U.S. DEPARTMENT OF ENERGY | Office of Science

ARM Data File Standards Version: 1.1

ARM Standards Committee

January 2015

ARM CLIMATE RESEARCH FACILITY

DOE/SC-ARM-15-004

ARM Data File Standards Version: 1.1

ARM Standards Committee

January 2015

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

Contents

Tables

Standards Committee

This document was prepared by the 2014 Standards Committee consisting of:

Ken Kehoe (OU) *Chair*	Mike Jensen (BNL)
Sherman Beus (PNNL)	Raymond McCord (ORNL)
Alice Cialella (BNL)	Renata McCoy (LLNL)
Scott Collis (ANL)	Sean Moore (ATK)
Brian Ermold (PNNL)	Justin Monroe (OU)
Robin Perez (PNNL)	Brad Perkins (LANL)
Stefanie Shamblin (ORNL)	Tim Shippert (PNNL)
Chitra Sivaraman (PNNL)	

ANL = Argonne National Laboratory	*ATK = Alliant Techsystems Inc.*
BNL = Brookhaven National Laboratory	*LANL = Los Alamos National Laboratory*
LLNL = Lawrence Livermore National Laboratory	*ORNL = Oak Ridge National Laboratory*
OU = University of Oklahoma	*PNNL = Pacific Northwest National Laboratory*

1.0 Introduction

The Atmospheric Radiation Measurement (ARM) Climate Research 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 diverse data sets containing observational and derived data, currently accumulating at a rate of 30 TB of data and 150,000 different files per month (http://www.archive.arm.gov/stats/storage2.html). 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 multi-year 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 the 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 data sets must adhere to required ARM standards to be published in ARM archives, unless an exception is granted. The historical data will be brought into compliance with the standards as it is reprocessed.

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 http://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.

1.1 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 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.

1.1.1 Example of Tools Using Standards

The ARM Facility has many individual software tools using the standards listed in this document. Conforming to the standards enables the ARM Facility 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, NCVweb, and ARM*STAR at the Data Quality Office
- DSView, Ingest and VAP processing at the Data Management Facility
- Data Discovery, storage, custom data files at the ARM Data Archive
- Process Configuration Management and Metadata Management Tool at the External 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 be in compliance with the ARM standard. To reference a standards version number, all required standards must be met except those allowed to deviate by the exceptions committee. Unless indicated, all standards listed are required. If the required standards are not met, data will not be published in the ARM Data Archive unless an exception is granted.

A few required 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 standards, which 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 fields (i.e., quality control, source, or state indicator field) 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 field or metadata is used, the required and recommended standards listed in those sections apply.

4.0 Significant Changes

This section lists changes to the existing de-facto standards that may require the most attention.

- changing from .cdf to.nc file extension
- require both base_time & time_offset, and time in CF convention methods
- additional time cell boundaries for time-averaged data
- additional coordinate cell boundaries for coordinate-averaged data
- removal of qc_time as a required field
- explicit criteria for filename data level
- reduction in use of abbreviations in field names
- explicit method for state indicator fields
- addition of *datastream* and *platform_id* global attributes
- *missing_value* field attribute required with conditions
- *standard_name* field attribute required if a primary field and the standard name exists in the CF table
- explicit method for integer quality-control fields
- explicit method for source fields.

4.1 Changes from Version 1.0

A summary of changes from *ARM Data File Standards Version: 1.0* include the following:

- a reference for coordinate variables defined with *standard_name*, including the distinction of above ground level (AGL) vs. mean sea level (MSL)
- requiring flag numbers for state fields be positive if the optional attributes are used
- defining all possible sources when using the source field method including a “no source” or “default” value
- defining the method of requesting new *standard_name* to CF
- the creation of an official method for the standards exception process.

5.0 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.

Version 3/classic format netCDF is the ARM Facility's choice for final data format because it supports efficient data storage and reliable/robust documentation of the data structure. More information about netCDF is available at http://www.unidata.ucar.edu/packages/netcdf/faq.html.

High-volume data may be treated as a special case and allowed to use netCDF version 4 to take advantage of the compression option. Use of netCDF 4 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. Use of netCDF 4 will be granted through the Exception Committee process.

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 http://www.hdfgroup.org and http://www.hdfeos.org.

6.0 Construction of Data Filename

6.1 File Naming Conventions for Processed Data

ARM netCDF files are named according to the following naming convention. All characters are lowercase except for 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). 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. Exceptions may be made for moving deployments such as ship and aircraft, or for large geographic areas for satellite data.

(inst) is the ARM instrument abbreviation (e.g., mwr, met, ecor, mpl), or the name of a ARM value-added product (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. The instrument abbreviation must not end with a number as this can be confused with the data temporal resolution descriptor or other optional descriptors following the instrument abbreviation.

(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.

(temporal) is an optional description of data temporal resolution (e.g., 30m, 1h, 5s, 200ms, 14d). All temporal resolution descriptors require a unit identifier. Accepted abbreviations include: ns=nanosecond, us=microsecond, ms=millisecond, s=second, m=minute, h=hour, d=day, mo=month, yr=year. It is recommended that the end-user primary datastream 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 ARM Facility designation. A facility is designated with a capital letter followed by one or two numbers not padded with zeros (e.g., S1, C1, E13, B4, M1, I4). Extended facilities around a central facility at the remote sites (excluding Southern Great Plains (SGP)) indicate the correlation to the central facility by matching the first of the two required numerical characters. Example: central facility TWP-C2 is related to extended facilities E20, E21, E22, while TWP-C1 is related to E10, E11, E12.

External data products that cover a large locale use the facility designation of X1, while data products that are 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.

Character coding designations: B = boundary, C = central, E = extended, I = intermediate, M = mobile, S = supplemental, X = external data site.

Notes:

- S0<#> has been used to indicate a supplemental facility co-located with the main facility. The continuation of this convention is not recommended.
- Instances of a co-located deployment of the mobile aerosol observing system (MAOS) with the 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.

(dl) data level is the two-character descriptor consisting of one lowercase letter followed by one number, except for RAW data level, which will consist of two characters (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 UTC time in hour, minute, second format, consisting of exactly six characters and indicating 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 seconds 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 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.

6.1.1 Filename Length

The **TOTAL** length of a filename sent to the ARM Data Archive **MUST** be 60 characters or less to meet the requirements of the current ARM Data Archive database system. The ARM Data Archive uses a 64-character filename field 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 ARM Data Archive.

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

6.1.2 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 lowercase letter followed by one number (except for RAW data).

00: raw data – primary raw datastream 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 typically used as input to higher-level data products. Not intended for distribution to data users.

a1: calibration factors applied and converted to geophysical units.

a2 to **a9**: further processing on *a1* level data that does not merit *b*-level classification. This level also applies to external satellite files 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 has 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 always used as input to higher-level data products. Not intended for distribution to data users.

b1: quality-control checks applied to at least one measurement and stored in an accompanying quality-control field, 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 always used as input to a higher-level VAP. 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, which have had algorithms applied by an external source.

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 in algorithm.

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.

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 field 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 fields satisfying *b* level requirements.
- Data level *c0* to *c9* is restricted to data derived or calculated through value-added processing. Lower-level datastreams will be kept in the ARM Data Archive 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 ARM Data Archive.

6.1.3 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 Facility leaders through and Engineering Change Request (ECR).

6.1.4 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 ARM Data Archive suggests file sizes under 20 GB, but can manage file sizes up to 8 TB.

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.

6.2 Guidelines for Original RAW Filename

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.

6.3 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 six-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 sub-directory 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.

6.4 File Naming Conventions for TAR Bundles

TAR bundles are named according to the following naming convention:

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

Where:

(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.

tar is the TAR bundle file extension.

It is recommended to create 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. It is recommended that the data files within the TAR file contain enough information to describe the data including location and time.

6.5 File Naming Conventions for Field Campaign TAR Bundles

Field 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 ARM Data Archive by the field campaign administrator.

000000 is the hhmmss field (the hhmmss resolution is not currently is 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, then 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.

6.6 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.

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 photograph.

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, but are not recommended for future development.

6.7 Guidelines to Name Quicklook Plot Filenames

The standard convention for VAP quicklook plot filenames created at the Data Management Facility 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

6.8 Case Sensitive File Naming

Data filenames 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.

7.0 Guideline for netCDF File Structure

7.1 Dimensions

7.1.1 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.

It is recommended that the number of dimensions used in a single file be as few as possible. Fields 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.

7.1.2 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 it is not recommended to deviate from the Gregorian calendar. The *calendar* field attribute is optional if the calendar used is Gregorian. Note, the use of a Julian calendar vs. a Gregorian calendar may have slight differences since the Gregorian calendar defines one year as 365.242198781 days vs. 365.25 days in a Julian calendar.

Time is defined through the use of both a *time* field, and *base_time* and *time_offset* fields. 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. The units of time, *base_time* and *time_offset* must be the same and the values of *time* and *time_offset* must be the same. This will decrease the likelihood of a user interpreting the time values incorrectly.

It is recommended to start the time at UTC midnight and indicate this format in the *long_name*. Starting time at midnight allows for easy interpolation of the values (i.e., dividing the time field by 3600 to convert from seconds to hours).

7.1.3 base_time and time_offset Fields

Time in ARM netCDF files is indicated in 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 timezone is UTC, but a different time zone may be defined using a timezone offset from UTC.

Time is indicated with the combination of two fields (*base_time*, *time_offset*) where the result is 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* field 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").

7.1.4 time Field

The *time* field 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 field 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 at http://ferret.wrc.noaa.gov/noaa_coop/coop_cdf_profile.html). Other conventions besides “seconds since” are allowed but not recommended. The use of “months since” and “years since” are 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: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
```

7.1.5 Time Bin Boundary

Most data values are reported as an average of values over a predefined number of samples. Indicating the bin boundaries and the location of the reported time value within the bin is critical to properly understand the reported data. For all non-instantaneous data, the values of each averaging time bin is required. A *bounds* field attribute indicates the corresponding two-dimensional field dimensioned by *time* and a bounds dimension containing the bin boundary values. CF convention does not require a *long_name* attribute for the bound field, but it is recommended to add 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 field.

The existence of a time bounds field along with an “ARM-<#>”. Conventions global attribute indicates all fields dimensioned by time are assumed averaged over the *time* bounds period unless a *cell_methods* field attribute exists. The method described in the *cell_methods* field attribute supersedes the time-averaged assumption. See *cell_methods* 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: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 = “One minute average 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* field contains the starting and ending time values for each time bin. If the *units* attribute is omitted the values are offset from to the *time:units* time. The individual *time* value indicates where within the bin *time* is reported. The *long_name* and *units* attributes are not required for *time_bounds*, and the field may not have missing values.

The optional *time_bounds*:*bound_offsets* attribute declares the width of each averaging period. Setting the attribute requires that every averaging period is expected to be consistent. If the averaging period is not consistent, the attribute is omitted.

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 seconds 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 bin. The averaging period is consistent so the optional attribute bound_offsets equals [-30, 30].

7.1.6 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. It is recommended to use singular names and not use 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 field.

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.

7.1.7 Reference for Coordinate Units

Some coordinate fields use units that require a frame of reference declaration. Examples include the difference between height AGL vs. height above MSL. Both of these coordinate fields have units of meters, 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.

7.1.8 Referencing AGL and MSL

When referencing 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 than above 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 vs. GMT). Currently, CF has no standard name for *mean sea level*.

7.1.9 Coordinate Bin Dimension

Binned data is common in atmospheric data and needs sufficient metadata to describe the bin ranges. Typically, binned data is evenly spaced and reported at the center of the bin value. 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 bin with a two-dimensional array. A *long_name* and *units* attribute are recommended but not required.

If the bin size is consistent, the optional *bound_offsets* attribute describes the size of the bin. If the bin size is not consistent, *bound_offsets* is omitted.

Example:

```
dimensions:
    time = UNLIMITED ; // (1440 currently)
    bin = 21 ;
    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
    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* variable contains values corresponding to each binned sample. The bin range is contained 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* values and its value 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 bin.

The variable referred to in the bounds attribute (*bin_bounds* in this example) does not require a *long_name* or *units* attribute.

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.

7.1.10 Additional Dimension

Additional dimension may be needed for string arrays, bounds, or other dimensions 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. Some 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 = "unitless" ;
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."

7.1.11 Cell Method Attribute

The optional *cell_methods* field attribute describes how the data was derived by indicating the method used. This method is well defined by the CF convention and is extensionable to describing multi-dimensional data sets. Additional description can be found at http://cfconventions.org/.

The addition of *cell_methods* to a data field describes how the data was 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 multi-dimensional 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";

7.2 Location Fields

The instrument location is described using latitude, longitude, and altitude fields. The required unit of latitude is degrees north, a field name of *lat*, and standard_name = "*latitude*". The required unit of longitude is degrees east, a field name of *lon*, and *standard_name = "longitude"*. The recommended unit of altitude is meters above MSL. The required field 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. Use of *standard_name* attributes are recommended. The use of the specific *lat*, *lon*, and *alt* field names are required to be consistent with historical data. *lat*, *lon,* and *alt* fields can be dimensioned by time 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";
```

7.3 Guidelines for Construction of Field Names

A field name should convey a basic understanding of the associated data. File space is not an issue, and cryptic field names are typically only understood by the person who originally created the name. ARM field 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 uppercase letters sparingly.
- The field name is constructed by joining the names to the qualifiers using underscores (_).
- Field names are recommended to be concise. One has to be reasonable when picking field names.
- Abbreviations are recommended only for limiting excessively long field names, for following previous conventions, or for clarity.
- Field name lengths are required to not exceed 64 characters to comply with ARM Data Archive database storage requirements.
- Single-character names are not recommended.

- Common field names are recommended to use common ARM field names which follow the standards and are used in other datastreams to promote clarity across datastreams. Review pick list for common field names.
- Field names and dimensions are recommended to be singular (i.e., temperature not temperatures).temperatures)
- Greek letters are not allowed in netCDF3. It is recommend to not spell out Greek letters, formula symbols, or units.

Field names should be as concise. However, it also is recommended that names be spelled out as much as possible. For example, “temperature” rather than, “temp” is recommended, unless the full field name becomes unreasonably long. In this instance, “temp” would be the abbreviation. Field names should be as descriptive as possible. For example, *atmospheric_temperature* is more descriptive than *temperature* alone. A field labeled *temperature* could describe air temperature, instrument temperature, derived temperature, etc.

7.3.1 Field Names Hierarchy

Name hierarchy is used for field 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 field 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 field name is related to the DOD for which it exists. A field name should convey the required information to distinguish the different fields, 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 field name. Or, if every field in the file is an average, there is no need to indicate average in the field name.

It is recommended that related field names repeat the same basic pattern for similar fields. This may result in using an abbreviation for the basic field. If the field was not accompanied by other fields, the abbreviation would not be used. For example, a datastream containing a measurement of aerosol optical thickness with no accompanying fields would use *aerosol_optical_thickness*. If the measurement has accompanying fields extending the field name length, the field 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.

7.3.2 Field Name Abbreviations and Qualifiers

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

7.3.2.1 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

7.3.2.2 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

7.3.2.3 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

7.3.2.4 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

7.4 State Indicator Field

Some fields 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 field is typically metadata rather than data. The indication of a state is recommended to follow CF convention formatting suggestions.

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

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

7.4.1 Exclusive 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 defined *flag_values* and *flag_meanings* field attributes. The different flag meanings are strings separated by a single-space character. Individual flag meanings may not contain spaces and consist 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 method used for reading data. Some implementations may convert attribute names to program variables. A "-" character is not allowed in most programming language variables names.

```
int hatch_status (time);
    hatch_status:long_name = "Hatch status";
    hatch_status:units = "unitless";
    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

7.4.2 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* field 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. A more detailed description of the state may be made through optional *bit_<#>_description* attributes.

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" ;
sensor_status:bit_2_description = "Hardware fault" ;
sensor_status:bit_3_description = "Offline mode";
sensor_status:bit_4_description = "Instrument performing calibration";
sensor_status:bit_5_description = "Instrument in maintenance mode";

To detect which bits have been set, repeatedly 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 data value is 6, its binary representation is 00000110, so the second and third bits are set. Recursively, using AND for each *flag_masks* value ([1, 2, 4, 8, 16]) has six results [0, 2, 4, 0, 0], indicating only the second and third flags have been set: *hardware_fault* and *offline_mode*.

7.5 Field Attributes

In general, field 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_<#>*).

7.5.1 Required Field Attributes

- *long_name*: Must be unique in regards to the other fields 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 conference). Long names may not change without a DOD change. First letter of the *long_name* attribute value are recommended to be capitalized.
- *units*: See current list of approved unit descriptors in Appendix C. Must be Unidata *udunits* compliant unless units descriptor is currently unlisted.

7.5.2 Required with Conditions

- *missing_value*: If the data field uses a specific value to represent no data a *missing_value* attribute must be declared. There is no required value but the recommended value is -9999. Do not include with coordinate fields. The value must be the same type as the corresponding data values. The value is recommended to be outside the valid data range.
- *standard_name*: Required if a primary field and the standard name exists in the CF table.

7.5.3 missing_value vs. _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.

When a netCDF file is initially created, all data values are set to a standard fill value, differing by data type. During the write state, the values are changed to data values. Therefore, if a fill value exists in the netCDF file, 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 the writing software uses a value different than the default netCDF fill value, there will be two different values indicating non-data values. Therefore, a user may need to mask the *missing_value*, and default fill value or *_FillValue* from the analysis.

7.6 Standard_Name Attribute

When possible, it is strongly recommended to include a CF *standard_name* attribute to officially describe the data. Official string values for the *standard_name* attribute must be taken from the CF standard name table. Creating new string values when a standards name does not exist is not recommended.

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”;
```

7.7 ARM Standard Field Attribute Names

- valid_min
- valid_max
- valid_delta
- qc_min

- qc_max
- resolution
- comment
- comment_<#> (used for multiple distinct comments within a single field)
- precision
- accuracy
- uncertainty
- 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)

7.7.1 Other Possible Attributes (Not All Inclusive)

- valid_range
- actual_wavelength
- corrections
- filter_wavelength
- FWHM (capital letters ok)
- sensor_height
- positive
- source

7.8 Sensor Height

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 fields represent data at different heights, each field indicates the sensor height with the *sensor_height* attribute. The presence of a *sensor_height* field attribute supersedes the global attribute. To determine the height of the sensor above MSL, add *sensor_height* value to *alt* field 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 vs. 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”;
```

7.9 Attribute Datatype

Field attributes set to a numeric value must match the same data type as defined for the corresponding data field type.

Example:

```
double wind_direction (time);
      wind_speed:long_name = “Mean wind direction”;
      wind_speed:units = “degree”;
      wind_speed;missing_value = -9999.; // Value is set as double precision instead of float precision
```

8.0 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.

Required and Recommended Global Attributes

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

(Required global attributes are **bold**.)

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: C = Conventions = “ARM-1.0 CF-1.6/Radial instrument_parameters radar_parameters radar_calibration”;

Reference hyperlink: http://www.unidata.ucar.edu/software/netcdf/docs/netcdf.html#Attribute-

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 fields 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-new line-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 ranges 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 full path used to create 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 consist 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, else 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: sampling_interval = “5 minute”;

sensor_height

Definition: Records height of all sensors AGL. If multiple sensors at different heights exist, use field-level attribute. See *Sensor Height* section for format details. If *sensor_height* is defined at field level for all relevant fields, 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 VAP 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 it.

Example: “http://www.arm.gov/data/vaps/armbe/armbecldrad” doi

Definition: Digital Object Identifier (DOI) number used to reference the data. Please contact ARM Data Archive (armarchive@ornl.gov) for generating DOIs.

Example: “10.5439/1039926”; // Note this is a character string

doi_url

Definition: Full Uniform Resource Locator including DOI numbers. Please contact ARM Data Archive (armarchive@ornl.gov) for generating DOIs.

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-new line-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”;

9.0 Quality-Control Parallel Fields

In addition to the data fields, optional quality control (QC) fields 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 fields may use integer value method for single value test results, or bit packing method for multiple value test results. The decision of which method to use is left to the developer/mentor/translator.

9.1 Bit-Packed Numbering Discussion

QC fields 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://engineering.arm.gov/~shippert/ARM_bits.html

or in PDF format:

https://engineering.arm.gov/~shippert/ARM_bits.pdf

9.2 Standard Bit-Packed Quality Control Fields

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

The *flag_method* = “*bit*” field attribute indicates the values are bit-packed.

QC fields are type integer (recommend 32-bit integer), unless appreciated to a higher precision to accommodate more tests than the integer resolution can accommodate. If greater than 32 tests are required, a method must be proposed to the Exception Committee for review.

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

Required attribute for data field:

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

Required attributes for QC field:

- *long_name* = "*Quality check results on field: <field's long_name attribute value>";*
- *units = "unitless"*;
- *description = "This field 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"*;

Attributes describing the QC tests may be defined at either the field or global level. A mixture of field or global level definitions is allowed in the same file, but definitions may only occur in one location for a single field (global level or field level). Field-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.

9.2.1 Field-Level Bit Description

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

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

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

The following field attributes are optional:

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

Each <#> indicates the bit number. Examples are shown in Table 1:

Table 1. Optional field attribute examples.

Bit	**Field 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*

9.2.2 Standard ARM Quality Control

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

Standard ARM QC bits use this specific format when defined as field attributes. The bit numbers 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 valid_min";*
- *bit_2_assessment = "Bad";*
- *bit_3_description = "Value is greater than the valid_max";*
- *bit_3_assessment = "Bad";*

For a field-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.

9.2.3 Unused ARM Quality Control Bit

When an individual bit is unused, but must be declared, the field *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* field.

Required attributes:

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

Optional attribute:

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

The declaration of a *valid_min* or *valid_max* does not require the addition of QC fields. However, if a QC field exists and the valid_min or valid_max attributes are defined, the test is recommended to be implemented.

9.2.4 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 field attribute name. The bit description must not change between DOD versions. If a parameter value might change, it is recommended to phrase the description in a generic manner, reference an external source, or use a referenced field attribute.

When a test references another field in the same file, it is recommended that the field 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 field: Upwelling broadband
    radiation";
    qc_upwelling_broadband:units = "unitless";
    qc_upwelling_broadband:flag_method = "bit";
    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 two standard deviations of
    historical mean";
    qc_upwelling_broadband:bit_3_assessment = "Bad" ;
```

In this example, the test*_parameter_value* QC field 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 field unless the parameter value can be clearly described with the attribute name and has significant value to the data field. The location of the attribute (with data or QC field) is left to the developer. Historically the *valid_min*, *valid_max* and *valid_delta* are listed with the data field. This convention should be continued because: 1) the CF convention uses *valid_min* and *valid_max* 2) the attribute name clearly describes how the values can be used as limits 3) to continue with historical datastreams 4) the value can be understood and used without the accompanying QC field.

9.2.5 QC Test Performed Indicator

A QC bit indicates when data fails a test. By definition, the test bit is not set if the test was not performed. Some users may need to know if a test was/was not performed. This method is only valid for bit-packed QC.

To indicate if a test was/was not performed, an optional bit is defined and set when the test abandons. Only test abandonment will set the bit. This preserves the simple zero vs. non-zero QC field interpretation method.

Required additional attribute for QC field:

- *bit_<#>_test_abandoned* = “bit_<#>“; // Set to the bit number of the test unable to be performed

Excluded attribute for QCfield:

- *bit_<#>_assessment*; // This attribute is not used with a test indicator bit. The exclusion of this attribute allows for automated procedures to mask “Bad” and “Indeterminate” bit numbers only.

Example:
int qc_upwelling_broadband (time);
qc_upwelling_broadband:long_name = “Quality check results on field: Upwelling broadband radiation”;
qc_upwelling_broadband:units = “unitless”;
qc_upwelling_broadband:flag_method = “bit”;
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_1_test_abandoned = “bit_2”;
qc_upwelling_broadband:bit_2_description = “mfr10m_cosine_solar_zentih_angle test not able to be completed.”;

9.2.6 Bit-Packed Global Attribute Declaration for Quality Control

The description of each test may be listed in the global attribute section if multiple fields use the exact same bit and *description*. The description field attribute must exist, indicating that the test descriptions are listed in the global attributes. The attribute name follows the same format as the field-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 ARM Data Archive. For Standard ARM QC global attribute declarations, the existence of *valid_min*, *valid_max* or *valid_delta* data field attributes serve as indicator if the test was attempted.

Required QC field 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 valid_min";*
> *qc_bit_2_assessment = "Bad";*
> *qc_bit_3_description = "Value is greater than the valid_max";*
> *qc_bit_3_assessment = "Bad";*
> *qc_bit_4_description = "Difference between current and previous sample values exceeds valid_delta limit";*
> *qc_bit_4_assessment = "Indeterminate";*
> *qc_bit_comment = "The QC field 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."*

9.2.7 valid_min/valid_max vs. qc_min/qc_max Attribute Discussion

CF convention clearly states that *valid_min*, *valid_max,* and *valid_range* are to be used in conjunction with *_FillValue* to define the **valid** values. By definition, a non-valid value is masked 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 may be in conflict. Use of third-party software may have unintended consequences resulting in valid data being removed from the analysis. *valid_min* and *valid_max* values must be chosen carefully. If the *valid_min* and *valid_max* attribute values are intended to be used as QC limits, where the absolute exclusion of the values outside of the range defined by the two attributes would have consequences, the use of *qc_min* and *qc_max* field attributes are recommended. Complementing QC attributes and fields are updated to refer to *qc_min* and *qc_max* instead of *valid_min* and *valid_max*.

Example:
int qc_upwelling_broadband (time);
> *qc_upwelling_broadband:long_name = "Quality check results on field: Upwelling broadband radiation";*
> *qc_upwelling_broadband:units = "unitless";*
> *qc_upwelling_broadband:flag_method = "bit";*
> *qc_upwelling_broadband:bit_1_description ="Value is equal to missing_value";*
> *qc_upwelling_broadband:bit_1_assessment = "Bad";*
> *qc_upwelling_broadband:bit_2_description ="Value is less than the valid_min";*
> *qc_upwelling_broadband:bit_2_assessment = "Bad";*
> *qc_upwelling_broadband:bit_3_description = "Value is less than the qc_min";*
> *qc_upwelling_broadband:bit_3_assessment = "Bad";*
> *qc_upwelling_broadband:bit_4_description="Value is greater than the qc_max";*
> *qc_upwelling_broadband:bit_4_assessment = "Indeterminate";*

9.2.8 Multiple-Field Summarized Quality Control

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

Requirements for Multiple-Field QC field:

- QC field name is prepended with "qc_" and the base name not match any existing data field 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 = "unitless";
    qc_signal_return:flag_method = "bit";
    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";
```

9.2.9 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 field 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 field: Attenuated backscatter, co-
    polarization";
    qc_signal_return_copol:units = "unitless";
    qc_signal_return_copol:flag_method = "bit";
    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";

9.3 Integer Quality Control Fields

The optional integer QC field has the same name as the data field with the addition of a "qc" prepended the field name joined with an underscore. The standard integer QC field 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 variables names.

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

Required attribute for data field:

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

Required attributes for QC field:

- *long_name = "Quality check results on field: <field's long_name attribute value>" ;*
- *units = "unitless" ;*
- *description = "This field 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."*

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 field: Upwelling broadband radiation" ;
  qc_upwelling_broadband:units = "unitless" ;
  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” ;

9.3.1 Integer Global Attribute Declaration for Quality Control

The description of each test may be listed in the global attribute section if multiple fields use the exact same flag number and description. The *description* field attribute must exist, indicating that the test descriptions are listed in the global attributes. The attribute name follows the same format as the field-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 ARM Data Archive. For Standard ARM QC global attribute declarations, the existence of *valid_min*, *valid_max* or *valid_delta* data field attributes serve as indicator if the test was attempted.

Required QC field 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 valid_min";
qc_flag_2_assessment = "Bad" ;
qc_flag_3_description = "Value is greater than the valid_max" ;
qc_flag_3_assessment = "Bad" ;
qc_flag_4_description = "Difference between current and previous sample values exceeds valid_delta limit" ;
qc_flag_4_assessment = "Indeterminate" ;
qc_flag_comment = "The QC field 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." ;

9.4 Ancillary Quality Control Fields

The “aqc” convention can be used to allow for the inclusion of QC fields that cannot be updated to meet the bit-packed or integer QC format. The required *long_name* and *units* field attributes also apply to “aqc”.

One must be reasonable when choosing to use *aqc_<field>* instead of *qc_<field>*. The primary reason for choosing to use ancillary QC fields is to preserve the original format. There is no standard format for “aqc” other than the required *long_name* and *units* attributes.

10.0 Guidelines to Describe Source

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

10.1 Source Field Attribute - Time Independent

If the source does not change, the input is indicated with an optional *source* data field attribute. Enough information to fully trace the values must be provided with a syntax of *"<datastream_name>:<field_name>"*. Multiple sources may be listed, separated by a single-space character. The source attribute may optionally describe a method or algorithm instead of an input *datastream:field*. If no source was used, then set attribute to *"no_source_available"*.

Example:

- ARM datastream and field name:
  - *source = "sgpmetE13.b1:atmospheric_temperature" ;*
  - *source = "sgpmwrC1.b1:vap sgpmwrpC1.b1:vapor" ;*
- Algorithm:
  - *source = "myers_briggs" ;*
  - *source = "rutherfurd_1.2" ;*
  - *source = "calvin_3.2 hobbs_1" ;*

10.2 Source Field - Time Dependent

For describing a time dependent source, an optional source field is used. If the source field is referenced by one data fieldb it is recommended to use the data field name preceded by "source" and joined to the data field name with an underscore (i.e., *source_atmos_temperature* for *atmos_temperature*). If a source field is used for multiple data fields it is recommended to use a name different than any of the data fields (i.e., *source_atmos_state* for *atmos_temperature, atmos_pressure, relative_humidy, wind_speed and wind_direction*).

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

10.3 Source Field - Flag Method

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

Required attribute for data field:

- *ancillary_variables* = <source field name>;

Required attributes for source field:

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

Optional attribute for source field:

- 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 field 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 method used for reading data. Some implementations may convert attribute names to program variables. A "-" character is not allowed in most programming language variables names.

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

Example:

```
float aod (time) ;
    aod:long_name = "Aerosol optical depth" ;
    aod:units = "unitless" ;
    aod:missing_value = -9999.f ;
    aod:ancillary_variables = "source_aod" ;
int source_aod(time) ;
    source_aod:long_name = "Source for field: Aerosol optical depth" ;
    source_aod:units = "unitless" ;
    source_aod:flag_method = "integer";
    source_aod:description = "This field contains integer values which should be interpreted as listed." ;
    source_aod:flag_0_description = "no_source_available" ;
    source_aod:flag_1_description = "sgpmfrC1.c1:aerosol_optical_depth" ;
    source_aod:flag_2_description = "sgpmfrsrC1.b1:aerosol_optical_depth" ;
    source_aod:flag_2_comment = "Fill gaps of 3 days or less via interpolation" ;
    source_aod:flag_3_description ="sgpnimfraod1michC1.c1:aod" ;
    source_aod:flag_4_description ="sgpnimfraod1michE13.c1:aod" ;
    source_aod:flag_5_description ="sgpnimfraod1michE13.c1:aod sgpnimfraod1michC1.c1:aod" ;
```

10.4 Source Field - 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, the use of the bit-packed method is recommended. Only one method of indicator is allowed at a time (no mixing of integer and bit-packed values in the same field). The use of this type of method is indicated by the use of the *flag_method* field attribute.

Required field attribute:

- *long_name = "Source of field: <data field's long_name attribute value or generic description if used for multiple fields>" ;*
- *units = "unitless" ;*
- *description = "This field 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" ;*

Example:

```
int source_aod (time) ;
    source_aod:long_name = "Source for field: Aerosol optical depth" ;
    source_aod:units = "unitless";
    source_aod:description = "This field 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 = "sgpmfrsrC1.c1:aerosol_optical_depth" ;
    source_aod:bit_2_description = "sgpmfrsrC1.b1:aerosol_optical_depth" ;
    source_aod:bit_2_comment = "Fill gaps of 3 days or less via interpolation" ;
    source_aod:bit_3_description = "sgpnimfraod1michC1.c1:aod" ;
    source_aod:bit_4_description = "sgpnimfraod1michE13.c1:aod" ;
```

11.0 Process for Evaluating Exceptions

This section describes the ARM data standards exception request process.

11.1 Identifying Exceptions

There are two primary methods used to identify exceptions from the required standards. The first method involves the use of the ARM Process Configuration Management (PCM) tool at https://engineering.arm.gov/pcm/Main.html. The PCM tool is used by ingest and VAP developers, and allows them to 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 are simple visual inspection of data products by members of the ARM Data Management Facility (DMF), ARM Data Quality Office, ARM Data Archive, 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.

11.2 Exception Request

During the development of an instrument ingest or VAP (http://www.arm.gov/publications/tech_reports/doe-sc-arm-tr-093.pdf), 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 ARM Data Archive
- representative from the DQ Office
- 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 ARM Data Archive members will only serve an initial one-year term. After the initial year, the members will serve a two-year term.

11.3 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 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.

11.4 Options for the Standards Committee

11.4.1 Approval

If the request is approved, this means 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 DQR, indicating 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 ARM Data Archive data discovery tools.

11.4.2 Denial

If the request is denied, this indicates the committee believes that the benefit to conforming to the standards justifies the cost of doing so and 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).

11.4.3 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.

11.4.4 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.

11.4.5 2015 Standards Committee

Table 2. 2015 Standards Committee.

Group	Primary Member	Incoming Member*
VAP Manager	Chitra Sivaraman (PNNL)	Brian Ermold (PNNL)
Metadata QC Reviewer	Alice Cialella (BNL)	Rick Wagener (BNL)
ARM Data Archive	Giri Palanisamy (ORNL)	Harold Shanafield (ORNL)
DQ Office	Ken Kehoe (OU) [Chair]	Josh King (OU)
Translator/Mentor	Laura Riihimaki (PNNL)	Jenni Kyrouac (ANL)

* The incoming member will assume the role of primary member after a term except for the VAP manager who will not step down from the primary member status. A new incoming member will be chosen every cycle.

11.5 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 Climate Research 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
<field>	A general placeholder for a field name
ADI	ARM Data Integrator (Formerly known as ISDE)
AGL	Above Ground Level
AMF	ARM Mobile Facility
ARM	Atmospheric Radiation Measurement
CF	Climate and Forecast
Developer	Person responsible for software development
DOD	Data Object Design
DOI	Digital Object Identifier
DQR	Data Quality Report
ECO	Engineering Change Order
ECR	Engineering Change Request
EWO	Engineering Work Order
HDF	Hierarchical Data Format
IATA	International Air Transport Association
Instrument Class	A convenient name for a grouping of specific instruments which share important str°uctural and physical properties. E.g., 1ebbr, 5ebbr, 30ebbr, 1440ebbr instruments [instrument_codes] all belong to the “ebbr” instrument class.
Instrument	A single piece of hardware or group of hardware that records a measurement
IOP	Intensive Operational Period
MAOS	Mobile Aerosol Observing System
Mentor	Person responsible for instrument installation and general operations
MPEG	Moving Picture Expert Group
MSL	Mean Sea Level
NaN	Not a number indicator
PCM	Process Configuration Management
PI	Principal Investigator
PNG	Portable Network Graphics
QC	Quality Control
RAW	Data file created by instrument
SGP	Southern Great Plains
Site	Geographical Region
Translator	Person responsible for VAP development and maintenance
UTC	Coordinated Universal Time
VAP	Value-Added Product

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 of the following XML files:

- SI unit prefixes
- SI base units
- SI derived units
- Units accepted for use with the SI
- Non-SI units

Base Units

Table 3. Base units.

Base Quantity	Unit Name	Symbol	Comment
Length, distance, height	meter	m	
Mass	gram	g	
Time	second	s	
	minute	min	
	hour	h	hr also used
	day	d	day also used
	Gregorian year	a	exactly 365.242198781 d, yr also used
Temperature, thermodynamic or absolute, brightness temperature	Kelvin	K	

Derived Units, First Order

Table 4. Derived units, first order.

Base Quantity	Unit Name	Symbol	Comment
Frequency, sample rate	hertz	Hz, s^-1	
Force	newton	N	
Energy	joule	J	
Power	watt	W	
Electric Potential, voltage	volt	V	common in uncalibrated quantities
Electrical Resistance	ohm	ohm	since SI symbol Capital Omega cannot be easily represented

Commonly Used Derived Units

Table 5. Commonly used derived units

Base Quantity	Unit Name	Symbol	Comment
Atmospheric pressure, barometric pressure, station pressure	kilopascal	kPa	use hPa only to replace old mbar, do not use mbar
Density, water vapor density, absolute humidity, concentration of trace substance	gram per cubic meter	g/m^3	
Energy Flux Density, irradiance, heat flux, net radiation	watt per square meter	W/m^2	
Plane angle, azimuth, elevation, wind direction, zenith	degree	degree	radian (rad) is sometimes used but not common in atmospheric science
Latitude	degree north	degree_N	
Longitude	degree east	degree_E	

Base Quantity	**Unit Name**	**Symbol**	**Comment**
Precipitable water vapor	centimeter	cm	mm also acceptable
Precipitation	millimeter	mm	hundredths of inches also used but less preferred
Precipitation rate	millimeter per second	mm/s	
Radiance	watt per square meter per steradian	W m^-2 sr^-1	W/(m^2 sr) also acceptable
Relative humidity	percent	%	fraction (unitless) also used but less preferred
Solid angle	steradian	sr	
Temperature, dry bulb, wet bulb, dewpoint, potential, equivalent potential, virtual	celsius	degC	
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 also used but less preferred, do not use milibar (mbar)
Wavelength	nanometer	nm	micrometer (um) also used but less preferred
Wavenumber	inverse centimeter	cm^-1	

Odd and Ends

Table 6. Odds and ends.

Base Quantity	**Unit Name**	**Symbol**	**Comment**
Bins		unitless	
Mass density	gram per cubic centimeter	g/cm^3,	
Number density	inverse cubic centimeter	1/cm^3	
Molar mixing ratio	micro-mol per mol	umol/mol	
Volumetric mixing ratio	parts per million by volume	ppmV	
Counts		count	
Ratio, fraction	fraction	unitless	% allowed but less preferred for ratios
Probability	fraction	unitless	
Relative power		dB	mostly used for radar return signals
Soil moisture content by volume	cubic meter per cubic meter	m^3/m^3	
Soil water potential	kilopascal	kPa	

Prefixes

Table 7. Prefixes.

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

Appendix D
ARM netCDF Data File Example

```
data file name = 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 field: Atmospheric
        temperature” ;
        qc_atmospheric_temperature:units = “unitless” ;
        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 field: Atmospheric temperature” ;
        source_atmospheric_temperature:units = “unitless”;
        source_atmospheric_temperature:description = “This field 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 = “unitless” ;
        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" ;
    float lat ;
        lat:long_name = "North latitude" ;
        lat:units = "degree_N" ;
        lat:standard_name = "latitude" ;
        lat:missing_value = -9999. ;
        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:missing_value = -9999.f ;
        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” ;
        alt:missing_value = -9999.f ;

// global attributes:
:command_line = "tempprofile -d 20130101 -f sgp.C1" ; ;
:Conventions =”ARM_Convention-1.0 CF-1.6” ;
:process_version = "ingest-met-4.10-0.el5" ;
```

:dod_version = "tempprofile-b1-2.0" ;
:input_datastreams = "sgpsondewnpnC1.b1 : 6.1 : 20130101 ;\n sgpaeriprofC1.c1 : 1.1 : 20130101.000000 ;\n sgp1290rwpC1.c1: Release_1_4 : 20130101.000000 ;\n conwarfX1.a1 : Release_2_9 : 20130101.000000" ;
:site_id = "sgp" ;
:platform_id = "tempprofile" ;
:facility_id = "C1" ;
:data_level = "c1" ;
:location_description = "Southern Great Plains (SGP), Lamont, OK (C1) ;
:datastream = "sgptempprofileC1.c1" ;
:title = "Atmospheric Radiation Measurement (ARM) program best estimate of atmospheric temperature profile" ;
:institution = "United States Department of Energy - Atmospheric Radiation Measurement (ARM) program"
:description = "Best estimate of atmospheric temperature profile over Lamont, OK" ;
:references = "http://www.arm.gov/data/vaps/" ;
:history = "created by user dsmgr on machine ruby at 1-Jan-2007,2:43:02" ;