DOE/SC-ARM-TR-136

Introduction to Reading and Visualizing ARM Data

J Mather

March 2013

Contents

1.0 Introduction

Atmospheric Radiation Measurement (ARM) Program standard data format is NetCDF 3 (Network Common Data Form). The object of this tutorial is to provide a basic introduction to NetCDF with an emphasis on aspects of the ARM application of NetCDF. The goal is to provide basic instructions for reading and visualizing ARM NetCDF data with the expectation that these examples can then be applied to more complex applications.

1.1 The NetCDF Data Format

Information about NetCDF and NetCDF libraries are available from the Unidata NetCDF web site:

http://www.unidata.ucar.edu/software/netcdf/

From the Unidata web page, "NetCDF is a set of software libraries and self-describing, machine-independent data formats that support the creation, access, and sharing of array-oriented scientific data. In addition to the original Fortran and C libraries, many languages provide interfaces to NetCDF. Popular data analysis packages used around ARM and ASR are Matlab and interactive data language (IDL). These are both commercial packages– but a free (or inexpensive for some distributions) high-level language supporting Python seeing growing use in ARM is Python.

http://www.python.org

1.2 Python Packages used in this Tutorial

This tutorial will make use of Python because of its growing popularity and because it can be obtained without cost – particularly for students. There are many scientific data analysis packages available for Python. This tutorial will make use of several key packages:

netcdf4-python (also supports NetCDF 3):

http://netcdf4-python.googlecode.com/svn/trunk/docs/netCDF4-module.html

and matplotlib (for data visualization):

http://matplotlib.org

These packages can be installed from source code. Instructions for doing this can be found on a number of sites for example, here is a site describing the installation process for matplotlib on a Mac including dependencies:

http://www.astro.washington.edu/users/rowen/BuildingMatplotlibForMac.html

However, an easier approach is to make use of one of several scientific python distributions:

Enthought Python Distribution (EPD):

http://www.enthought.com/products/epd.php

Or

Python (x,y): http://code.google.com/p/pythonxy/

For this tutorial we are using the Enthought distribution which is available for Windows, Mac, Redhat and Ubuntu Linux and is free for students and staff at degree-granting institutions. **It is assumed for the rest of this tutorial that you have installed the EPD distribution of Python or an equivalent set of packages.**

1.3 Loading ARM data into the EPD Python Interface

Before beginning, we will be working with a fairly simple data file, surface meteorological data from the North Slope of Alaska site in Barrow:

nsametC1.b1.20130210.000000.cdf

If you are unfamiliar with ARM file name conventions, a detailed description can be found at: http://www.arm.gov/data/docs/plan#naming. In this example file, the first three characters indicate the site, “nsa” = North Slope of Alaska; the next three characters indicate the type of measurements contained in the file, “met” = surface meteorological parameters; C1 indicates the specific facility at the North Slope of Alaska – “C1” = Barrow. The next two characters indicate the level of data processing, “b1” indicates that basic data quality checks have been applied to the data. Finally, the numeric string indicates the date and time of the beginning of the file in the form: yyyymmdd.hhmmss.

To prepare to work with this data file, create a directory, e.g. ~/tutorial/ in your home directory or another convenient location and place the sample meteorological data in this directory.

To begin using the EPD python distribution, first execute the Pylab utility from the Enthought folder (note: if you are using the Python(x, y) distribution instead of EPD, the Python environment utility is called the “IPython Qt Console”). This should create a terminal window. Within this window, change directories to the data directory:

```
: cd ~/tutorial
```

You can open this data from the Pylab terminal as follows:

First load the python-netcdf package
```
:  from netCDF4 import Dataset
```

You can then load the meteorology data
```
: nc_file = Dataset(‘nsametC1.b1.20130210.000000.cdf’, ‘r’, format = ‘NETCDF3_CLASSIC’)
```

The variable “nc_file” will store information about the NetCDF file. The argument, r, indicates we will be opening the file for reading only while the specification of NETCDF3_CLASSIC indicates that the ARM data use the older data format.

To see if the operation was successful, use the Python command “whos”

```
: whos
```

At any time you can use the “whos” command to get a snapshot of what variables are loaded in the Pylab environment. Now you should see something like:

```
Variable   Type       Data/Info
-------------------------------
Dataset    type       <type 'netCDF4.Dataset'>
nc_file    Dataset    <netCDF4.Dataset object at 0x7706c30>
```

Before we can do anything with this file, we have to find out what variables are available in the nsamet file. We will do this using the netcdf4-python interface. However – as a side note – even if you use a complete analysis package like EPD or a commercial version, it is a very good idea to also download and build your own NetCDF libraries from Unidata. In addition to giving you the opportunity to giving you more flexibility for installing NetCDF applications, this process also comes with the simple but powerful utility “ncdump” which allows you to find quickly a great deal of information about the contents of a NetCDF file. The descriptive header information about our file from ncdump is provided as an Appendix at the end of this document.

To get an unformatted list of variables in the nsamet file type

```
: print nc_file.variables
```

We have also provided a Python script called “ncdumpy” that approximates “ncdump”. You can get information about file variables with that script by typing

```
: run ncdumpy  nsametC1.b1.20130210.000000.cdf
```

We will start with the mean surface air temperature, “temp_mean”. We can learn more about this by first assigning this variable:

```
: temp_mean = nc_file.variables[‘temp_mean’]
```

We can learn more about this variable such as its dimension and size

```
: print temp_mean.dimensions
```

Here we learn that temperature is a 1-dimensional variable depending only on time

```
: print temp_mean.shape
```

This tells us that there are 1440 time records (note that 1440 is typical because it represents the number of minutes in a day and many ARM data have a time resolution of 1-minute).

We can actually load the temperature data and print it using the following

```
: tm_val = temp_mean[:]
```

1.4 Creating a Basic Plot of 1-Dimensional ARM Data

To plot these values, we need to load the matplotlib package:

```
: from pylab import *
```

You can now plot these values:

```
: plot(tm_val)
```

You should see values ranging from -32 to -24 C. You can save this plot to an output file. The following will do this to a PNG format file named "testplot.png'.

```
:  savefig('testplot.png',format='png')
```

This simple example, plots temperature as a function of record number only. To plot as a function of time, you will also need to import a time variable. In our sample NetCDF file, as in many ARM data files, there are three time variables: base_time, time_offset, and time.

"base_time" is a scalar variable and reports the "Epoch" time or the number of seconds from 00:00 GMT (Greenwich Mean Time, or Universal Time), January 1, 1970 to the time stamp of the data file (recorded in the file name). Using base_time is awkward at first but provides a consistent way of dealing with absolute time (versus time from the start of the file) which is useful for concatenating files for example.

"time_offset" is a 1-dimensional variable reporting the number of seconds from base_time to each data record.

"time" is a 1-dimensional variable defined here as the number of seconds since midnight GMT. Typically ARM data files span the time 00:00 to 23:59 GMT (Greenwich Mean Time, equivalent to UT or Universal Time). Please note that while in this example we will use the "time" variable, there is discussion ongoing to use the variable "time" in other ways to conform to other international standards so pay attention to the definition of time-related variables. An example of using the base_time and time_offset is given in the Python script: convert_time.py shown in Appendix B. If you run this sample file you should see the following output:

```
: run convert_time
    base_time = 1360454400 and time_offset of the first record is 0.0
    The date is: 2/10/2013
    And the time (GMT) is 0:0:0.
```

To access the time data, load the time variable as the temperature was loaded previously.

```
: time_obj = nc_file.variables['time']
```

Next load the values into the workspace

```
: time = time_obj[:]
```

Before redoing the temperature plot, convert the time to a more useful form – time in hours.

```
: time_hour = time/3600
```

Now redo the plot

```
: plot(time_hour, tm_val)
```

Now so far we have been doing all this from the command line. But like most (perhaps all) high level languages it is often more convenient to assemble commands in a script and run the commands in a batch. You have been provided a simple plotting script entitled "plot_temperature.py" that executes the already discussed commands but additionally adds axes labels. This script can easily be adapted to other plotting applications. To run the script from the PyLab prompt simply type

```
: run plot_temperature
```

This script is provided in Appendix C.

1.5 Additional Resources

Note there are many plotting examples provided at the matplotlib web site under their "Gallery" tab: http://matplotlib.org/gallery.html. In addition to 1-dimensional plots, examples include histograms, contour plots, pie charts, images etc.

There is also a great deal of information on-line for using the various Python scientific libraries. Good resources include:

Standard Python Documentation: http://www.python.org/doc

A tutorial for manipulating arrays: http://www.scipy.org/Tentative_NumPy_Tutorial

Plotting in Python: http://matplotlib.org/ and Appendix D.

And for Matlab users, a convenient table relating Matlab and Python operations: http://www.scipy.org/NumPy_for_Matlab_Users

There is also information about ARM-specific data issues available on the ARM web pages and we plan to add to this information. Some examples of available information include:

Information about Time: http://www.arm.gov/data/time

And bit-packed Quality Control flags: https://engineering.arm.gov/~shippert/ARM_bits.html. Regarding the quality control flags, ARM has a standard methodology for coding quality control information. This reference provides detailed information on understanding and reading these fields.

For additional information about ARM including measurements, sites, publications, and ordering data, please explore the ARM website: http://www.arm.gov/ and if you have any questions, please let us know: http://www.arm.gov/about/contact.

Appendix A
Header Contents of Sample Data File using ncdump

If you have installed NetCDF libraries, you can use the following command to see the header of our NetCDF file:

```
ncdump –h nsametC1.b1.20130210.000000.cdf
```

specifying the –h option limits the output to the header information only. Omitting the –h option results in the header and all the data being printed. It is also possible to print one or more variables using the –v option:

```
ncdump –v <var_1>,<var_2>,…<var_n> nsametC1.b1.20130210.000000.cdf
```

Here is the output of the header only for the example data file. The header includes a list of variables and a description of a set of attributes attached to each variable. These attributes include a description of the variable referred to as the “long name”, the variable units, the acceptable range, and the standard missing value. At the end of the header are a set of Global Attributes that apply to the entire file. These attributes include information like the software used to process the file and the location of the site where the data were collected.

```
netcdf nsametC1.b1.20130209.000000 {
dimensions:
    time = UNLIMITED ; // (1440 currently)
variables:
    int base_time ;
        base_time:string = "9-Feb-2013,0:00:00 GMT" ;
        base_time:long_name = "Base time in Epoch" ;
        base_time:units = "seconds since 1970-1-1 0:00:00 0:00" ;
    double time_offset(time) ;
        time_offset:long_name = "Time offset from base_time" ;
        time_offset:units = "seconds since 2013-02-09 00:00:00 0:00" ;
    double time(time) ;
        time:long_name = "Time offset from midnight" ;
        time:units = "seconds since 2013-02-09 00:00:00 0:00" ;
    int qc_time(time) ;
        qc_time:long_name = "Quality check results on field: Time offset from midnight" ;
        qc_time:units = "unitless" ;
        qc_time:description = "This field contains bit packed values which should be interpreted as listed.
No bits set (zero) represents good data." ;
        qc_time:bit_1_description = "Delta time between current and previous samples is zero." ;
        qc_time:bit_1_assessment = "Indeterminate" ;
        qc_time:bit_2_description = "Delta time between current and previous samples is less than the
delta_t_lower_limit field attribute." ;
        qc_time:bit_2_assessment = "Indeterminate" ;
        qc_time:bit_3_description = "Delta time between current and previous samples is greater than the
delta_t_upper_limit field attribute." ;
        qc_time:bit_3_assessment = "Indeterminate" ;
```

```
		qc_time:delta_t_lower_limit = 60. ;
		qc_time:delta_t_upper_limit = 60. ;
		qc_time:prior_sample_flag = 1 ;
		qc_time:comment = "If the \'prior_sample_flag\' is set the first sample time from a new raw file
will be compared against the time just previous to it in the stored data. If it is not set the qc_time value for
the first sample will be set to 0." ;
	float atmos_pressure(time) ;
		atmos_pressure:long_name = "Atmospheric pressure" ;
		atmos_pressure:units = "kPa" ;
		atmos_pressure:valid_min = 80.f ;
		atmos_pressure:valid_max = 110.f ;
		atmos_pressure:valid_delta = 1.f ;
		atmos_pressure:missing_value = -9999.f ;
	int qc_atmos_pressure(time) ;
		qc_atmos_pressure:long_name = "Quality check results on field: Atmospheric pressure" ;
		qc_atmos_pressure:units = "unitless" ;
		qc_atmos_pressure:description = "See global attributes for individual bit descriptions." ;
	float temp_mean(time) ;
		temp_mean:long_name = "Temperature mean" ;
		temp_mean:units = "C" ;
		temp_mean:valid_min = -60.f ;
		temp_mean:valid_max = 30.f ;
		temp_mean:valid_delta = 10.f ;
		temp_mean:missing_value = -9999.f ;
	int qc_temp_mean(time) ;
		qc_temp_mean:long_name = "Quality check results on field: Temperature mean" ;
		qc_temp_mean:units = "unitless" ;
		qc_temp_mean:description = "See global attributes for individual bit descriptions." ;
	float temp_std(time) ;
		temp_std:long_name = "Temperature standard deviation" ;
		temp_std:units = "C" ;
	float rh_mean(time) ;
		rh_mean:long_name = "Relative humidity mean" ;
		rh_mean:units = "%" ;
		rh_mean:valid_min = 0.f ;
		rh_mean:valid_max = 104.f ;
		rh_mean:valid_delta = 30.f ;
		rh_mean:missing_value = -9999.f ;
	int qc_rh_mean(time) ;
		qc_rh_mean:long_name = "Quality check results on field: Relative humidity mean" ;
		qc_rh_mean:units = "unitless" ;
		qc_rh_mean:description = "See global attributes for individual bit descriptions." ;
	float rh_std(time) ;
		rh_std:long_name = "Relative humidity standard deviation" ;
		rh_std:units = "%" ;
	float vapor_pressure_mean(time) ;
		vapor_pressure_mean:long_name = "Vapor pressure mean, calculated" ;
		vapor_pressure_mean:units = "kPa" ;
		vapor_pressure_mean:valid_min = 0.001f ;
		vapor_pressure_mean:valid_max = 4.3f ;
		vapor_pressure_mean:valid_delta = 1.f ;
		vapor_pressure_mean:missing_value = -9999.f ;
	int qc_vapor_pressure_mean(time) ;
		qc_vapor_pressure_mean:long_name = "Quality check results on field: Vapor pressure mean,
calculated" ;
		qc_vapor_pressure_mean:units = "unitless" ;
		qc_vapor_pressure_mean:description = "See global attributes for individual bit descriptions." ;
	float vapor_pressure_std(time) ;
		vapor_pressure_std:long_name = "Vapor pressure standard deviation" ;
		vapor_pressure_std:units = "kPa" ;
	float wspd_arith_mean(time) ;
		wspd_arith_mean:long_name = "Wind speed arithmetic mean" ;
		wspd_arith_mean:units = "m/s" ;
		wspd_arith_mean:valid_min = 0.f ;
		wspd_arith_mean:valid_max = 100.f ;
		wspd_arith_mean:valid_delta = 20.f ;
		wspd_arith_mean:missing_value = -9999.f ;
	int qc_wspd_arith_mean(time) ;
		qc_wspd_arith_mean:long_name = "Quality check results on field: Wind speed arithmetic mean"
;
		qc_wspd_arith_mean:units = "unitless" ;
		qc_wspd_arith_mean:description = "See global attributes for individual bit descriptions." ;
	float wspd_vec_mean(time) ;
		wspd_vec_mean:long_name = "Wind speed vector mean" ;
		wspd_vec_mean:units = "m/s" ;
		wspd_vec_mean:valid_min = 0.f ;
		wspd_vec_mean:valid_max = 100.f ;
		wspd_vec_mean:valid_delta = 20.f ;
		wspd_vec_mean:missing_value = -9999.f ;
	int qc_wspd_vec_mean(time) ;
		qc_wspd_vec_mean:long_name = "Quality check results on field: Wind speed vector mean" ;
		qc_wspd_vec_mean:units = "unitless" ;
		qc_wspd_vec_mean:description = "See global attributes for individual bit descriptions." ;
	float wdir_vec_mean(time) ;
		wdir_vec_mean:long_name = "Wind direction vector mean" ;
		wdir_vec_mean:units = "deg" ;
		wdir_vec_mean:valid_min = 0.f ;
		wdir_vec_mean:valid_max = 360.f ;
		wdir_vec_mean:missing_value = -9999.f ;
	int qc_wdir_vec_mean(time) ;
		qc_wdir_vec_mean:long_name = "Quality check results on field: Wind direction vector mean" ;
		qc_wdir_vec_mean:units = "unitless" ;
		qc_wdir_vec_mean:description = "See global attributes for individual bit descriptions." ;
	float wdir_vec_std(time) ;
		wdir_vec_std:long_name = "Wind direction vector mean standard deviation" ;
		wdir_vec_std:units = "deg" ;
		wdir_vec_std:missing_value = -9999.f ;
	int pwd_err_code(time) ;
		pwd_err_code:long_name = "PWD alarm" ;
		pwd_err_code:units = "unitless" ;
		pwd_err_code:missing_value = -9999 ;
	int pwd_mean_vis_1min(time) ;
```

```
		pwd_mean_vis_1min:long_name = "PWD 1 minute mean visibility" ;
		pwd_mean_vis_1min:units = "m" ;
		pwd_mean_vis_1min:valid_min = 0 ;
		pwd_mean_vis_1min:valid_max = 20000 ;
		pwd_mean_vis_1min:missing_value = -9999 ;
	int qc_pwd_mean_vis_1min(time) ;
		qc_pwd_mean_vis_1min:long_name = "Quality check results on field: PWD 1 minute mean
visibility" ;
		qc_pwd_mean_vis_1min:units = "unitless" ;
		qc_pwd_mean_vis_1min:description = "See global attributes for individual bit descriptions." ;
	int pwd_mean_vis_10min(time) ;
		pwd_mean_vis_10min:long_name = "PWD 10 minute mean visibility" ;
		pwd_mean_vis_10min:units = "m" ;
		pwd_mean_vis_10min:valid_min = 0 ;
		pwd_mean_vis_10min:valid_max = 20000 ;
		pwd_mean_vis_10min:missing_value = -9999 ;
	int qc_pwd_mean_vis_10min(time) ;
		qc_pwd_mean_vis_10min:long_name = "Quality check results on field: PWD 10 minute mean
visibility" ;
		qc_pwd_mean_vis_10min:units = "unitless" ;
		qc_pwd_mean_vis_10min:description = "See global attributes for individual bit descriptions." ;
	int pwd_pw_code_inst(time) ;
		pwd_pw_code_inst:long_name = "PWD instantaneous present weather code" ;
		pwd_pw_code_inst:units = "unitless" ;
		pwd_pw_code_inst:valid_min = 0 ;
		pwd_pw_code_inst:valid_max = 99 ;
		pwd_pw_code_inst:missing_value = -9999 ;
	int qc_pwd_pw_code_inst(time) ;
		qc_pwd_pw_code_inst:long_name = "Quality check results on field: PWD instantaneous present
weather code" ;
		qc_pwd_pw_code_inst:units = "unitless" ;
		qc_pwd_pw_code_inst:description = "See global attributes for individual bit descriptions." ;
	int pwd_pw_code_15min(time) ;
		pwd_pw_code_15min:long_name = "PWD 15 minute present weather code" ;
		pwd_pw_code_15min:units = "unitless" ;
		pwd_pw_code_15min:valid_min = 0 ;
		pwd_pw_code_15min:valid_max = 99 ;
		pwd_pw_code_15min:missing_value = -9999 ;
	int qc_pwd_pw_code_15min(time) ;
		qc_pwd_pw_code_15min:long_name = "Quality check results on field: PWD 15 minute present
weather code" ;
		qc_pwd_pw_code_15min:units = "unitless" ;
		qc_pwd_pw_code_15min:description = "See global attributes for individual bit descriptions." ;
	int pwd_pw_code_1hr(time) ;
		pwd_pw_code_1hr:long_name = "PWD 1 hour present weather code" ;
		pwd_pw_code_1hr:units = "unitless" ;
		pwd_pw_code_1hr:valid_min = 0 ;
		pwd_pw_code_1hr:valid_max = 99 ;
		pwd_pw_code_1hr:missing_value = -9999 ;
	int qc_pwd_pw_code_1hr(time) ;
```

```
		qc_pwd_pw_code_1hr:long_name = "Quality check results on field: PWD 1 hour present weather
code" ;
		qc_pwd_pw_code_1hr:units = "unitless" ;
		qc_pwd_pw_code_1hr:description = "See global attributes for individual bit descriptions." ;
	float pwd_precip_rate_mean_1min(time) ;
		pwd_precip_rate_mean_1min:long_name = "PWD 1 minute mean precipitation rate" ;
		pwd_precip_rate_mean_1min:units = "mm/hr" ;
		pwd_precip_rate_mean_1min:valid_min = 0.f ;
		pwd_precip_rate_mean_1min:valid_max = 999.99f ;
		pwd_precip_rate_mean_1min:valid_delta = 100.f ;
		pwd_precip_rate_mean_1min:missing_value = -9999.f ;
	int qc_pwd_precip_rate_mean_1min(time) ;
		qc_pwd_precip_rate_mean_1min:long_name = "Quality check results on field: PWD 1 minute
mean precipitation rate" ;
		qc_pwd_precip_rate_mean_1min:units = "unitless" ;
		qc_pwd_precip_rate_mean_1min:description = "See global attributes for individual bit
descriptions." ;
	float pwd_cumul_rain(time) ;
		pwd_cumul_rain:long_name = "PWD cumulative liquid precipitation" ;
		pwd_cumul_rain:units = "mm" ;
		pwd_cumul_rain:valid_min = 0.f ;
		pwd_cumul_rain:valid_max = 99.99f ;
		pwd_cumul_rain:valid_delta = 50.f ;
		pwd_cumul_rain:missing_value = -9999.f ;
	int qc_pwd_cumul_rain(time) ;
		qc_pwd_cumul_rain:long_name = "Quality check results on field: PWD cumulative liquid
precipitation" ;
		qc_pwd_cumul_rain:units = "unitless" ;
		qc_pwd_cumul_rain:description = "See global attributes for individual bit descriptions." ;
	float pwd_cumul_snow(time) ;
		pwd_cumul_snow:long_name = "PWD cumulative snow" ;
		pwd_cumul_snow:units = "mm" ;
		pwd_cumul_snow:valid_min = 0.f ;
		pwd_cumul_snow:valid_max = 999.f ;
		pwd_cumul_snow:valid_delta = 100.f ;
		pwd_cumul_snow:missing_value = -9999.f ;
	int qc_pwd_cumul_snow(time) ;
		qc_pwd_cumul_snow:long_name = "Quality check results on field: PWD cumulative snow" ;
		qc_pwd_cumul_snow:units = "unitless" ;
		qc_pwd_cumul_snow:description = "See global attributes for individual bit descriptions." ;
	float cmh_temp(time) ;
		cmh_temp:long_name = "CMH temperature" ;
		cmh_temp:units = "C" ;
		cmh_temp:valid_min = -60.f ;
		cmh_temp:valid_max = 30.f ;
		cmh_temp:valid_delta = 10.f ;
		cmh_temp:missing_value = -9999.f ;
	int qc_cmh_temp(time) ;
		qc_cmh_temp:long_name = "Quality check results on field: CMH temperature" ;
		qc_cmh_temp:units = "unitless" ;
		qc_cmh_temp:description = "See global attributes for individual bit descriptions." ;
	float cmh_dew_point(time) ;
		cmh_dew_point:long_name = "CMH dew point" ;
		cmh_dew_point:units = "C" ;
		cmh_dew_point:valid_min = -60.f ;
		cmh_dew_point:valid_max = 30.f ;
		cmh_dew_point:valid_delta = 10.f ;
		cmh_dew_point:missing_value = -9999.f ;
	int qc_cmh_dew_point(time) ;
		qc_cmh_dew_point:long_name = "Quality check results on field: CMH dew point" ;
		qc_cmh_dew_point:units = "unitless" ;
		qc_cmh_dew_point:description = "See global attributes for individual bit descriptions." ;
	float cmh_sat_vapor_pressure(time) ;
		cmh_sat_vapor_pressure:long_name = "CMH saturation vapor pressure, calculated" ;
		cmh_sat_vapor_pressure:units = "kPa" ;
		cmh_sat_vapor_pressure:valid_min = 0.001f ;
		cmh_sat_vapor_pressure:valid_max = 4.3f ;
		cmh_sat_vapor_pressure:valid_delta = 1.f ;
		cmh_sat_vapor_pressure:missing_value = -9999.f ;
	int qc_cmh_sat_vapor_pressure(time) ;
		qc_cmh_sat_vapor_pressure:long_name = "Quality check results on field: CMH saturation vapor
pressure, calculated" ;
		qc_cmh_sat_vapor_pressure:units = "unitless" ;
		qc_cmh_sat_vapor_pressure:description = "See global attributes for individual bit descriptions." ;
	float cmh_vapor_pressure(time) ;
		cmh_vapor_pressure:long_name = "CMH vapor pressure, calculated" ;
		cmh_vapor_pressure:units = "kPa" ;
		cmh_vapor_pressure:valid_min = 0.001f ;
		cmh_vapor_pressure:valid_max = 4.3f ;
		cmh_vapor_pressure:valid_delta = 1.f ;
		cmh_vapor_pressure:missing_value = -9999.f ;
	int qc_cmh_vapor_pressure(time) ;
		qc_cmh_vapor_pressure:long_name = "Quality check results on field: CMH vapor pressure,
calculated" ;
		qc_cmh_vapor_pressure:units = "unitless" ;
		qc_cmh_vapor_pressure:description = "See global attributes for individual bit descriptions." ;
	float cmh_rh(time) ;
		cmh_rh:long_name = "CMH relative humidity, calculated" ;
		cmh_rh:units = "%" ;
		cmh_rh:valid_min = 0.f ;
		cmh_rh:valid_max = 105.f ;
		cmh_rh:valid_delta = 30.f ;
		cmh_rh:missing_value = -9999.f ;
	int qc_cmh_rh(time) ;
		qc_cmh_rh:long_name = "Quality check results on field: CMH relative humidity, calculated" ;
		qc_cmh_rh:units = "unitless" ;
		qc_cmh_rh:description = "See global attributes for individual bit descriptions." ;
	float dew_point_mean(time) ;
		dew_point_mean:long_name = "Dew point mean, calculated" ;
		dew_point_mean:units = "C" ;
		dew_point_mean:valid_min = -60.f ;
		dew_point_mean:valid_max = 30.f ;
```

```
        dew_point_mean:valid_delta = 10.f ;
        dew_point_mean:missing_value = -9999.f ;
    int qc_dew_point_mean(time) ;
        qc_dew_point_mean:long_name = "Quality check results on field: Dew point mean, calculated" ;
        qc_dew_point_mean:units = "unitless" ;
        qc_dew_point_mean:description = "See global attributes for individual bit descriptions." ;
    float dew_point_std(time) ;
        dew_point_std:long_name = "Dew point standard deviation" ;
        dew_point_std:units = "C" ;
    float trh_err_code(time) ;
        trh_err_code:long_name = "Temperature and relative humidity sensor error code" ;
        trh_err_code:units = "unitless" ;
    float logger_volt(time) ;
        logger_volt:long_name = "Logger voltage" ;
        logger_volt:units = "V" ;
        logger_volt:missing_value = -9999.f ;
        logger_volt:valid_min = 10.f ;
        logger_volt:valid_max = 15.f ;
        logger_volt:valid_delta = 5.f ;
    int qc_logger_volt(time) ;
        qc_logger_volt:long_name = "Quality check results on field: Logger voltage" ;
        qc_logger_volt:units = "unitless" ;
        qc_logger_volt:description = "See global attributes for individual bit descriptions." ;
    float logger_temp(time) ;
        logger_temp:long_name = "Logger temperature" ;
        logger_temp:units = "C" ;
        logger_temp:missing_value = -9999.f ;
        logger_temp:valid_min = -25.f ;
        logger_temp:valid_max = 50.f ;
        logger_temp:valid_delta = 10.f ;
    int qc_logger_temp(time) ;
        qc_logger_temp:long_name = "Quality check results on field: Logger temperature" ;
        qc_logger_temp:units = "unitless" ;
        qc_logger_temp:description = "See global attributes for individual bit descriptions." ;
    float lat ;
        lat:long_name = "North latitude" ;
        lat:units = "degree_N" ;
        lat:valid_min = -90.f ;
        lat:valid_max = 90.f ;
    float lon ;
        lon:long_name = "East longitude" ;
        lon:units = "degree_E" ;
        lon:valid_min = -180.f ;
        lon:valid_max = 180.f ;
    float alt ;
        alt:long_name = "Altitude above mean sea level" ;
        alt:units = "m" ;

// global attributes:
        :command_line = "met_ingest -s nsa -f C1" ;
        :process_version = "ingest-met-4.10-0.el5" ;
        :dod_version = "met-b1-1.2" ;
        :site_id = "nsa" ;
        :facility_id = "C1: Barrow, Alaska" ;
        :data_level = "b1" ;
        :input_source = "/data/collection/nsa/nsametC1.00/MetData.20130209000000.dat" ;
        :sampling_interval = "variable, see instrument handbook" ;
        :averaging_interval = "60 seconds" ;
        :averaging_interval_comment = "The time assigned to each data point indicates the end of the
averaging interval." ;
        :serial_number = "" ;
        :standard_measurement_height = "2m" ;
        :wind_measurement_height = "10m" ;
        :pwd = "Present Weather Detector" ;
        :cmh = "Chilled Mirror Hygrometer" ;
        :qc_standards_version = "1.0" ;
        :qc_method = "Standard Mentor QC" ;
        :qc_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.\n",
                "\n",
                "The QC field values make use of the internal binary format to store the results of the
individual QC tests. This allows the representation of multiple QC states in a single value. If the test
associated with a particular bit fails the bit is turned on. Turning on the bit equates to adding the integer
value of the failed test to the current value of the field. The QC field\'s value can be interpreted by
applying bit logic using bitwise operators, or by examining the QC value\'s integer representation. A QC
field\'s integer representation is the sum of the individual integer values of the failed tests. The bit and
integer equivalents for the first 5 bits are listed below:\n",
                "\n",
                "bit_1 = 00000001 = 0x01 = 2^0 = 1\n",
                "bit_2 = 00000010 = 0x02 = 2^1 = 2\n",
                "bit_3 = 00000100 = 0x04 = 2^2 = 4\n",
                "bit_4 = 00001000 = 0x08 = 2^3 = 8\n",
                "bit_5 = 00010000 = 0x10 = 2^4 = 16" ;
        :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 values exceeds valid_delta." ;
        :qc_bit_4_assessment = "Indeterminate" ;
        :zeb_platform = "nsametC1.b1" ;
        :history = "created by user dsmgr on machine iron at 9-Feb-2013,1:39:00, using $State: zebra-
zeblib-4.23-0.el5 $" ;
}
```

Appendix B
Time Manipulation Script, “convert_time.py”

```
# convert_time.py
# Example code to convert ARM (Epoch) time to other forms
# First import the NetCDF4 package
from netCDF4 import Dataset

# Load the time package
import time

# Load the NetCDF file into a data object
nc_file = Dataset('nsametC1.b1.20130210.000000.cdf','r',format='NETCDF3_CLASSIC')

# Load base_time (epoch time) and time_offset
bt_obj = nc_file.variables['base_time']
to_obj = nc_file.variables['time_offset']

# Load the time and temperature data into the workspace
base_time = bt_obj[:]
time_offset = to_obj[:]
s = 'base_time = ' + repr(base_time[0]) + ' and time_offset of the first record is ' + repr(time_offset[0])
print s

# The utility gmtime converts epoch time to a structure with the following form
# Index Attribute
# 0  year                        # 6 Week_day
# 1  month                       #7  Year_day
# 2  month_day                   #8 Daylight Savings Time flag
# 3  hour
# 4  minute
# 5  second

# For example, to calculate the date and time for the first data point in this file:
# First add the first element of the time offset to the base_time
t0 = base_time + time_offset[0]

# Next convert this time from epoch to the above structure
ts = time.gmtime(t0)

# Print date and time by converting the appropriate structure elements to strings
str_date = 'The date is: ' + repr(ts[1]) + '/' + repr(ts[2]) + '/' + repr(ts[0])
print str_date
str_time = 'and the time (GMT) is: ' + repr(ts[3]) + ':' + repr(ts[4]) + ':' + repr(ts[5])
print str_time
```

Appendix C
Basic 1-Dimensional Plotting Script, “plot_temperature.py”

```
# plot_temperature.py
# Example code to plot ARM NetCDF data

# First import the NetCDF4 package
from netCDF4 import Dataset

# Next import plotting functions
from pylab import *

# Load the NetCDF file into a data object
nc_file = Dataset('nsametC1.b1.20130210.000000.cdf','r',format='NETCDF3_CLASSIC')

# Load the time and temperature as data objects
temp_obj = nc_file.variables['temp_mean']
time_obj = nc_file.variables['time']

# Load the time and temperature data into the workspace
temperature = temp_obj[:]
time = time_obj[:]

# Convert the time from seconds since midnight GMT to hours since midnight GMT
time_hour = time/3600

# Now plot the data
plot(time_hour, temperature)

# Limit the axes ranges
xlim(0,24)
# You can also force the ticks to be at 6-hour intervals (for example)
xticks([0,6,12,18,24])

# Add axes labels and a title
xlabel("Hours Since Midnight")
ylabel("Temperature (Degrees C)")
title("Temperature at Barrow: February 10, 2013")

# Output as a PNG format file
savefig('testplot.png', format='png')

# When plotting from within a script you have to instruct the script to display the output
show()
```

Appendix D
Useful Python Plotting Commands

The following plotting commands excerpted from matplotlib.pyplot.plotting() list of commands See http://matplotlib.org/api/pyplot_summary.html for complete list, including all arguments.

Bar(xvals, yvals)	create bar graph
Boxplot()	turns the axes box on or off
Clf()	clears figure
Close()	closes figure window
Colorbar()	add color bar
Errorbar()	add error bar
Figlegend()	add a legend
Hist()	plot a histogram
Legend()	add legend to figure
Minorticks_on()/_off()	turn minor axis ticks on or off
Plot(xvals, yvals, ‘g—‘)	mark points with green dashed line
Savefig(fname, format=’png’)	save figure to a file
Scatter(xvals,yvals, marker=’+’)	scatter plot
Subtitle(‘This is the Title’)	add a title to the plot
Xlabel(‘Time’)	add a x label
Xscale()	set scaling for x-axis
Xlim(), ylim()	x and y limits of the axes
Ylabel(‘Temperature (K)’)	add a y label
Yscale()	set scaling for y-axis

String characters to control the line style or marker:

character	description
'-'	solid line style
'--'	dashed line style
'-.'	dash-dot line style
':'	dotted line style
'.'	point marker
','	pixel marker
'o'	circle marker
'v'	triangle_down marker
'^'	triangle_up marker
'<'	triangle_left marker
'>'	triangle_right marker
'1'	tri_down marker
'2'	tri_up marker
'3'	tri_left marker
'4'	tri_right marker
's'	square marker
'p'	pentagon marker
'*'	star marker
'h'	hexagon1 marker
'H'	hexagon2 marker
'+'	plus marker
'x'	x marker
'D'	diamond marker
'd'	thin_diamond marker
'|'	vline marker
'_'	hline marker

Color abbreviations:

character	color
‘b’	blue
‘g’	green
‘r’	red
‘c’	cyan
‘m’	magenta
‘y’	yellow
‘k’	black
‘w’	white

More integer and string characters to control markers:

marker	description
7	caretdown
4	caretleft
5	caretright
6	caretup
''	nothing
'None'	nothing
' '	nothing
None	nothing
'8'	octagon
3	tickdown
0	tickleft
1	tickright
2	tickup