are padded with 0. Example: February 4, 2012 = "20120204".

(hhmmss) is the coordinated universal time (UTC) time in hour, minute, second format consisting of exactly six characters and indicates the start time of the first data point in the file. Single-digit values are padded with a "0". Sub-second times are truncated to the integer of the second value. Example: 5:00:19.57 UTC = "050019". The time sample may not exceed 23:59:59. Hours greater than or equal to 24, or minutes or seconds greater than 60, will cause problems with time conversion programs.

nc is the netCDF file extension. The CF convention file extension for netCDF files was changed from *cdf* to *nc* in 1994 in order to avoid a clash with the National Aeronautics and Space Administration (NASA) CDF file extension, or with "Channel Definition Format" files. A number of third-party utilities require the *nc* extension or build the tools expecting a *nc* file extension (i.e., Panoply, IDV, ncBrowse). For backwards compatibility, ARM will continue to allow the use of *cdf* file extension for historical data. As data is reprocessed, the filename extension will be updated to *nc* if feasible.

5.1.1 Filename Length

The **TOTAL** length of a filename sent to the ADC **MUST** be 60 characters or less to meet the requirements of the current ADC database system. The ADC uses a 64-character filename variable in the database, and appends a version level to the end of the filename. (Archive version descriptor examples: ".v1", ".v13"). Four characters are reserved for the period, "v" and 1- or 2-character numbers describing the version of the file received at the ADC.

In addition to full filename length, the datastream, (sss)(inst)(qualifier)(temporal)(Fn).(dl), **MUST be** 33 characters or less to comply with the ADC database.

The final filename length requirement includes limiting the instrument description part of a filename, (inst)(qualifier)(temporal), to 24 characters or less to comply with the ADC database.

5.1.2 Facility Code Descriptions

Facility Codes were originally used to distinguish spatially separated fixed measurement locations within a given site and to distinguish the locations based on the complement of instruments deployed. The meaning has been extended to refer to specific measurement assets (e.g., aerial platforms in AAF) and to distinguish data products that are not intended for wide distribution (D-facilities).

Some facility codes are now obsolete but have to be maintained because of the existence of data products in the ADC that are intended for public distribution. Here are the descriptions in alphabetical order:

A = Argonne Boundary Layer Experiment (ABLE) [Retired]

After the termination of ABLE, the ADC incorporated the ABLE data, because of the large overlap of both the type of instrumentation deployed (radar wind profilers [RWPs]) and some spatial overlap of the deployed locations.

B = Boundary Facilities [Retired]

The boundary facilities existed only during the large-footprint Southern Great Plains (SGP) atmospheric observatory era intended to capture the fluxes across the outer boundary of the SGP domain.

C = Central Facility

For the fixed locations having one or more facilities and is one of the main heavily instrumented sites.

D = Diagnostic Data Products

This is not a facility at all; it is neither a measurement platform, nor complement of instruments, nor tied to a specific location, but rather is attached to data products created for short-term diagnostic purposes to be shared by expert data users only and not intended for general distribution. Data with the D<#> facility code are to be made available to a small set of people performing tasks such as calibration, grooming, and alignment (CGA) requiring diagnostics files. This facility code should be used for real-time, onsite diagnostics and calibration.

E = Extended Facilities

The original concept of an extended facility at the SGP was a standard minimal set of instrumentation to be deployed at numerous locations within a global climate model grid-cell-size domain including the solar and infrared radiation station (SIRS), multifilter rotating shadowband radiometer (MFRSR), soil water and temperature system (SWATS), and energy balance Bowen ration station (EBBR)/eddy correlation flux measurement system (ECOR). The SGP facilities were numbered in increasing numerical order as they were created. Therefore there is no specific reasoning for the numbering scheme as was done with the other fixed-site extended facilities.

Extended facilities around a central facility at the remote locales with multiple central facilities (Tropical Western Pacific [TWP], North Slope of Alaska [NSA]) indicate the association of the extended facility to the central facility by matching the first of the two required digits to the central facility number. Example: central facility TWP-C2 is related to extended facilities E20, E21, E22, while TWP-C1 is related to E10, E11, E12.

F = ARM Aerial Facility −Manned Aircraft

Each of the manned airborne platforms that are part of or on loan to the ARM Aerial Facility (AAF) has a F-facility code designator and remains tied to the specific aircraft for each deployment (e.g., F1 =

Gulfstream G-1, F2 = Cessna). Unmanned aircraft will use the U facility ID. Duplicate instruments on a single aircraft will not use additional co-located facility indicators, but will rather have different instrument code names using the same facility indicator to describe the differences. Instrument location on the aircraft is fundamental to the measurement and results in a different measurement. For example, <SSS>aafcpcfcviF1.a1 and <SSS>aafcpcfisoF1.a1 use the same manufactured instrument but the location of the inlet has a significant influence on the measurement values. This also allows all instruments to use the same facility indicator for aircraft grouping.

I = Intermediate Facility

These facilities are typically instrumented with RWP and other radars and located at intermediate distances from the central facility to optimize radar coverage.

L = Local? Or Logistics? Facility [Retired]

Used only once for L3 = Blackwell/Tonkawa Airport, Oklahoma for the duration of a campaign, likely a reference instrument on the ground for comparisons with the aircraft instruments.

M = Mobile Facility

For each of the ARM Mobile Facility sites, the central location where all the data are collected and the majority of the instruments are deployed is designated M1, it is thus equivalent to C1 for the fixed sites.

N = Network of Measurement Locations

Used for derived data products that combine data from multiple sensor locations, e.g., all MFRSR data from all Extended Facilities at SGP.

Q = Quality Assurance (Pre-Deployment Integration) [Retired]

Only used for SHB for integration sites. Can be thought of as being superseded by the D-facility code that now serves a similar purpose.

S = Supplemental or Ancillary Facility

Most commonly used for Mobile Facility deployments for either additional instruments deployed away from M1 or for redundant instruments at the same location that require a different facility code to distinguish the data products running the same ingest.

Instances of a co-located deployment of the mobile aerosol observing system (MAOS) with the National Oceanic and Atmospheric Administration (NOAA) AOS will result in the MAOS using S1 for the facility indicator unless S1 is used to describe a different location than the location of the MAOS.

Supplemental facility designations (S<#>) is only used for mobile facility deployments, co-located facility indicator, or intensive operational period (IOP) data sets.

At the fixed sites, S0<#> has been used in the past to indicate a supplemental facility co-located with the main facility. The continuation of this convention is not recommended.

For non-aircraft moving platforms or duplicate instruments where the duplication is part of the measurement strategy, use two S facility indicators when the concept of a central or default instrument does not exist (i.e., ship RAD or ship MET winds). This will reduce confusion about using a single instrument from the M facility when both instruments must be used in tandem and using a single instrument is incorrect. Additional platforms must take the requirement of S1 and S2 facility code into consideration when determining the additional platform facility name. For example, additional sites as part of an AMF deployment may need to start at S3 to accommodate the primary ship use of M1, S1, and S2. We recommend using S1 for port, S2 for starboard, S3 for bow, and S4 for stern of the moving platform. Location (port/starboard/bow/stern) must be indicated in facility long name.

U = ARM Aerial Facility −Unmanned Aerial System

Unmanned aerial systems (UAS) use a U-facility code designator. AAF UASs will reserve a specific U facility ID for each UAS and these will not change when the aircraft is used with a different site ID. Facility IDs U1 to U39 are reserved for assigned aircraft. Unmanned aircraft that do not need a specific U facility ID will start at U40 and the ID number can be recycled. U1 is reserved for Data Hawk and U2 is reserved for ArcticShark.

Duplicate instruments on a single aircraft will not use additional co-located facility indicators: rather, they will have different instrument code names using the same facility indicator to describe the differences. Instrument location on the aircraft is fundamental to the measurement and results in a different measurement. For example, <SSS>aafcpcfcviU1.a1 and <SSS>aafcpcfisoU1.a1 use the same manufactured instrument but the location of the inlet has a significant influence on the measurement values. This also allows all instruments to use the same facility indicator for aircraft grouping.

X = eXternal Facilities

External data products that cover a large locale use the facility designation of X1, while data products specific to an ARM facility follow the extended facility two-numeral character naming. X10 is always reserved for external data specific to C1 or M1 (i.e., X20 for C2 or M2). All other locations near or associated with the C1 or M1 central facility are numbered X11, X12, etc. For example, external data associated specifically with the SGP-C1 facility would be named SGP-X10, while nearby locations are SGP-X11, SGP-X12, etc.

Some data products that are technically externally sourced have been adopted and run through the regular ARM ingest and given the appropriate ARM facility code, e.g., the Australian Bureau of Meteorology (BoM) sondes at TWP-C3.

5.1.3 Data Level

Data levels are based on the “level of processing” with the lowest level of data being designated as RAW or “00” data. Each subsequent data level has minimum requirements and a data level is not increased until ALL the requirements of that level as well as the requirements of all data levels below that level have been met. A data level will consist of one lower-case letter followed by one number (except for RAW data). Data files in a format other than netCDF follow the same requirements listed below except for the conversion to netCDF requirements.

00: raw data − primary raw data stream collected directly from instrument.

01 to **99**: raw data − redundant datastream, sneakernet data (transfer of data files by physically moving removable media), or external data that may consist of higher-order products, but require further processing to conform to ARM standards.

a0: raw data converted to netCDF. This data level is used as input to higher-level data products. Preliminary or not intended for distribution to data users.

a1: calibration factors applied and converted to geophysical units. This level is reserved for non-netCDF or non-numeric data products intended for distribution to users, for example, image files or movie files.

a2 to **a9**: further processing on *a1*-level data that does not merit b-level classification. This level also applies to external satellite files that are converted from TDF to HDF format. Example: instrument mentor reviewing the data and replacing bad data with missing value, or additional calibration factors added to data after data have been processed as an a1 datastream. A description of the further process must be included in the netCDF header, instrument handbook, or technical paper available to data users.

b0: intermediate-quality-controlled datastream. This data level is used as input to higher-level data products. Preliminary or not intended for distribution to data users.

b1: quality control checks applied to at least one measurement and stored in an accompanying quality control variable meeting quality control standards listed in this document. The addition of *qc_time* does not force the datastream to b level. External data may contain additional quality control flags specified by the external data source.

b2 to **b9**: further processing on *b1*-level data that does not merit *c*-level classification. Example: additional quality control test or different parameters used in processing. A description of the further process must be described in the netCDF header, instrument handbook, or technical paper available to data users.

c0: intermediate VAP. This data level is used as input to a higher-level VAP. Preliminary or not intended for distribution to data users.

c1: derived or calculated VAP using one or more measured or modeled data as input. For external data, .*c1*-level data may contain gridded model data, satellite data, or other data that have had algorithms applied by an external source. A description of the process must be included in the netCDF header or technical paper available to data users.

c2 to **c9**: further processing applied to a *c1*-level datastream using the same temporal resolution. Possible reasons for increasing levels include better calibration, better coefficients for algorithms, or reprocessing using different averaging resolution algorithms.

s1: summary file consisting of a subset of the parent *b*- or *c*-level file with simplified quality control and "Bad" values set to missing value indicator. The *s*-level number must match the *b*- or *c*-level file used as input.

s2 to **s9**: summary file for higher *c*-level datastreams.

m0: ARM model data intended to make predictions using complex methods that may not be on the same time step or physical gridding as the input datastreams. Not intended for distribution to data users. Initial use is for Large-Eddy Simulation (LES) ARM Symbiotic Simulation and Observation (LASSO) data.

m1 to **m9**: further processing applied to ARM model data intended to make predictions using complex methods that may not be on the same time step or physical gridding as the input datastreams. Initial use is for LASSO data.

Notes:

- Not every data level needs to be produced for each instrument data set. Example: if conversion from RAW to netCDF, calibration, and engineering units are applied in a single processing step during conversion from RAW to netCDF format, then an *a0* data product would not be produced.
- Quality control checks applied to a data variable by the instrument (not by ingest) do not require the data level to be increased from an *a1* level to *b1*, unless the netCDF data file provides accompanying QC variables satisfying *b*-level requirements.
- Data level *c0* to *c9* are restricted to data derived or calculated through value-added processing. Lower-level datastreams will be kept in the ADC if useful for evaluating an instrument or cross-checking another datastream. If the lower-level data does not need to be kept, it will be removed from the ADC.
- For consistency with historical data, image datastreams use the *a1* level, and a single day's worth of images, are typically tarred into a single file for archiving.

5.1.4 Best Estimate

The use of "be" in a filename indicates the datastream is a best estimate. This designation indicates an official decree from the ARM facility that the values used are ARM's best attempt at representing the scientific quantity. Use of the best estimate designation requires approval from ARM leaders through an Engineering Change Request (ECR).

5.1.5 File Duration

To control the number of small files and to facilitate the use of ARM data, the file period for datastreams and typical value-added processes span 24 hours over a UTC day. Datastreams with solar data or statistical products may choose to use a different time period when appropriate.

Very large data sets may be routinely split into two or more netCDF files per day to increase usability or stay within single file size limits. The ADC suggests file sizes under 20 GB, but can manage file sizes up to 8 TB. Be reasonable when choosing file size.

Daily data files are allowed to split when metadata information changes (example: instrument serial number or calibration change). ARM standard processing expects a file to split when a metadata change is detected.

5.2 Guidelines for Original RAW Filenames

The RAW filename created by the instrument is often decided by the instrument vendor. Requesting the vendor to change the filename format is typically not possible and is not a requirement. After the data system retrieves the RAW instrument file, the data system will rename the file to the appropriate ARM standards (i.e., the 00-level data filename).

When possible, the original filename produced on the instrument or instrument data system should contain adequate information to determine the origin of the file including:

- unique site and facility indicator
- yyyymmdd (year, month, day of month) or yyyyjjj (year, day of year)
- hhmmss (hour, minute, second), hhmm (hour, minute), or sequence
- number if more than one raw file per day
- indication of instrument type or vendor.

Often, it is not possible to include all this information. In those instances, it is important to include adequate header information inside the file to permit the user to determine the source/original data and provide a reference date (including year) and time.

5.3 File Naming Conventions for RAW ARM Data

RAW ARM data files to be ingested are named according to the following naming convention:

(sss)(inst)(Fn).00.(yyyymmdd).(hhmmss).raw.(xxxx.zzz)

Where:

00 is the data level. RAW data is the first data file and shall be labeled with the lowest-possible level

raw is the indicator that the file contains RAW data

(xxxx.zzz) is the original raw data filename produced on the instrument

Example raw data filename:
nsamwrC1.00.20021109.140000.raw.20_20021109_140000.dat

This file is from the North Slope of Alaska Barrow site. It contains raw microwave radiometer data for November 9, 2002, for the hour beginning 14:00:00 UTC.

RAW instrument data are recommended to be collected hourly resulting in 24 RAW data files per day. These files are bundled into daily Tape ARchive (TAR) files before archival.

Underscores and dashes are not allowed in the filename left of and including the six-digit time (hhmmss). Underscores can be treated as wildcard characters in some databases. Due to the method of implementation, underscores are allowed to the right of the 6-digit time in the filename. If possible, do not use underscores in the filename.

Occasionally, data files may become corrupt or contain bad data that causes the ingest to fail. To allow the ingest to continue processing, bad data files are moved to a subdirectory named “bad” with the offending raw file renamed with “bad” replacing the “raw” portion of the name. The TAR file containing the “bad” data file is not renamed.

5.4 File Naming Conventions for TAR Bundles

TAR bundles are named according to the following naming convention:

(sss)(inst)(Fn).(dl).(yyyymmdd).(hhmmss).(xxx).(zzz).tar

Where:

(**dl)** is the file level indicator. Must be two characters including lower-case letters or numbers

(yyyymmdd) is the start date from the first data filename within the TAR bundle

(hhmmss) is the start time from the first data filename within the TAR bundle

(zzz) is the optional extension from the original raw data filename, usually the format of the file or an instrument serial number

(xxx) Lower-case characters or numbers used to help describe the contents of the TAR file. An example is “raw” to indicate the contents of the TAR file contain RAW data. Required to be three or fewer characters to accommodate current ADC database.

tar is the TAR bundle file extension

We recommend creating one TAR file for each date.

The example raw file from above is archived in a TAR bundle named:
nsamwrC1.00.20021109.000000.raw.dat.tar

Some RAW data files are not ingested, but are collected and placed in a TAR file. The TAR filename must follow the standards, but the non-ingested data file within the TAR file may have filenames not matching the standards. We recommend that the data files within the TAR file contain enough information to describe the data including location and time.

5.5 File Naming Conventions for Variable Campaign TAR Bundles

Variable Campaign TAR bundles are named according to the following naming convention:

(sss)(yyyy)(FC)X1.i0.(yyyymmdd).000000.tar.(pi-inst).(ident)(<#>of<#>)

Where:

(sss) is the three-letter code for the location of the field campaign

(yyyy) is the year that the field campaign took place or began

(FC) is the abbreviated name of the field campaign

X1.i0 indicates external field campaign principal investigator (PI) data set

(yyyymmdd) is the date the TAR file was sent to the ADC by the field campaign administrator

000000 is the hhmmss field (the hhmmss resolution is not currently in use)

tar is the TAR bundle file extension

(pi-inst) is the name of the PI and the abbreviation for the instrument producing the data

(ident) is an optional additional identifier if more distinction in the pi-inst pair is needed

(<#>of<#>) is an optional identifier for the total number of packets in the PI data set, e.g., "1of3", "2of3", "3of3"

One TAR file is created for each PI data set, unless over 2 GB. If the TAR file is over 2 GB, the TAR file must be split into less than 2 GB units and an extension <#>of<#> is included.

The example raw file from above will be archived in a TAR bundle named:
nsa2004mpaceX1.i0.20060125.000000.tar.tooman-dfcvis.c2.1of4

The length of the TAR filename must be 60 characters or less.

5.6 Other Data Formats

ARM data may be stored in a format other than netCDF for special data sets. The basic naming convention for processed files does not differ, but the final extension changes accordingly:

asc: ASCII data format (other extensions include txt, csv, dat)

hdf: Hierarchical Data Format (HDF) data format (limited to satellite data)

png: Portable Network Graphics (PNG) data format. Recommended for drawings, sketches, and data plots.

jpg: Joint Photographic Expert Group (JPEG) data format. Recommended for photographs.

mpg: Moving Picture Expert Group (MPEG) format. Recommended for movie format.

pdf: For formatted documents and graphics-rich documents, portable document format (PDF) file type is recommended.

Other data formats (e.g., gifs) may also exist.

5.7 Guidelines to Name Quick-Look Plot Filenames

The standard convention for VAP quick-look plot file names created at the ADC is as follows.

datastream.level.date.time.description.extension

Note: The delimiter is a "." (period) except within the description when it is an "_" (underscore). An underscore is currently acceptable to the right of the datastream, (sss)(inst)(qualifier)(temporal)(Fn).(dl), part of the name. Using underscores in the datastream section may cause problems with databases that use underscores as wildcard characters.

Example:

sgp30ebbrE9.b1.20100101.000000.latent_heat_flux.png

5.8 Case-Sensitive File Naming

Data file names are case sensitive. *example.DAT* and *example.dat* may be interpreted as two different names by ingest and bundling routines. Instruments should be consistent in the way the original filenames are assigned, including case.

6.0 Guideline for netCDF File Structure

6.1 Dimensions

6.1.1 Time Dimension

The time dimension is defined as "unlimited" and is the first dimension of a variable using the time dimension. netCDF3 requires the unlimited dimension to be the first dimension in multi-dimensional arrays. This allows proper concatenation of data along the unlimited dimension.

The recommended order of the dimension definitions start with time followed by coordinate dimensions.

We recommend that the number of dimensions used in a single file be as few as possible. Variables consisting of a single data value are defined as scalars unless the Data Object Design (DOD) is used with other instances where multiple values may exist.

Time

Time in processed data files must be increasing and may not repeat. The time variable in any file except RAW cannot have a missing value or NaN. Files failing these requirements will be sent to the instrument mentor or VAP translator for review.

ARM uses the Gregorian calendar in processed data files. Other calendars are allowed with the addition of an attribute describing the CF calendar name, although we do not recommend deviating from the Gregorian calendar. The calendar variable attribute is optional if the calendar used is Gregorian. Note: the

use of a Julian calendar versus a Gregorian calendar may have slight differences since the Gregorian calendar defines one year as 365.242198781 days versus 365.25 days in a Julian calendar.

Time is defined through the use of both a *time* variable, and *base_time* and *time_offset* variables. Historically ARM has used the *base_time* and *time_offset* method. For consistency with historical data and to accommodate the emerging CF standard, both time formats must be declared in the processed netCDF file. Both time formats work by indicating the number of time steps from an initial time.

We recommend starting the time at UTC midnight and indicating this format in the *long_name*. Starting time at midnight allows for easy interpolation of the values (i.e., dividing the time variable by 3600 to convert from seconds to hours).

6.1.2 base_time and time_offset Variables

Time in ARM netCDF files is indicated in Coordinated Universal Time (UTC), and is represented as "seconds since January 1, 1970 00:00:00", also known as epoch time. For example, an epoch time of 1 means "Thursday January 1, 1970 00:00:01 UTC"; an epoch time of 992794875 is "Sunday June 17, 2001 16:21:15 UTC". The default time zone is UTC, but a different time zone may be defined using a timezone offset from UTC.

Time is indicated with the combination of two variables (*base_time, time_offset*) where the result is the number of seconds since epoch time. *base_time* contains a single scalar value stored as a long integer, and *time_offset* contains a time-series of values stored as double precision floating point numbers, one for each time-step in the file. The epoch time for sample index i is given by the value *base_time+ time_offset[i]*. *base_time + time_offset[0]* is the time corresponding to the time stamp in the filename. This method will allow representing time steps down to 1 microsecond within a one-year time interval.

The linking of *base_time* and *time_offset* is indicated with the *ancillary_variables* variable attribute for *time_offset* set to "*base_time*" and *base_time* set to "*time_offset*".

The string attribute of *base_time* is set to the string description of the *base_time* value (i.e., "17-Sep-2012, 23:07:00 GMT").

6.1.3 time Variable

The *time* variable follows CF convention and is recommended to be defined as "seconds since" a Unidata UDUNITS defined time. The default time zone is UTC, but a different time zone may be defined using a time zone offset from UTC. *time* is a "coordinate variable" or a variable with the same name as the time dimension. This enables generic netCDF tools to work with ARM data. (See, for example, the COARDS netCDF conventions linked from cfconventions.org.) Other conventions besides "seconds since" are allowed but not recommended. The use of "months since" and "years since" is not recommended unless explicitly defined.

Example:

dimensions:
time = UNLIMITED ; // (1440 currently)
variables:
int base_time ;
base_time:string = "18-Sep-2012,00:00:00 GMT"; ;
base_time:long_name = "Base time in Epoch" ;
base_time:units = "seconds since 1970-1-1 0:00:00 0:00" ;
base_time:ancillary_variables = "time_offset" ;
double time_offset (time) ;
time_offset:long_name = "Time offset from base_time" ;
time_offset:units = "seconds since 2012-09-18 00:00:00 0:00" ;
time_offset:ancillary_variables = "base_time" ;
time_offset:calendar = "gregorian" ; // Optional attribute when set to gregorian
double time (time) ;
time:long_name = "Time offset from midnight" ;
time:units = "seconds since 2012-09-18 00:00:00 0:00" ;
time:calendar = "gregorian" ; // Optional attribute when set to Gregorian
time:standard_name = "time" ;

6.1.4 Time Boundary

Most data values are reported as a calculation using multiple values over a predefined number of samples. Indicating the boundaries over which values are used and the location of the reported time value within the range is critical to properly understanding the reported data. For all non-instantaneous data, the values of each time range are required. A *bounds* variable attribute indicates the corresponding two-dimensional variable dimensioned by *time* and a bounds dimension containing the boundary values for each averaging period. CF convention does not require a *long_name* attribute for the bound variable, but we recommend adding the attribute. The *units* attribute is not recommended.

A new dimension set to 2 is added to store the start and end time values. This dimension does not require a coordinate variable. The recommended name is *bin*.

The existence of a time bounds variable along with an "ARM-<#>" *Conventions* global attribute indicates all variables dimensioned by *time* are assumed averaged over the time bounds period unless a *cell_methods* variable attribute exists. The method described in the *cell_methods* variable attribute supersedes the *time*-averaged assumption. See *Cell Method Attribute* for further description.

Example:

dimensions:
time = UNLIMITED ; // (1440 currently)
bound = 2 ;
variables:
double time (time) ;
time:long_name = "Time offset from midnight" ;
time:units = "seconds since 2013-01-25 00:00:00 0:00" ;
time:standard_name = "time" ;
time:bounds = "time_bounds";
double time_bounds (time, bound) ;
time_bounds:long_name = "Time cell bounds" ; // Optional
bound_offsets = -30., 30. ; // Optional. Only provide if all periods are the same offsets
float atmos_temperature (time) ;
atmos_temperature:long_name = "Average atmospheric temperature" ;
atmos_temperature:units = "degC" ;
relative_humidity:cell_methods = "time: mean" ;
float relative_humidity (time) ;
relative_humidity:long_name = "Instantaneous relative humidity" ;
relative_humidity:units = "%" ;
relative_humidity:cell_methods = "time: point" ;

The *time_bounds* variable contains the starting and ending time values for each time range. If the *units* attribute is omitted, the values are offset from to the *time:units* time. The individual *time* value indicates where within the range *time* is reported. The *long_name* and *units* attributes are not required for *time_bounds*, and the variable may not have missing values.

The optional *time_bounds:bound_offsets* attribute declares the width of each range. Setting the attribute requires that each range is consistent. If the range is not consistent, the attribute is omitted. The reserved ARM *attribute bound_offsets* is used with ARM Data Integrator (ADI) to auto-generate the bounds variable.

For example, if *time* is defined as the number of seconds since January 25, 2013 00:00:00 UTC:

time = [0., 60., 120., 180., 240., …]
time_bounds = [[-30., 30., 90., 150., 210., …]
[30., 90., 150., 210., 270., …]]

In this example, the first time sample is reported at 0 second added to January 25 2013 00:00:00 UTC. The first time sample is bounded by the start time greater than or equal to January 24, 2013 23:59:30 UTC (subtract 30 seconds), and end time less than January 25, 2013 00:00:30 UTC (add 30 seconds). In this example, the time value relative to the start and end time indicates that the time values are reported at the center of the range. The averaging period is consistent so the optional attribute bound_offsets equals [-30, 30].

6.2 Coordinate Dimensions

If a coordinate dimension is used, then a variable with the same name as the dimension is recommended to be added with the required *long_name* and *units* attributes. Examples of coordinate dimensions are *bin*, *height*, *range*, or *depth*. The name of the dimension should clearly articulate the values. We recommend using singular names and not abbreviations. The *long_name* attribute should be as concise as possible in describing what the values represent. A dimension defined in the netCDF file for the purpose of writing string characters or as an index does not require a corresponding variable.

Example:

dimensions:
time = UNLIMITED ; // (1440 currently)
range = 1999 ;

variables:
float range(range) ;
range:long_name = "Distance from transceiver to center of corresponding bin" ;
range:units = "km"; ;

A coordinate variable may not have a *missing_value*, *_FillValue* or *NaN* value, and must be monotonically increasing or decreasing.

6.2.1 Reference for Coordinate Units

Some coordinate variables use units that require a frame of reference declaration. Examples include the difference between height above ground level (AGL) versus height above mean sea level (MSL). Both of these coordinate variables have units of distance, but are measured from different reference points. The reference point is required to be documented with the CF *standard_name* method. Refer to the CF standard name table for definitions. Declaring the frame of reference in the *long_name* is optional and not recognized as the official method. If the frame of reference is declared in both *standard_name* and *long_name*, the *standard_name* is used.

6.2.2 Referencing AGL and MSL

When referencing above ground level (AGL) use *standard_name = "height"*. *height* is measured above a surface. Over land it refers to ground level, while over the ocean refers to the ocean surface. This is different from above mean sea level (MSL).

When referencing above MSL use *standard_name = "altitude"*. Technically, *altitude* refers to the *mean geoid*, not *mean sea level*. The difference between *mean sea level* and *mean geoid* is small, with the two terms typically used interchangeably (similar to UTC versus Greenwich Mean Time [GMT]). Currently, CF has no standard name for *mean sea level*.

6.2.3 Coordinate Boundary

Data summarized over a range is common in atmospheric data and needs sufficient metadata to describe the ranges. The range of summarized data is described as binning. Typically, binned data is evenly spaced and reported at the center of the range. To report the range of binned values ARM follows the CF conventions. The CF convention uses the *bounds* attribute to indicate the corresponding variable indicating the start and end location of each range with a two-dimensional array. *long_name* and *units* attributes are recommended but not required.

If the range is consistent, the optional *bound_offsets* attribute describes the range of the bin. If the range is not consistent, *bound_offsets* is omitted. The reserved ARM *bound_offsets* attribute is used with ADI to auto-generate the bounds variable.

Example:

dimensions:
time = UNLIMITED ; // (1440 currently)
bin = 21 ; // Dimension name does not need to be "bin" but is recommended
bound = 2 ; // Use of "bound" as dimension name recommended
variables:
float bin(bin) ;
bin:long_name = "Center of droplet size bin"; ;
bin:units = "um" ;
bin:bounds = "bin_bounds" ;
float bin_bounds(bin, bound) ;
bin_bounds:long_name = "Droplet size bin bounds" ; // Optional
bin_bounds:units = "um" ; // Optional
bound_offsets = -5, 5 ; // Optional. Only provide if all steps are the same offsets. Works with ADI to auto generate array.
float ccn_number_concentration(time, bin) ;
ccn_number_concentration:long_name = "AOS Cloud Condensation Nuclei number concentration" ;
ccn_number_concentration:units = "count" ;
ccn_number_concentration:missing_value = -9999.f ;
ccn_number_concentration:cell_methods = "bin: sum" ; // Optional

In this example, the *bin* coordinate variable contains values describing the binned sample and the chosen location within the range. The location within the range is often immaterial and only serves as a placeholder. But knowledge of the location within the range is critical to some studies. The bin range is described in the *bin_bounds* variable with *bin_bounds[i,0]* containing the initial bound (values are greater than or equal to) and *bin_bounds[i,1]* containing the final bound value (values are less than). The *bin[i]* range is bounded by the two *bin_bounds[i,*]* values and its value relative to the two *bin_bounds[i,*]* values indicates where within the bin the value is being reported (i.e., beginning, middle, end). Typically, the reported value is in the center of the range.

This example also uses the optional *cell_methods* attribute to describe the method used. See Cell Method Attribute or CF documentation for explanation of this attribute.

An example of how to indicate changing bin values for each time step is found in Appendix B.

6.2.4 Additional Dimension

An additional dimension may be needed for string arrays, bounds, or other dimensions that are not intended to be used as coordinate variables. The number of additional dimensions should be minimized and named in a clear and concise way to describe their use. Examples include string array or coefficients for equations.

Example:

dimensions:
time = UNLIMITED ; // (1440 currently)
string_length = 13 ;

char status_string (time, string_length) ;
status_string:long_name = "Warning, alarm, and internal status information" ;
status_string:units = "1" ;
status_string:comment = "The values reported by the instrument have the form FEDCBA987654 and contains Alarm (A), Warning (W), and internal status (S) information. Each character is a hexadecimal representation of four bits, i.e., values between 0 and 9 are presented with respective numbers and values 10, 11, 12, 13, 14, and 15 are presented with letters A, B, C, D, E, and F, respectively. "

6.2.5 Cell Method Attribute

The optional *cell_methods* variable attribute describes how the data were derived by indicating the method used. This method is well defined by the CF convention and is extensionable to describing multidimensional data sets. Additional description can be found at cfconventions.org.

The addition of *cell_methods* to a data variable describes how the data were derived to both human and automated software enabling the data to be regridded or analyzed with generic tools.

The format includes the dimension name followed by the method in a *"dimension_name: method"* format. This format allows different methods to be indicated for different dimensions.

Example:

Precipitation Measurements

- Average, maximum, statistics or point value
  - *temperature:cell_methods = "time: mean" ;*
  - *temperature_max:cell_methods = "time: maximum" ;*
  - *temperature_std:cell_methods = "time: standard_deviation" ;*
  - *pressure:cell_methods = "time: point" ;*

To indicate more complicated methods additional information can be included in parentheses after the method.

- Precipitation amount

- *precipitation_rate:cell_methods = “time: sum (interval: 1 min)” ;*
- *precipitation_total:cell_methods = “time: sum (interval: 24 hr comment: summed over one UTC calendar day)” ;*

For multidimensional data, the order indicates the order of operation. In the following example the data is averaged over the time dimension first, and then the median values are calculated for the height dimension. The left-most operation is performed first.

- Averaged over time cells and then median over height cells
  - *temperature:cell_methods = “time: mean height: median” ;*

6.3 Location Variables

The instrument location is described using latitude, longitude, and altitude variables. The required unit of latitude is degrees north, a variable name of *lat*, and *standard_name = “latitude”*. The required unit of longitude is degrees east, a variable name of *lon*, and *standard_name = “longitude”*. The recommended unit of altitude is meters above mean sea level. The required variable name is *alt*, and *standard_name = “altitude”*. The altitude measurement references the altitude of ground level relative to MSL. The instrument height AGL is defined with the *sensor_height* attribute. See Sensor Height section for a full explanation. The use of the specific *lat*, *lon*, and *alt* variable names are required to be consistent with historical data. *lat*, *lon*, and *alt* variables can be vectors for mobile platforms when needed.

Example:

float lat ;
lat:long_name = “North latitude”; ;
lat:units = “degree_N” ;
lat:standard_name = “latitude” ;
lat:valid_min = -90.f ;
lat:valid_max = 90.f ;
float lon ;
lon:long_name = “East longitude” ;
lon:units = “degree_E” ;
lon:standard_name = “longitude” ;
lon:valid_min = -180.f ;
lon:valid_max = 180.f ;
float alt ;
alt:long_name = “Altitude above mean sea level” ;
alt:units = “m” ;
alt:standard_name = “altitude” ;

Notice the use of *valid_min* and *valid_max* attributes with the *lat* and *lon* variables. These are not quality control limits, but limits of acceptable values to be used by the data user. A value outside these ranges should not be used and may be excluded automatically by the process reading the netCDF data file.

6.4 Guidelines for Construction of Variable Names

A variable name should convey a basic understanding of the associated data. File space is not an issue, and cryptic variable names are typically only understood by the person who originally created the name. ARM variable name guidelines are as follows:

- The first character is required to be a letter character. Only letters, numbers or underscores are allowed per netCDF requirements. Use upper-case letters sparingly.
- The variable name is constructed by joining the names to the qualifiers using underscores (_).
- Variable names are recommended to be concise. One has to be reasonable when picking variable names.
- Abbreviations are recommended only when needed for limiting excessively long variable names, for following previous conventions, or for clarity.
- Variable name lengths are required not to exceed 64 characters to comply with ADC database storage requirements.
- Single-character names are not recommended.
- Common variable names are recommended to use common ARM variable names that follow the standards and are used in other datastreams to promote clarity across datastreams. Review pick list for common variable names.
- Variable names and dimensions are recommended to be singular (i.e., temperature not temperatures)
- Greek letters are not allowed in netCDF3. We recommend not spelling out Greek letters, formula symbols, or units.

Variable names should be as concise as possible. For example, “temperature” is recommended to be fully spelled out unless the full variable name becomes unreasonably long, where “temp” would be the abbreviation. *atmospheric_temperature* is more descriptive of the measurement than *temperature* alone. A variable labeled *temperature* could describe air temperature, instrument temperature, derived temperature, etc.

6.4.1 Variable Names Hierarchy

Name hierarchy is used for variable differentiation within the same file. If a conflict arises, then the following hierarchy is used.

1. [super prefix] For example, qc, aqc, be, source
2. [prefix] For example, interpolated, calibrated, instantaneous
3. [measurement] For example, vapor_pressure, pressure, temperature
4. [subcategory] For example, head, air, upwelling, shortwave, hemisphere
5. [medium] For example, earth, satellite, sea, atmosphere
6. [height/depth] For example, 10m, 2cm, 5km
7. [enumeration] For example, e, w, n, s, a, b, 1, 2
8. [source name] For example, smos, met,
9. [algorithm] For example, fibonacci, wrf
10. [quantity] For example, mean, standard deviation, maximum, summation

Example of variable names using hierarchy:

- *qc_atmospheric_temperature_10m*
- *soil_temperature_swats*
- *wind_speed_5m*
- *relative_humidity*
- *qc_vapor_pressure_aeri_std*
- *rain_rate_attenuation_csapr*
- *source_absorption_coefficient_405nm*
- *qc_log_backscatter_xpol_std*

The creation of a variable name is related to the DOD for which it exists. A variable name should convey the required information to distinguish the different variables, but does not need to completely describe the corresponding data. For example, if a DOD contains data from a single instrument, there is no need to indicate the instrument in the variable name. Or, if every variable in the file is an average, there is no need to indicate average in the variable name.

We recommend that related variable names repeat the same basic pattern for similar variables. This may result in using abbreviations for the basic variable. If the variable was not accompanied by other variables, the abbreviation would not be used. For example, a datastream containing a measurement of aerosol optical thickness with no accompanying variables would use *aerosol_optical_thickness*. If the measurement has accompanying variables extending the variable name length, the variable names then use the same base name, i.e., *aot, aot_1020nm, aot_1020nm_francis_mean_10min, aot_1020nm_francis_mode_10min, aot_1020nm_francis_mean_10min_std*. This method informs the data user that the measurements are correlated.

Use of abbreviations is not recommended to help international data users fully understand the data. However, very long variable names may become unreasonable, so abbreviations may be used. We recommend using abbreviations only when the variable name becomes excessively long (i.e., 25 characters or more). When abbreviations are used, we recommend using the values listed below.

6.4.2 Variable Name Abbreviations and Descriptors

To assist international data users with fully understanding the data, the use of abbreviations is not recommended unless a variable name becomes excessively long (i.e., 25 characters or more). When abbreviations are used, it is recommended to use values listed in this section.

Prefix Qualifier

- inst = instantaneous
- fgp = fraction of good points
- be = best estimate
- qc = quality control
- aqc = ancillary quality control or alternate quality control
- inter = interpolated

Measurement Qualifier

- temp = temperature
- snr = signal to noise ratio
- lat = latitude
- lon = longitude
- alt = altitude
- navg = number of points averaged
- aod = aerosol optical depth
- aot = aerosol optical thickness (aod is preferred to aot)
- precip = precipitation
- rh = relative humidity
- wspd = wind speed
- wdir = wind direction

Subcategory Qualifier

- low = lower
- high = higher
- up = upwelling or coming from below
- down = downwelling or coming from above
- long = longwave
- short = shortwave
- pol = polarization
- hemisp = hemispheric
- ref = reference
- ir = infrared
- vis = visible
- uv = ultraviolet
- coef = coefficient
- scat = scattering
- aux = auxiliary
- rot = rotational
- copol = co-polarization
- xpol = cross-polarization
- depol = depolarization
- diff = delta or difference
- anc = ancillary

Quantity Qualifier

- std = standard deviation
- mean = arithmetic mean
- avg = arithmetic average (mean is preferable to average when the two are used interchangeably)
- mode = arithmetic mode

- med = arithmetic median
- var = variance
- sum = summation
- min = minimum
- max = maximum
- stderr = standard error
- log = logarithm
- ln = natural logarithm

6.5 State Indicator Variable

Some variables are intended to indicate a particular state of the instrument or a flag indicating some correlating event (i.e., hatch status of open or closed, detection of cloud, instrument cycling through a series of calibrations). This variable is typically metadata rather than data and separate from quality control information. A user may choose not to implement quality control information, but typically always needs to implement state information. We recommend that indication of a state follow CF convention formatting with the following format.

Two slightly different formatting methods are available with the choice of method depending on two criteria:

- Are the flags mutually exclusive?
- Is it possible for more than one state to exist simultaneously?

6.5.1 Exclusive States

Data type is byte, short integer, or long integer. Definition of all possible states and description of the states are described using the CF-defined *flag_values* and *flag_meanings* variable attributes. The different flag meanings are strings separated by a single-space character. Individual flag meanings may not contain spaces and consists of words connected with underscores. A more detailed description of the state may be made through an optional *flag_<#>_description* attribute.

Flag numbers are required to be greater than or equal to zero if the optional *flag_<#>_description* attributes are used. Negative flag numbers listed in the *flag_<#>_description* may cause issues with methods used for reading data. Some interpreted languages implementations may convert attribute names to program variables. A "-" character is not allowed in most programming language variable names.

int hatch_status (time) ;
hatch_status:long_name = "Hatch status" ;
hatch_status:units = "1" ;
hatch_status:missing_value = -9999 ;
hatch_status:flag_values = 0, 1, 2 ; // Array of values
hatch_status:flag_meanings = "hatch_open hatch_closed in_transition" ;
hatch_status:flag_0_description = "Hatch is open" ; // Optional
hatch_status:flag_1_description = "Hatch is closed" ; // Optional
hatch_status:flag_2_description = "Hatch is in transitional state" ; // Optional

6.5.2 Inclusive States

Data type is byte, short integer, or long integer. Definition of the possible states and description of the states are described using the CF *flag_masks* and *flag_meanings* variable attributes. The existence of the *flag_masks* attribute indicates bit-packed values. The *flag_masks* attribute declares the bit mask values to repeatedly use with a bit-wise AND operator to search for matching enumerated values. The values of *flag_masks* will always be powers of two. A more detailed description of the state may be made through optional *bit_<#>_description* attributes. We recommend listing all possible states to ensure automated software always has a state to match to a value. To accommodate historical data, a value of 0 is allowed to indicate none of the states as long as a comment attribute describes the state of no bits set.

int sensor_status(time) ;
sensor_status:long_name = "Sensor Status" ;
sensor_status:missing_value = -9999 ;
sensor_status:flag_masks = 1, 2, 4, 8, 16 ; // Array of values
sensor_status:flag_meanings = "low_battery hardware_fault offline_mode calibration_mode maintenance_mode" ;
sensor_status:bit_1_description = "Low battery" ; // Optional
sensor_status:bit_2_description = "Hardware fault" ; // Optional
sensor_status:bit_3_description = "Offline mode" ; // Optional
sensor_status:bit_4_description = "Instrument performing calibration" ; // Optional
sensor_status:bit_5_description = "Instrument in maintenance mode" ; // Optional

To detect which bits have been set, repeatedly use bit-wise AND the variable values with each *flag_mask* element to search for matching values. When a result is equal to the corresponding *flag_masks* element, that condition is true. For example, if the state value is 6, its binary representation is 00000110, so the second and third bits are set. Recursively using bitwise AND for each *flag_masks* value ([1, 2, 4, 8, 16]) has five results [0, 2, 4, 0, 0] indicating only the second and third flags have been set; hardware_fault and offline_mode.

6.6 Variable Attributes

In general, the variable attribute names are lowercase. Words are separated by an underscore. A single lengthy comment attribute is preferred to multiple comment attributes (i.e., use *comment* or *comment_on_noise* and *comment_on_resolution* instead of *comment_<#>*).

6.6.1 Required Variable Attributes

- *long_name*: Must be unique in regard to the other variables in the same netCDF file. Be as clear and concise as possible. (As a guideline, think about displaying this value on a plot presented at a conference.) First letter of the *long_name* attribute value is recommended to be capitalized. Long names may not change without a DOD version change. This is to ensure correct matching with searches in the Data Discovery Tool, information about the measurement is in the correct attribute, and how ARM defines a unique DOD.

- *units*: See current list of recommended unit descriptors in Appendix C. Must comply with current Unidata *udunits* unless units descriptor is currently not listed. The unit must be scientifically correct, correctly understood by the user, and work with automated codes. **"unitless" is represented with a "1"**. Units are represented as a multiplier to the data resulting in a data * units concept. For example, units of *backscatter* in CEIL are "1/(sr*km*10000)". For a backscatter value of 100.0, the backscatter value after multiplication is 0.01/(sr*km).

6.6.2 Required with Conditions

- *missing_value* or *_FillValue*: If the data variable uses a single specific value to represent no data, a *missing_value* or *_FillValue* variable attribute must be declared. There is no required value but the recommended value is - 9999. Do not include *missing_value* with coordinate variables. The value must be the same type as the corresponding data values. The value is recommended to be outside the valid data range. If *missing_value* and *_FillValue* are both used, we recommend using the same value for both *_FillValue* and *missing_value*. If more than one value is used to represent missing value or non-valid data, use *valid_range* CF attribute to define range of data and use a non-valid data value outside of the *valid_range*. See CF convention for further description.
- *standard_name*: Required if a primary variable and the standard name exists in the CF table.

6.6.3 missing_value versus _FillValue Discussion

Historically ARM has used the *missing_value* attribute to indicate the value used to indicate a missing data value. CF convention has transitioned from the use of *missing_value* and is suggesting the use of the *_FillValue* attribute. The *NCO toolkit* has also transitioned from the use of *missing_value* to *_FillValue*. To correctly use *NCO* tools, the missing value must be indicated in the *_FillValue* attribute, or the data file must correctly use *valid_min* and *valid_max* or *valid_range*.

When a netCDF file is initially created, all data values are set to netCDF *_FillValue*. If the *_FillValue* attribute is defined, that value is used as the fill values. If no *_FillValue* attribute is defined, a default value is used differing by data type. During the write state, the fill values are changed to the written values. Therefore, if the default netCDF *_FillValue* value exists in the netCDF file but a *_FillValue* attribute was not defined, something has gone wrong during the writing process.

A *missing_value* is the value used to indicate no data and has been introduced into the data by the writing software. If *_FillValue* is not defined or defined as a value different from *missing_value*, there may be two different values indicating non-data values. This will require a user to mask the *missing_value*, and default fill value or *_FillValue* from the analysis. To reduce the work required by the data users, we recommend using the same value for *_FillValue* and *missing_value*.

6.6.4 standard_name Attribute

When possible, we strongly recommend including a CF *standard_name* attribute to officially define the data. Official string values for the *standard_name* attribute must be taken from the CF standard name table. Creating a new string value when a standard name does not exist is not allowed.

Link to table: http://cfconventions.org/standard-names.html

Example:

float sea_level_pressure(time) ;
sea_level_pressure:long_name = “Mean sea level pressure” ;
sea_level_pressure:units = “hPa”; ;
sea_level_pressure:missing_value = -9999.f
sea_level_pressure:standard_name = “air_pressure_at_sea_level ;
sea_level_pressure:ancillary_variable = “qc_sea_level_pressure
instrument_status” ;

float qc_sea_level_pressure(time) ;
qc_sea_level_pressure:long_name = “Quality check results on variable: Mean sea level pressure” ;
qc_sea_level_pressure:units = “1” ;
qc_sea_level_pressure:flag_method = “bit” ;
qc_sea_level_pressure:fail_min = 900.f ;
qc_sea_level_pressure:fail_max = 1200.f ;
qc_sea_level_pressure:bit_1_description = “Value less than fail_min” ;
qc_sea_level_pressure:bit_1_assessment = “Bad” ;
qc_sea_level_pressure:bit_2_description = “Value greater than fail_max” ;
qc_sea_level_pressure:bit_2_assessment = “Bad” ;

short instrument_status(time) ;
instrument_status:long_name = “Instrument status” ;
instrument_status:units = “1” ;
instrument_status:missing_value = -9999 ;
instrument_status:flag_values = 0, 1 ;
instrument_status:flag_meanings = “online offline” ;

6.6.5 ARM Standard Variable Attribute Names

- resolution
- comment
- comment_<#> (used for multiple distinct comments within a single variable)
- precision
- accuracy
- bit_<#>_description (for inclusive, bit-based flags)
- flag_<#>_description (for exclusive, state-based flags)
- bit_<#>_assessment (for inclusive, bit-based flags)
- flag_<#>_assessment (for exclusive, state-based flags)
- corrections_applied (set to space delimited list of keywords)
- sensor_height
- wavelength
- actual_wavelength
- filter_wavelength
- source

6.6.6 Sensor Height Attribute

If the declaration of the height of an instrument above a surface is desired, it is declared with an optional *sensor_height* attribute. If all sensors are at the same height for a datastream, a global attribute may be used. If different variables represent data at different heights, each variable indicates the sensor height with the *sensor_height* attribute. The presence of a *sensor_height* variable attribute supersedes the global attribute. To determine the height of the sensor above MSL, add *sensor_height* value to *alt* variable value.

The *sensor_height* attribute format is: numerical value, CF udunit compliant unit, "AGL" all separated with a single-space character. A negative value represents a measurement below ground level. The value is the height of the sensor AGL or, in the case of above water, above the surface.

For indicating the reference of height measurements, for example, AGL versus above MSL, see the *Reference for Coordinate Units* section.

Example:

float wind_speed (time) ;
wind_speed:long_name = "Mean wind speed" ;
wind_speed:units = "m/s" ;
wind_speed;missing_value = -9999.f ;
wind_speed:sensor_height = "10.5 m AGL" ;

6.6.7 Attribute Datatype

Variable attributes set to a numeric value intended for direct use with the variable values must match the same type as defined for the corresponding variable type. (e.g., *_FillValue, missing_value, valid_min, valid_max, valid_range*) Numeric variable attributes used for other processes that work with the variable indirectly should be set to the appropriate type.

Example:

double wind_direction *(time) ;*
wind_direction:long_name = "Mean wind direction" ;
wind_direction:units = "degree" ;
wind_direction:missing_value = -9999. ; // Value is set as double type to match variable
wind_direction:_DeflateLevel = 1 ; // Used for compression not directly with values and netCDF requires set to integer
wind_direction:_ChunkSizes = 300 ; // Used for compression not directly with values and netCDF requires set to integer
wind_direction:_Shuffle = "true" ; // Used for netCDF compression not directly with values so set to character

int qc_wind_direction *(time) ;*
qc_wind_direction:long_name = "Quality check results on variable: Mean wind direction" ;
qc_wind_direction:units = "1" ;

qc_wind_direction:missing_value = -9999 ; // Value is set as int type to match qc_wind_direction type
qc_wind_direction:fail_min = 0.f ; // Value is set as int type to match wind_direction type
qc_wind_direction:fail_max = 0.f ; // Value is set as int type to match wind_direction type
qc_wind_direction:bit_1_description = "Value less than fail_min" ;
qc_wind_direction:bit_1_assessment = "Bad" ;
qc_wind_direction:bit_2_description = "Value greater than fail_max" ;
qc_wind_direction:bit_2_assessment = "Bad" ;

In the next example, data are compressed using packing and the data type changed from a float to short. Since the data are written to the netCDF file as a short, the corresponding *missing_value* attribute must have the same data type. The *scale_factor* and *add_offset* are used in the conversion from packed short to float so those values must be written to the netCDF file as floats to indicate the unpacked type.

Example:

short wind_direction (time) ;
wind_direction:long_name = "Mean wind direction" ;
wind_direction:units = "degree" ;
wind_direction:scale_factor = 0.5f ; // Set to the type values should be after unpacking
wind_direction:add_offset = -100000.f ; // Set to the type values should be after unpacking
wind_direction:missing_value = 45001 ; // Value is set as type short with scale_factor and add_offset packing applied. The actual missing value is - 9999

6.7 Global Attributes

All global attributes must have a value. If a value is unknown at the time of file creation, the attribute must clearly indicate that no known value exists. A standard value of "unknown" or -9999 set to the proper data type is recommended (127 for type byte). Recommended attributes may be omitted if the value is expected to be unknown. If required attributes must be written, but a value is not expected to exist, the use of "N/A" is recommended.

6.7.1 Required and Recommended Global Attributes

The order of global attributes is not a requirement, but we recommend the order listed in this document.

(Required global attributes are **bold-underline**)

Digital Object Identifier

A digital object identifier (DOI) is required in all baseline and evaluation datastreams, with two exceptions. A DOI should be requested for each instrument code plus data level and listed under the required doi global attribute.

Exceptions:

Radar datastreams are allowed to request the DOI using instrument class plus data level or only instrument class without the data level. For example, XSAPRCFR2 datastreams use doi:10.5439/1467902, which does not include scan strategy or other datastream class code information. The same DOI number will be used for multiple datastreams.

LASSO is allowed to use a single DOI for each Alpha data product release. For example, all LASSO Alpha1 data products use doi:10.5439/1256454.

command_line

definition: Records command line used to run the ingest or VAP. If the command is run multiple times to generate the individual file, list the command used to generate the initial file. If a single command line is not used to generate the file, list necessary parameters to set for creating the file.
example: command_line = “langley -d 20130116 -p mfrsr -f sgp.E13” ; ;
formerly: Command_Line

command_line_comment:

definition: Records the exceptional switches used in the command line.
example: command_line_comment = “-D updates the glue database file, -C will process only the data below ~18km” ;

Conventions

definition: The ARM Convention version plus any Conventions that the file conforms to. The ARM Convention indicator consists of “ARM” prepended to the standards document version number joined with a hyphen (-). It is recommended to list ARM Convention first in the list. This is a CF attribute as well.
example: Conventions=”ARM-1.0 Cf/Radial-1.6 instrument_parameters radar_parameters radar_calibration” ;
reference hyperlink: http://www.unidata.ucar.edu/software/netcdf/docs/netcdf.html#Attribute-Conventions

process version

definition: Records the version of the ingest or VAP running on production
example: process_version = “ingest-met-4.10-0.el5” ;
formerly: software_version, Version

dod version

definition: Records version of the ARM DOD represented in this file.
example: dod_version = ”met-b1-2.0” ;

input_datastreams: (VAP or ingest reading ARM datastream only; required with conditions)

definition: Records the itemized list of input datastreams available at runtime, process versions, and filename date ranges. May be omitted if source attribute or source variables are used to describe input datastreams. The datastream, version, and date range are separated by a space-colon-space (“ : “). The individual datastream entries are separated by a space, semicolon, newline, space (“ ;\n ”). If multiple files exist for a single date, but not all files are used, the

individual ranges used should be itemized as separate entries. The separator between dates in a given date-time range is a hyphen ("yyyymmdd.hhmmss-yyyymmdd.hhmmss"). If the time period spans a single date, then no hyphen or end date should be included and the date range is a single date-time ("yyyymmdd.hhmmss").
example: input_datastreams = "sgpsondewnpnC1.a1 : 6.1 : 20010208.232700-20010210.053400 ;\n sgpmwrlosC1.b1 : 1.17 : 20010209.000000 ;\n sgp1twrmrC1.c1: Release_1_4 : 20010209.000000 ;\n sgparscl1clothC1.c1 : Release_2_9 : 20010209.000000" ;

input_source: (ingest only; required only if reading RAW data)
definition: Records the name of the first RAW file with a full path used to create a daily netCDF file. If more than one initial RAW file is used, list the file most useful to describe the ingest process.
example: input_source = "/data/collection/sgp/sgpswatsE10.00/1167508800.icm" ;

site_id:
definition: Three-letter site designation
example: site = "sgp" ;
reference hyperlink: http://www.arm.gov/sites

platform_id:
definition: Instrument description including descriptive and temporal qualifiers
example: "mfrsraod1mich"
reference hyperlink: http://www.arm.gov/instruments

facility_id:
definition: Facility identifier
example: facility_id = "E10" ;
reference hyperlink: http://www.arm.gov/sites

data_level:
definition: Records data level
example: data_level = "a1" ;
formerly: proc_level
reference hyperlink: http://www.arm.gov/data/docs/plan

location_description:
definition: Description of location. The location description consists of the geographical region for fixed locations or campaign name for mobile facility experiments followed by the closest city or town. The geographical region or campaign name should be spelled out followed by the appropriate acronym in parentheses.
example 1: location_description="Southern Great Plains (SGP), Lamont, Oklahoma" ;
example 2: location_description="Storm Peak Lab Cloud Property Validation Experiment (STORMVEX), Christie Peak, Steamboat Springs, Colorado" ;

datastream:

definition: Datastream identifier. This will equal site_id + platform_id + facility_id+ "." + data_level
example: datastream = "sgpmfrsrE32.b1" ;

serial number: (ingest only, required with stipulation)

definition: Records serial number of instrument(s) used to collect data. Only required if the serial number is expected to be known at runtime and is capable of changing. If multiple instruments exist, then specify instrument: otherwise, only provide serial number. Individual serial number entries are separated by a space-semicolon-new line-space (" ;\n "). Instrument descriptors are separated from the serial number with a colon-space (": ").
Type is recommended to be character.
example 1: serial_number = "54321DT" ;
example 2: serial_number = "PIR1-DIR: 31312F3 ;\n PIR2-DIR: 30167F3 ;\n Diffuse PSP: 33271F3 ;\n NIP: 31876E6 ;\n PSP-DS: 33703F3 ;\n SKY-IR: 1845" ;

sampling_interval:

definition: Records expected sampling interval. If the instrument sampling interval is different, it should be noted in the instrument documentation. Format is interval time and compliant udunit descriptor separated by a single-space character.
example: sampling_interval = "400 us" ;
formerly: sample_int

averaging_interval:

definition: Records expected averaging interval. This is in addition to the time bound method of describing averaging interval.
example: averaging_interval = "5 minute" ;

sensor_height:

definition: Records height of all sensors AGL. If multiple sensors at different heights exist, use variable-level attributes. See Sensor Height section for format details. If *sensor_height* is defined at variable-level for all relevant variables, a global attribute should not be defined.
example: sensor_height = "10 m AGL" ;
formerly: sensor_location

title:

definition: A succinct English language description of what is in the data set. The value would be similar to a publication title.
example: "Atmospheric Radiation Measurement (ARM) program Best
Estimate cloud and radiation measurements (ARMBECLDRAD)" ;

institution:

definition: Specifies where the original data was produced. If provided the value exactly matches the value listed here. Exceptions will be allowed on a case-by-case basis.
value: "United States Department of Energy - Atmospheric Radiation Measurement (ARM) program"

description:

definition: Longer English language description of the data
example: “ARM best estimate hourly averaged QC controlled product, derived from ARM observational Value-Added Product data: ARSCL, MWRRET, QCRAD, TSI, and satellite; see source_* for the names of original files used in calculation of this product” ;

references:

definition: Published or web-based references that describe the data or methods used to produce them.
example: “http://www.arm.gov/data/vaps/armbe/armbecldrad”

doi:

definition: Digital Object Identifier (DOI) number used to reference the data.
Request a new DOI through DOI tile in i.arm.gov
example: “10.5439/1039926” ; // Note this is a character string
Note: A DOI references the datastream class plus level

doi_url:

definition: Full Uniform Resource Locator including Digital Object Identifier numbers. Request a new DOI at https://www.archive.arm.gov/armdoi
example: “http://dx.doi.org/10.5439/1039926” ;

history:

definition: Records the user name, machine name, and the date in CF *udunit* or ISO 8601 format. If the file is modified, the original value is retained and new information is appended to the attribute value with statements separated by a space, semicolon, newline, space (“ ;\n “). Strongly recommended to be the last global attribute.
example: history = “created by user dsmgr on machine ruby at 1-Jan-2007,2:43:02” ;

6.8 Quality Control Parallel Variables

In addition to the data variables and state variables, optional quality control (QC) variables may be added to store relevant information about the quality of a data sample. To encourage consistency among ARM data products, ingested data and VAP data files will use the same QC standards. QC variables may use an integer value method for single-value test results, or a bit-packing method for multiple-value test results. The decision of which method to use is left to the developer/mentor/translator. For historical reasons ARM standards do not follow the same CF format for state variables. The CF state variable method may be added for interoperability.

6.8.1 Bit-Packed Numbering Discussion

QC variables may use a bit-packed technique to allow multiple pieces of information to be stored in one numerical value. A more in-depth discussion of the technique can be found at:

https://i.arm.gov/content/arm-bit-packed-qc

6.8.2 Standard Bit-Packed Quality Control Variables

The QC variable has the same name as the data variable with the addition of a "qc" prepended to the variable name joined with an underscore. Example: *qc_temperature*

The *flag_method* = "bit" variable attribute indicates the values are bit-packed.

QC variables are type integer (recommend 32-bit integer), unless appreciated to a higher precision to accommodate more tests than the integer resolution can accommodate.

The QC variable is linked to the data variable with the declaration of an *ancillary_variables* data variable attribute with the value equal to the QC variable name. Multiple ancillary variables may be listed separated by a single-space character.

Required attribute for data variable:

- *ancillary_variables* = the corresponding QC variable name(s)

Required attributes for QC variable:

- *long_name = "Quality check results on variable: <variable's long_name attribute value>" ;*
- *units = "1" ;*
- *description = "This variable contains bit packed integer values, where each bit represents a QC test on the data. Non-zero bits indicate the QC condition given in the description for those bits; a value of 0 (no bits set) indicates the data has not failed any QC tests." ;*
- *flag_method = "bit" ;*

Recommended attribute for QC variable:

- *standard_name = "quality_flag"*

CF has added a standard name of "quality_flag" that can be used to define the ancillary quality control variable using the standard CF method of using *standard_name*. This standard name is a subset of *status_flag*. At this time *quality_flag* is only a recommendation.

Attributes describing the QC tests may be defined at either the variable or global level. A mixture of variable- or global-level definitions is allowed in the same file, but definitions may only occur in one location for a single variable (global level or variable level). Variable-level definitions have priority over global definitions. If the definition of QC bits are explained in the global attributes, a description attribute must point the user to the global attributes for QC bit descriptions.

6.8.3 Variable-Level Bit Description

The following variable attributes are required to describe a QC test at the variable level:

- *bit_<#>_description = "<General description of QC test>" ;*
- *bit_<#>_assessment = <state> ;*

Options for bit_<#>_assessment <state> are "**Bad**" or "**Indeterminate**" only.

The following variable attributes are optional:

- *bit_descriptions*
- *comment*
- *bit_<#>_comment*

Each <#> indicates the bit number. Examples include:

Bit	Variable Attribute	Binary	Hex	Power	Bit-Packed Integer
1	bit_1_assessment	00000001	0x01	2^0	1
2	bit_2_assessment	00000010	0x02	2^1	2
3	bit_3_assessment	00000100	0x04	2^2	4
4	bit_4_assessment	00001000	0x08	2^3	8
5	bit_5-assessment	00010000	0x10	2^4	16

6.8.4 Standard ARM QC

Standard ARM QC is defined as the missing, minimum, and maximum checks performed on a data variable.

Standard ARM QC bits use this format when defined as variable attributes. The bit numbers used for each test are not required but are recommended. The assessment of the minimum or maximum test may be set to a value of "Indeterminate" if more appropriate.

- *bit_1_description = "Value is equal to missing_value" ;*
- *bit_1_assessment = "Bad" ;*
- *bit_2_description = "Value is less than the fail_min" ;*
- *bit_2_assessment = "Bad" ;*
- *bit_3_description = "Value is greater than the fail_min" ;*
- *bit_3_assessment = "Bad" ;*

For a variable-level attribute bit declaration, the existence of a bit declaration indicates the test could have been performed. If a bit is not defined, that bit is free.

6.8.5 Unused ARM QC Bit

When an individual bit is unused, but must be declared, the variable *bit_<#>_description* attribute is assigned the value "Not used", and the *bit_<#>_assessment* attribute assigned the value of "Bad". An optional explanation as to why the bit is reserved may be included in a separate *bit_<#>_comment* variable.

Required attributes:

- *bit_<#>_description = "Not used" ;*
- *bit_<#>_assessment = "Bad" ;*

Optional attribute:

- *bit_<#>_comment = statement describing why the bit is reserved*

6.8.6 Reporting Test Parameters in Description

Equation or limit parameters used in test analysis may be directly listed in the bit description or referenced by a variable attribute name. The bit description must not change between DOD versions. If a parameter value might change, we recommend phrasing the description in a generic manner, referencing an external source, or using a referenced variable attribute.

When a test references another variable in the same file, we recommend that the variable name be listed in the *bit_<#>_description* attribute to provide direct linkage.

Example:

float upwelling_broadband (time) ;
upwelling_broadband:long_name = "Upwelling broadband radiation" ;
upwelling_broadband:units = "W/m^2" ;
upwelling_broadband:missing_value = -9999.f;
upwelling_broadband:ancillary_variables = "qc_upwelling_broadband" ;
int qc_upwelling_broadband (time) ;
qc_upwelling_broadband:long_name = "Quality check results on variable:
upwelling broadband radiation" ;
qc_upwelling_broadband:units = "1" ;
qc_upwelling_broadband:flag_method = "bit" ;
qc_upwelling_broadband:standard_name = "quality_flag" ; // Optional
qc_upwelling_broadband:test_parameter_value = 0.03f ;
qc_upwelling_broadband:bit_1_description =
"mfr10m_cosine_solar_zenith_angle is less than 0.15" ;
qc_upwelling_broadband:bit_1_assessment = "Bad" ;
qc_upwelling_broadband:bit_2_description = "Percent difference is greater than
test_parameter_value" ;
qc_upwelling_broadband:bit_2_assessment = "Bad" ;
qc_upwelling_broadband:bit_3_description = "Value greater than 2 standard deviations of
historical mean" ;
qc_upwelling_broadband:bit_3_assessment = "Bad" ;

In this example, the *test_parameter_value* QC variable attribute is allowed to change without a DOD change to accommodate a varying test limit for the QC test represented by bit 2. There is no requirement for the name of the attribute, but the name should clearly reflect that the value is a QC test parameter. The test limit in bit 1 is not allowed to change without a DOD change because the description attribute would change. Test parameter values should be listed with the QC variable.

6.8.7 Bit-Packed Global Attribute Declaration for Quality Control

The description of each test may be listed in the global attribute section if multiple variables use the exact same bit and description. The *description* variable attribute must exist indicating that the test descriptions are listed in the global attributes. The attribute name follows the same format as the variable-level style except for a prepended "qc_". The prepending "qc_" to the bit description and assessment is to continue with historical format currently in the ADC.

Required QC variable attribute for global attribute bit declaration:

- *description = "See global attributes for individual QC bit descriptions." ;*

Example:

// global attributes:
qc_bit_1_description = "Value is equal to missing_value" ;
qc_bit_1_assessment = "Bad" ;
qc_bit_2_description = "Value is less than the fail_min" ;
qc_bit_2_assessment = "Bad" ;
qc_bit_3_description = "Value is greater than the fail_max" ;
qc_bit_3_assessment = "Bad" ;
qc_bit_comment = "The QC variable values are a bit-packed representation of true/false values for the tests that may have been performed. A QC value of zero means that none of the tests performed on the value failed."

6.8.8 valid_min/valid_max/valid_range Attribute Discussion

CF convention clearly states that *valid_min*, *valid_max*, and *valid_range* are to be used in conjunction with *_FillValue* or *missing_value* to define the **valid** values. By definition, a non-valid value should be removed from analysis. Historically, ARM used *valid_min* and *valid_max* as QC limits to suggest if a value should be used. Therefore, CF and ARM are in conflict. Use of third-party software may have unintended consequences resulting in valid data being removed from the analysis, or if *_FillValue* is not used, the missing value indicator indicated with the variable attribute *missing_value* may be used in calculations. If the use of *valid_min* and *valid_max* is desired, the values must be chosen carefully. The *valid_min* and *valid_max* attribute values are not intended to be used as QC limits. **The use of a different attribute name is strongly recommended for listing QC limit values.**

ADI has been updated to use *fail_min*, *fail_max*, *warn_min*, and *warn_max* variable attributes listed under the quality control variable instead of *valid_min* and *valid_max* variable attributes listed under the data variable (EWO0022253). *fail_min* and *fail_max* should be used with test assessment set to "Bad" while *warn_min* and *warn_max* should be used with test assessment set to "Indeterminate". Unless there is a specific reason to use *valid_min* and *valid_max* for QC limits, use *fail_min* and *fail_max* under quality control variable attributes.

Example:

float precipitative_water_vapor (time, height) ;
precipitative_water_vapor:long_name = “Best estimate of precipitative water vapor” ;
precipitative_water_vapor:units = “mm” ;
precipitative_water_vapor:missing_value = -9999.f ;
precipitative_water_vapor:valid_min = -1000.f ; // This is not used with the QC variable
precipitative_water_vapor:flag_values = [-8888, -7777] ; // this is a vector
precipitative_water_vapor:flag_meanings = “no_input_value_available really_goofy_result” ; // This is not a vector
precipitative_water_vapor:ancillary_variable = “qc_precipitative_water_vapor” ;
int qc_precipitative_water_vapor (time, height) ;
qc_precipitative_water_vapor:long_name = “Quality check results on variable:Best estimate of precipitative water vapor” ;
qc_precipitative_water_vapor:units = “1” ;
qc_precipitative_water_vapor:flag_method = “bit” ;
qc_precipitative_water_vapor:fail_min = 0.f ;
qc_precipitative_water_vapor:warn_max = 30.f ;
qc_precipitative_water_vapor:bit_1_description =”Value is equal to missing_value or other indicator value” ;
qc_precipitative_water_vapor:bit_1_assessment = “Bad” ;
qc_precipitative_water_vapor:bit_2_description = “Value is less than fail_min” ;
qc_precipitative_water_vapor:bit_2_assessment = “Bad”; ;
qc_precipitative_water_vapor:bit_3_description=“Value is greater than warn_max” ;
qc_precipitative_water_vapor:bit_3_assessment = “Indeterminate” ;

Many historical and some current ARM algorithms mix state and/or quality control information with the observed or derived data values. This practice is not recommended, but to accommodate these products correct usage of *valid_min*, *valid_max*, and *valid_range* must be followed.

In the example above, a *missing_value* attribute indicates -9999 as the value used to indicate when a value is simply missing. In addition to the missing value indicator, two additional state values defined by *flag_values* are used, -8888 and - 7777. These values indicate the state of the data value at that time-height step and should not be used blindly in scientific analysis such as incorporating them into a calculation of the mean precipitative water vapor. To correctly remove these values from analysis, the *flag_values* must be less than *valid_min*. This will allow software to automatically remove the values from the data set. The *valid_min* value (-1000) in this example is significantly lower than a quality control lower limit (0), but is still greater than the missing value indicator and two state indicator values: -9999, -8888, -7777. This allows a user to only read the *precipitative_water_vapor* variable and still use the data correctly without using the associated quality control variable.

Data users should be aware that the two state values could inadvertently be used in a calculation if they choose not to apply the quality control variable or to not use *valid_min* as an absolute lower limit for valid data. Some ARM VAPs have used this mix of state and quality control information with the data values, and this has the potential to create issues that are very difficult to track. As referenced above in the *precipitative_water_vapor* example, blindly using a mix of the state, quality control, and valid data values in a scientific calculation will likely have unintended consequences such as large-magnitude negative numbers in the calculation of average values.

If the *valid_min* value is provided as a quality control value rather than the suggested absolute lower limit for valid data, some users may choose not to filter data less than the *valid_min* for legitimate scientific reasons. For example, quality control limits can be set incorrectly or can change over time as additional information about the atmosphere or instrument operation are gleaned. In these cases a data user must ignore the quality control limits since valid data exist outside the limits. Using the precipitative water vapor example above, the two state values must still be removed from the analysis if *valid_min* is used as a quality control value and not used for data filtering. This case emphasizes why it is not suggested to use *valid_min* as a quality control value as many automated software tools will only correctly remove the state values if the valid range of data and quality limits of data are defined independently.

As ARM data files grow in size, the use of compressing techniques will become necessary for reasonable file sizes. CF recognizes two types of data compression; packing and compression. Compression uses techniques with no data precision loss using standard utilities like UNIX compress or GNU gzip. The packing technique uses scaling and offset values that will compress the data with precision loss. For this type of compression CF suggests "*When data to be packed contains missing values the attributes that indicate missing values (_FillValue, valid_min, valid_max, valid_range, [missing_value]) must be of the same data type as the packed data*." In this case CF uses the *valid_min* and *valid_max* attributes as absolute valid data limits and will exclude data from being uncompressed. Therefore, the data are being excluded from analysis before being uncompressed and there is no way for a user to examine the data for quality control purposes. The data need to be excluded before uncompressing as CF states "*Note that values that are identified as missing should not be transformed. Since the missing value is outside the valid range it is possible that applying a transformation to it could result in an invalid operation*."

6.8.9 Multiple Variable Summarized Quality Control

It is optional to summarize quality control for multiple data variables in a single QC variable. Multiple data variables may use the same QC variable with a small change to the QC variable. The previously declared QC standards apply to multi-variable QC variables.

Requirements for Multiple-Variable QC variable:

- QC variable name is prepended with "qc_" and the base name not match any existing data variable name
- *long_name = "Quality check results" ;*

Example:

float signal_return_copol(time, height) ;
signal_return_copol:long_name = "Attenuated backscatter, co-polarization" ;
signal_return_copol:units = "counts/microsecond" ;
signal_return_copol:missing_value = -9999.f ;
signal_return_copol:ancillary_variables = "qc_signal_return" ;
float signal_return_xpol(time, height) ;
signal_return_xpol:long_name = "Attenuated backscatter, cross-polarization" ;
signal_return_xpol:units = "counts/microsecond" ;
signal_return_xpol:missing_value = -9999.f ;
signal_return_xpol:ancillary_variables = "qc_signal_return" ;

int qc_signal_return(time, height) ;
qc_signal_return:long_name = “Quality check results” ;
qc_signal_return:units = “1” ;
qc_signal_return:flag_method = “bit” ;
qc_signal_return:standard_name = “quality_flag” ; // Optional
qc_signal_return:bit_1_description = “Value is equal to missing_value”;
qc_signal_return:bit_1_assessment = “Bad” ;
qc_signal_return:bit_2_description = “The instrument detects an A/D start (timing corruption) error” ;
qc_signal_return:bit_2_assessment = “Bad” ;

6.8.10 Dimensionally Summarized Quality Control

Multi-dimensional QC data may be summarized for one or more of the dimensions into the remaining dimensions. The decision to summarize QC and how is left to the translator/mentor/developer. A technical description of the process may be too long to describe in an attribute. If the method used is not described in a variable attribute, a description of the method must be described in detail in a technical document.

The previously declared QC standards apply to summarized QC.

Example:

float signal_return_copol(time, height) ;
signal_return_copol:long_name = “Attenuated backscatter, co-polarization” ;
signal_return_copol:units = “counts/microsecond” ;
signal_return_copol:missing_value = -9999.f ;
signal_return_copol:ancillary_variables = “qc_signal_return_copol” ;
int qc_signal_return_copol(time) ;
qc_signal_return_copol:long_name = “Quality check results on variable: Attenuated backscatter, co-polarization” ;
qc_signal_return_copol:units = “1” ;
qc_signal_return_copol:flag_method = “bit” ;
qc_signal_return_copol:standard_name = “quality_flag” ; // Optional
qc_signal_return_copol:comment = “A quality control failure anywhere along the profile will result in the QC bit being set.” ;
qc_signal_return_copol:bit_1_description = “Value is equal to missing_value” ;
qc_signal_return_copol:bit_1_assessment = “Bad” ;
qc_signal_return_copol:bit_2_description = “The instrument detects an A/D start (timing corruption) error” ;
qc_signal_return_copol:bit_2_assessment = “Bad” ;

6.8.11 Integer Quality Control Variables

The optional integer QC variable has the same name as the data variable with the addition of a "qc" prepended to the variable name joined with an underscore. The standard integer QC variable follows the same descriptive text format as bit-packed QC with the exception of changing "bit" to "flag" in all attribute names and using integer values instead of bit-packed values. Integer QC only allows one state to be set at a time.

Flag numbers are required to be greater than or equal to zero. Negative flag numbers listed in the *flag_<#>_description* may cause issues with the method used for reading data. Some implementations may convert attribute names to program variables. A "-" character is not allowed in most programming language variable names.

The *flag_method = "integer"* indicates the values are interpreted as integers.

Required attribute for data variable:

- *ancillary_variables* = the corresponding QC variable name(s)

Required attributes for QC variable:

- *long_name = "Quality check results on variable: <variable's long_name attribute value>" ;*
- *units = "1" ;*
- *description = "This variable contains integer values indicating the results of QC test on the data. Non-zero integers indicate the QC condition given in the description for those integers; a value of 0 indicates the data has not failed any QC tests." ;*
- *flag_method = "integer" ;*

Required attribute for global attribute bit declaration:

- *description = "See global attributes for individual QC flag descriptions." ;*

Recommended attribute for QC variable:

- *standard_name = "quality_flag"*

CF has added a standard name of "*quality_flag*" that can be used to define the ancillary quality control variable using the standard CF method of using *standard_name*. This standard name is a subset of *status_flag*. At this time *quality_flag* is only a recommendation.

Example:

float upwelling_broadband (time) ;
upwelling_broadband:long_name = "Upwelling broadband radiation" ;
upwelling_broadband:units = "W/m^2" ;
upwelling_broadband:missing_value = -9999.f;
upwelling_broadband:ancillary_variables = "qc_upwelling_broadband" ;
int qc_upwelling_broadband (time) ;

qc_upwelling_broadband:long_name = "Quality check results on variable: Upwelling broadband radiation" ;
qc_upwelling_broadband:units = "1" ;
qc_upwelling_broadband:flag_method = "integer" ;
qc_upwelling_broadband:flag_1_description = "Value is equal to missing_value" ;
qc_upwelling_broadband:flag_1_assessment = "Bad" ;
qc_upwelling_broadband:flag_2_description = "Value is less than 2 standard deviations of historical mean" ;
qc_upwelling_broadband:flag_2_assessment = "Indeterminate" ;
qc_upwelling_broadband:flag_3_description = "Value greater than 2 standard deviations of historical mean" ;
qc_upwelling_broadband:flag_3_assessment = "Indeterminate" ;

6.8.12 Integer Global Attribute Declaration for Quality Control

The description of each test may be listed in the global attribute section if multiple variables use the exact same flag number and description. The description variable attribute must exist indicating that the test descriptions are listed in the global attributes. The attribute name follows the same format as the variable-level style except for a prepended "qc_". The prepending "qc_" to the integer flag description and assessment is to continue with the historical format currently in the ADC.

Required QC variable attribute for global attribute bit declaration:

- *description = "See global attributes for individual QC flag descriptions." ;*

Example:

// global attributes:
qc_flag_1_description = "Value is equal to missing_value" ;
qc_flag_1_assessment = "Bad" ;
qc_flag_2_description = "Value is less than the fail_min" ;
qc_flag_2_assessment = "Bad" ;
qc_flag_3_description = "Value is greater than the fail_max" ;
qc_flag_3_assessment = "Bad" ;
qc_flag_comment = "The QC variable values are integers indicating the results of QC tests on the data. Non-zero integers indicate the QC condition given in the description for those integers; a value of 0 indicates the data has not failed any QC tests." ;

6.9 Guidelines to Describe Source

When multiple inputs or algorithms are used to compute data variables, it may be useful to indicate the source of the input or algorithm at the variable level. In such cases, an optional data variable attribute or optional variable indicating the source of the data may be added.

6.9.1 Source Variable Attribute −Time Independent

If the source does not change, the input is indicated with an optional source variable attribute. Enough information to fully trace the values should be provided with a syntax of *"<datastream_name>:<variable_name>"*, or if the site and facility match the *site_id* and *facility_id* in the global attributes the site and facility may be omitted with syntax of *"<instrument_class>.<level>:<variable_name>"*. A full datastream name is detected by the existence of a ":" and single capital letter in the datastream name versus no capital letter indicating the need for inserting *site_id* and *facility_id*. The use of the algorithm or any other source identifier other than an ARM datastream variable must not contain a colon (':'). Multiple sources may be listed separated by a single-space character and an optional carriage return. The source attribute may optionally describe a method or algorithm instead of an input *datastream variable*. To distinguish an algorithm versus variable in the same file, the user will need to scan all the variable names in the file. If there is a match, the source is a variable in the current file. If there is no match, the source is an algorithm. If no source was used, then set the attribute to "*no_source_available*".

Example:

- ARM datastream and variable name:
  - *source = "sgpmetE13.b1:atmospheric_temperature" ;*
  - *source = "sgpmwrC1.b1:vap sgpmwrpC1.b1:vapor" ;*
  - *source = "mwr.b1:vap" ;*
  - *source = "mwr.b1:vap mwrp.b1:vapor" ;*
  - *source = "direct_normal_narrowband_1 diffuse_narrowband_1" ; // This assumes the variables listed are in the same file*
- Algorithm:
  - *source = "myers_briggs" ;*
  - *source = "rutherfurd_1.2" ;*
  - *source = "calvin_3.2 hobbs_1" ;*

6.9.2 Time-Dependent Source Attribute

If the source can change as a function of time but implementing a time-dependent source variable is too onerous, a single source attribute may be used. The source attribute must contain the full set of possible input sources. Discretion for use of a source attribute for time-varying inputs is given to the instrument mentor or VAP translator.

6.9.3 Source Variable −Time Dependent

For describing a time-dependent source, an optional source variable is used. If the source variable is referenced by one data variable, we recommend using the data variable name preceded by "source" and joined to the data variable name with an underscore (i.e., *source_atmos_temperature* for *atmos_temperature*). If a source variable is used for multiple data variables, we recommend using a name different than any of the data variables (i.e., *source_atmos_state* for *atmos_temperature, atmos_pressure, relative_humidy, wind_speed* and *wind_direction*).

An *ancillary_variables* attribute with the data variable is used to indicate the corresponding source variable name.

6.9.4 Source Variable −Flag Method

Multiple sources may be listed separated by a single-space character. Data type is integer.

Required attribute for data variable:

- *ancillary_variables* = <source variable name> ;

Required attributes for source variable:

- *long_name = "Source for variable: <data variable's long_name attribute value or generic description if used for multiple variables>" ;*
- *units = "1" ;*
- *description = "This variable contains integer values which should be interpreted as listed." ;*
- *flag_method = "integer" ;*
- *flag_<#>;_description = Description of source*

Optional attribute for source variable:

- *flag_<#>_comment = optional attribute to provide more details on how the data was computed.*

The meanings of **all** possible integer source values are indicated in the source variable attributes *flag_<#>_description*. One of the integer source values must describe a no source or default value. If indicating no source was used, set flag_<#>_description = "no_source_available". If a source preference ranking is appropriate, lower numeric values indicate higher preference.

Flag numbers are required to be greater than or equal to zero. Negative flag numbers listed in the *flag_<#>_description* may cause issues with methods used for reading data. Some implementations may convert attribute names to program variables. A "-" character is not allowed in most programming language variable names.

If the source is constant in other dimensions, the source variable is recommended to be a function of time only.

Example:

float aod (time) ;
aod:long_name = "Aerosol optical depth" ;
aod:units = "1" ;
aod:missing_value = -9999.f ;
aod:ancillary_variables = "source_aod" ;
int source_aod(time) ;
source_aod:long_name = "Source for variable: Aerosol optical depth" ;
source_aod:units = "1" ;
source_aod:flag_method = "integer";
source_aod:description = "This variable contains integer values which should be interpreted as listed." ;
source_aod:flag_0_description = "no_source_available" ;

source_aod:flag_1_description = ”mfr.c1:aerosol_optical_depth” ; // Site and facility are not needed because facility_id is C1 in global attributes.
source_aod:flag_2_description = ”mfrsr.b1:aerosol_optical_depth” ;
source_aod:flag_2_comment = “Fill gaps of 3 days or less via interpolation” ;
source_aod:flag_3_description =”nimfraod1mich.c1:aod” ;
source_aod:flag_4_description =”sgpnimfraod1michE13.c1:aod” ; // Note the site and facility are needed because the facility does not match the facility_id in global attributes.
source_aod:flag_5_description=”sgpnimfraod1michE13.c1:aodsgpnimfraod1michC1.c1:aod” ; // When listing more than one reference, be kind and make it as clear as possible. In this case adding the site and facility makes it more clear and easier to read for C1 facility.

6.9.5 Source Variable −Bit-Packed Method

Some datastreams may use multiple sources for each time sample. As stated in the previous section, multiple sources may be indicated with the *flag_<#>_description* method. If listing all possible combinations of sources is prohibitively complicated, we recommend use of the bit-packed method. Only one method of indicator is allowed at a time (no mixing of integer and bit-packed values in the same variable). The use of this type of method is indicated by the use of the *flag_method* variable attribute.

Required variable attribute:

- *long_name = “Source for variable: <data variable’s long_name attribute value or generic description if used for multiple variables>;” ;*
- *units = “1” ;*
- *description = “This variable contains bit packed integer values, where each bit represents a source of the data. Non-zero bits indicate the source used in the description for those bits.” ;*
- *flag_method = “bit” ;*
- *bit_<#>_description = “<description of the source>” ; // Describes the single source*

Example:

int source_aod (time) ;
source_aod:long_name = “Source for variable: Aerosol optical depth” ;
source_aod:units = “1”;
source_aod:description = “This variable contains bit packed integer values, where each bit represents a source of the data. Non-zero bits indicate the source used in the description for those bits; a value of 0 (no bits set) indicates no source.” ;
source_aod:flag_method = “bit”;
source_aod:bit_1_description = ”mfrsr.c1:aerosol_optical_depth” ; // Site and facility are not needed because facility_id is C1 in global attributes
source_aod:bit_2_description = ”mfrsr.b1:aerosol_optical_depth” ;
source_aod:bit_2_comment = “Fill gaps of 3 days or less via interpolation” ;
source_aod:bit_3_description = ”nimfraod1mich.c1:aod” ;
source_aod:bit_4_description = ”sgpnimfraod1michE13.c1:aod” ;

6.10 ADI Transform Parameters Using cell_transforms

The ARM Data Integrator (ADI) is used to combine multiple ARM netCDF files and transform different time steps and dimensions into a single netCDF file. To ensure the process used to generate the new file is well documented, the parameters applied by the user that affect the transformation are required within the netCDF file at the variable level.

All transform parameters are described within the optional variable attribute *cell_transforms*. Transform parameters are grouped by dimension following the *CF cell_methods* style (see *Cell Method Attribute* section in this document or CF standards document section 7.3). Detailed information about transform parameters and values can be found at https://engineering.arm.gov/ADI_doc/framework.html#transform-parameters.

Each dimension of the transformation is documented with the order of the dimension indicating order of transformation. The dimension name is followed by a colon and space (": ") and the transform type. Additional transform parameters for that dimension are followed within parentheses using a key-value pair style using colon and space delimiter between each key and value. This document is not intended to describe all the parameters and their meanings, but is intended to describe the method used to document the values. For a description of the parameter values ,see the ADI documentation.

6.10.1 Transform Type

Transform type is required to describe the type of transform used using one the predetermined keywords. Future transform names will follow the same pattern.

- TRANS_BIN_AVERAGE
- TRANS_INTERPOLATE
- TRANS_SUBSAMPLE
- TRANS_PASSTHROUGH

Additional Parameters

Additional parameters are required if they are used in the ADI transformation. Do not list a parameter if it is not used. A parameter that applies to all dimensions can be listed individually within parentheses following the dimension name or listed at the end. Listing the parameter at the end infers the parameter applies to all dimensions ─ or example, a quality control parameter that applies to all dimensions and is applied uniformly.

Example:

varmint(time, distance) ;
long_name = "Number of varmints destroying ARM instruments normalized by size" ;
units = "count/kg" ;
cell_transforms = "time: TRANS_SUBSAMPLE (range: 300) distance: TRANS_SUBSAMPLE (range: 5) qc_bad: 1,2,3" ;

In this example, the time dimension is transformed with the *TRANS_SUBSAMPLE* method using a range parameter of 300, and the distance dimension is transformed using the *TRANS_SUBSAMPLE* method with a range parameter of 5. Both dimensions are transformed with a *qc_bad* parameters using [1,2,3] values.

6.11 Process for Evaluating Exceptions

This section describes the ARM data standards exception request process.

6.12 Identifying Exceptions

Two primary methods are used to identify exceptions from the required standards. The first method involves the use of the ARM Process Configuration Management (PCM) tool at i.arm.gov-> PCM. The PCM tool is used by ingest and VAP developers, and allows them to design Data Object Design (DOD) for ARM data products. The DODs define metadata in the netCDF header, and the PCM tool analyzes and validates the metadata against current ARM data standards. Exceptions are flagged for further review.

The second method for identifying exceptions is simple visual inspection of data products by members of the ARM Data Quality Office (DQO), ADC, and ARM instrument mentors and VAP translators. This method relies on the expertise of the various parties and will only be used after the developer has attempted to resolve issues flagged in the PCM tool.

6.12.1 Exception Request

During the development of an instrument ingest or VAP <http://www.arm.gov/publications/tech_reports/doe-sc-arm-tr-093.pdf?id=49>, if the product developer becomes aware that there are compelling issues that will make it difficult or impossible to meet the required datastream standards, the developer should document the issue in the form of a request for an exception to the required standards. This request should be submitted to the ARM Standards Committee <standardscomm@arm.gov>. This request should be made as early in the development process as possible. It would not be appropriate, for example, to proceed with extensive development when the need for an exception is known. A valid effort should be made to understand the standards during development, or request help from a knowledgeable person early in the development process for help understanding standards. In most cases, it is expected to be much easier to make adjustments and minimize deviation from the standards early in the development process.

The primary purpose of the ARM Standards Committee is to review requests to exempt data products from adherence to the ARM standards. They may also consider and make recommendations on changes to the standards themselves and recommend new standard_name to the CF convention. Changes to the standards document would be further reviewed and enacted through a Baseline Change Request.

The Standards Committee will consist of the following five individuals representing key datastream-related stakeholders:

- VAP manager
- metadata QC reviewer

- representative from the ADC
- representative from the DQO
- translator or mentor to represent the scientific community.

With the exception of the VAP manager, these positions will be filled on a rotating basis with a term of two-years. The incoming members should be identified at the beginning of the previous term. In this way, incoming members may serve as back-ups should a committee member be unavailable. To stagger the rotation of members, the metadata QC reviewer and ADC members will only serve an initial one-year term. After the initial year the members will serve a two-year term.

6.12.2 The Review Process

The Standards Committee should return a decision on an exceptions request as expeditiously as possible to minimize delays of the development process. The committee should strive to return decisions within two weeks of receipt of a request. If the committee is unable to meet due to the unavailability of one or more members for an extended period, those members may be replaced in a review by incoming members, or a suitable alternate should the incoming members also be unavailable. If a member of the committee has a conflict of interest (e.g., they are the translator or developer for the product), they should recuse themselves from the review and should be replaced by the incoming member or suitable alternate.

Committee members should consult with other stakeholders in their deliberation as necessary.

To grant an exception request, a majority is required to help ensure that a clear case has been made. If the committee has particular concerns about part of the request, they may provide a response to the petitioner indicating their concern over those specific points and request a revised proposal.

6.12.3 Options for the Standards Committee

Approval:

If the request is approved, this means that the data product developer will be permitted to continue with their development toward a product deviating from the ARM data standards on the points approved in their request. This product will be tagged with a transparent Data Quality Report (DQR) indicating that the product deviates from ARM standards. The DQR will summarize the ways in which the file deviates from the standards. The product will be fully discoverable through the ADC Data Discovery tools.

Denial:

If the request is denied, this indicates that the committee believes that the benefit of conforming to the standards justifies the cost of doing so and that the reasons put forth by the petitioner to bypass certain standards were not compelling. Denial by the committee is a programmatic denial to invest further in the development of the product under the terms proposed by the petitioner, so development of the product should cease unless a later compromise solution is obtained (see “Appeal” below).

Conditional Acceptance:

If the committee agrees with certain points of the petitioner's request but disagrees with others, they may indicate their conditional approval to the petitioner with a request to modify the points of concern.

Appeal:

If a petitioner's request is denied or certain elements of their request are denied, they may modify their request or gather additional background information to support their original request and resubmit their request to the Standards Committee in the form of an appeal. The petitioner may submit two such appeals for a given product.

7.0 CF standard_name Recommendations

Unidata CF maintains the database of names and descriptions to clarify the data values in the *standard_name* attribute. Most of the current names came from the modeling community and do not correctly describe many measurements collected by the ARM user facility. To correctly use the *standard_name* method, ARM will recommend names to the list with corresponding definitions. The current process for recommending a name is to send an e-mail to the cf-metadata@cgd.ucar.edu listserv for discussion by the CF user community. CF has developed a set of guidelines for the naming convention (<http://cfconventions.org/Data/cf-standard-names/docs/guidelines.html>). The discussion in the listserv will suggest updates if needed and decide on new name adoption. The typical timeline for new name adoptions is on the order of a few months.

New CF *standard_name* suggestions will come from the Standards Committee with a single committee member tasked with tracking the progress of a proposed *standard_name* until adopted.

Appendix A

Definitions

<#>	Enumerated number placeholder.
ADC	ARM Data Center
ADI	ARM Data Integrator
CF	Climate and Forecast
Developer	Person responsible for software development
DOD	Data Object Design
DQR	Data Quality Report
ECO	Engineering Change Order
ECR	Engineering Change Request
EWO	Engineering Work Order
Facility	A specific smaller geographical location within a Site where an instrument is located. Multiple facilities may exist for each Site. Can also be used to designate a particular aircraft or aerosol chamber, etc.
Instrument	A single piece of hardware or group of sensors hardware that records one or more measurements.
Instrument Class	The name of the group of data products produced from a specific instrument (e.g., EBBR, MET, ECOR).
Instrument Code	The specific process name (e.g., 30ebbr, irt200ms, qcrad1long) with descriptors when they apply (e.g., scan types for radars). Instrument codes remain constant, and datastream names are constructed from the instrument code, site, facility, and data level.
Mentor	Person responsible for instrument installation and general operations
Metadata	Information describing a set of data or a piece of hardware
NaN	Not a number indicator
PCM	Process Configuration Management
QC	quality control
RAW	Data file created by instrument
Site	Geographical region within which a set of measurements are being conducted.

Translator	Person responsible for VAP development and maintenance
UTC	Coordinated Universal Time
VAP	value-added product: A higher-order data product that includes derived quantities not measured directly or routinely, a combination from multiple sensors, or improvements to quality to fulfill unmet measurement needs of the ARM facility.
Variable	A variable in terms of netCDF files refers to one individual measured quantity and may consist of a single value represented as a scalar or multiple values represented as a vector or array. This is often used interchangeably with the term *field*.

Appendix B

Bin Values Changing Each Time Step

dimensions:
time = UNLIMITED ; // (1440 currently)
droplet_size = 21 ;
bound = 2 ;

variables:
double time(time) ;
time:long_name = “Time offset from midnight” ;
time:units = “seconds since 2013-01-06 00:00:00 0:00” ;
float droplet_size(time, droplet_size) ;
droplet_size:long_name = “Droplet size” ;
droplet_size:units = “um” ;
droplet_size:bounds = “droplet_size_bounds” ;
float droplet_size_bounds(time,droplet_size,bound) ;
float ccn_number_concentration(time, droplet_size) ;
ccn_number_concentration:long_name = “AOS ccn number concentration by bin” ;
ccn_number_concentration:units = “count” ;
ccn_number_concentration:missing_value = -9999.f ;
ccn_number_concentration:cell_methods = “droplet_size: sum” ;

Appendix C

ARM UDUNITS-Compliant Unit Descriptors

https://wiki.arm.gov/bin/view/Engineering/StandardizingDODs

For complete UDUNITS-compliant units reference, see UDUNITS-2 database that comprises the following XML files:

- SI unit prefixes : http://www.unidata.ucar.edu/software/udunits/udunits-2/udunits2-prefixes.xml
- SI base units : http://www.unidata.ucar.edu/software/udunits/udunits-2/udunits2-base.xml
- SI derived units : http://www.unidata.ucar.edu/software/udunits/udunits-2/udunits2-derived.xml
- Units accepted for use with the SI : http://www.unidata.ucar.edu/software/udunits/udunits-2/udunits2-accepted.xml
- Non-SI units : http://www.unidata.ucar.edu/software/udunits/udunits-2/udunits2-common.xml

The udunits2 units database is also searchable for name or symbol at https://www.cicsnc.org/pub/jbiard/Udunits2Tables.html

Recommended Units

Base Quantity	Unit Name	Symbol	Comment or Other Possible Units
length, distance, height	meter	m	cm inch, mm
frequency, sample rate	hertz	Hz	1/s
force	newton	N	
energy	joule	J	
power	watt	W	
electric potential, voltage	volt	V	
volume	liter	L	cc, cm^3, L, mL, m^3
atmospheric pressure, barometric pressure, station pressure	kilopascal	kPa	hPa, mbar, psi, inHg
density, water vapor density, absolute humidity, concentration of trace substance	gram per cubic meter	g/m^3	kg/m^3, g/cc, g/cm^3

Base Quantity	Unit Name	Symbol	Comment or Other Possible Units
energy flux density, irradiance, heat flux, net radiation	watt per square meter	W/m^2	
plane angle, azimuth elevation, wind direction, zenith	degree	degree	rad
latitude	degree north	degree_N	degree_S
longitude	degree east	degree_E	degree_W
precipitable water vapor	centimeter	cm	mm, in
precipitation	millimeter	mm	in, cm
precipitation rate	millimeter per second	mm/s	
radiance	watt per square meter per steradian	W/(m^2sr)	W m^-2sr^-1
relative humidity	percent	%	1
solid angle	steradian	sr	
temperature, dry bulb, wet bulb, dewpoint, potential, equivalent potential, virtual	celsius	degC	degF, K
velocity, wind speed, ascent rate	meters per second	m/s	
water vapor mixing ratio (per mass of dry air)	grams per kilogram	g/kg	
water vapor pressure	kilopascal	kPa	hPa, mbar
wavelength	nanometer	nm	um
wavenumber	inverse centimeter	cm^3-1	
bin	1	1	
mass density	gram per cubic centimeter	g/cm^3	
number density	inverse cubic centimeter	1/cm^3	count/cm^3
molar mixing ratio	micro-mol per mol	umol/mol	
volumetric mixing ratio	parts per million by volume	ppm	ppmv
counts	count	count	1
ratio, fraction	fraction	1	
probability	fraction	1	
relative power	deci-bell	dB	
soil moisture content by volume	cubic meter per cubic meter	m^3/m^3	cm^3/cm^3
soil water potential	kilopascal	kPa	

Prefixes

Prefix	Power of 10	Symbol
pico	-12	p
nano	-9	n
micro	-6	u
milli	-3	m
centi	-2	
deci	-1	d
hector	2	h
kilo	3	
ega	6	M
giga	9	G
tera	12	T

Appendix D

ARM netCDF Data File Example

data filename = sgptempprofile10sC1.c1.20130101.010203.nc

dimensions:

time = UNLIMITED ; // (14400 currently)
bound = 2 ;
height = 100 ;

variables:

int base_time ;

base_time:string = "01-Jan-2013,00:00:00 GMT" ;
base_time:long_name = "Base time in Epoch" ;
base_time:units = "seconds since 1970-1-1 0:00:00 0:00" ;
base_time:ancillary_variables = "time_offset" ;
double time_offset (time) ;
time_offset:long_name = "Time offset from base_time" ;
time_offset:units = "seconds since 2013-01-01 00:00:00 0:00" ;
time_offset:ancillary_variables = "base_time" ;
time_offset:bounds = "time_bounds" ;
double time (time) ;
time:long_name = "Time offset from midnight" ;
time:units = "seconds since 2013-01-01 00:00:00 0:00" ;
time:standard_name = "time" ;
time:bounds = "time_bounds" ;
double time_bounds (time, bound) ;
time_bounds:long_name = "Time cell bounds" ;
float height(height) ;
height:long_name = "Center of height bin" ;
height:units = "m" ;
height:standard_name = "height" ;
height:bounds = "height_bounds" ;
float height_bounds(height, bounds) ;
height_bounds:long_name = "Height bin bounds" ;
height_bounds:units = "m" ;
float atmospheric_temperature(time, height) ;

atmospheric_temperature:long_name = “Atmospheric temperature” ;
atmospheric_temperature:units = “degC” ;
atmospheric_temperature:missing_value = -9999.f ;
atmospheric_temperature:standard_name = “air_temperature“ ;
atmospheric_temperature:cell_methods = “time:mean height:mean“ ;
atmospheric_temperature:ancillary_variables = “qc_atmospheric_temperature source_atmospheric_temperature instrument_status” ;

int qc_atmospheric_temperature(time, height) ;
qc_atmospheric_temperature:long_name = “Quality check results on variable: Atmospheric temperature” ;
qc_atmospheric_temperature:units = “1” ;
qc_atmospheric_temperature:standard_name = “air_temperature” ;
qc_atmospheric_temperature:flag_method = “bit” ;
qc_atmospheric_temperature:comment = “A quality control bit set anywhere along the profile will result in the bit being set.” ;
qc_atmospheric_temperature:bit_1_description = “Value is equal to missing_value”;
qc_atmospheric_temperature:bit_1_assessment = “Bad” ;
qc_atmospheric_temperature:bit_2_description = “The instrument detected a hardware failure” ;
qc_atmospheric_temperature:bit_2_assessment = “Bad” ;
qc_atmospheric_temperature:bit_3_description = “Values greater than two standard deviations of historical distribution” ;
qc_atmospheric_temperature:bit_3_assessment = “Indeterminate” ;

int source_atmospheric_temperature (time) ;
source_atmospheric_temperature:long_name = “Source for variable: Atmospheric temperature” ;
source_atmospheric_temperature:units = “1” ;
source_atmospheric_temperature:standard_name = “air_temperature” ;
source_atmospheric_temperature:description = “This variable contains bit-packed integer values, where each bit represents a source of the data. Non-zero bits indicate the source used in the description for those bits; a value of 0 (no bits set) indicates no source.” ;
source_atmospheric_temperature:flag_method = “bit” ;
source_atmospheric_temperature:bit_1_description =”sgpsondewnpnC1.b1:tdry” ;
source_atmospheric_temperature:bit_2_description =”sgpaeriprofC1.c1:temperature” ;
source_atmospheric_temperature:bit_3_description =”sgp1290rwpC1.c1:temp” ;
source_atmospheric_temperature:bit_4_description =”conwarfX1.a1:atmos_temp” ;

int instrument_status(time) ;
instrument_status:long_name = “Instrument status” ;
instrument_status:units = “1” ;
instrument_status:missing_value = -9999 ;
instrument_status:flag_masks = 1, 2, 4, 8;
instrument_status:flag_meanings = “power_failure hardware_fault software_fault maintenance_mode” ;

www.arm.gov

U.S. DEPARTMENT OF
ENERGY

Office of Science