Usage of NCL, GrADS, PyHDF, GDL and GDAL to Access HDF Files

Choonghwan Lee (clee83@hdfgroup.org)
MuQun Yang (myang6@hdfgroup.org)
The HDF Group

This document explains how to access and visualize HDF4 and/or HDF5 files using five freely available software packages.

1 Prerequisite

This document assumes that users have a basic knowledge of the following data formats and the corresponding software packages:

- HDF4 (1)
- HDF5 (2)
- HDF-EOS2 (3)
- netCDF-4 (4)

2 Introduction

In this document we describe how to use NCL (5), GrADS (6), PyHDF (7), GDL (8), and GDAL (9) to access HDF files. We use NASA HDF files to show the procedures for accessing or visualizing the data with these open source packages. For the general usages of these packages, please refer to their user's guides, (10), (11) and (12), or websites. An evaluation of these packages can be found in the document *An Evaluation of HDF Support for NCL, GrADS, PyHDF, GDL, and GDAL* (13).

3 Environment

An Intel x86-based system running GNU/Linux was used to run all five packages. The packages were linked with HDF4.2r3, HDF5 1.8, and netCDF-4. We used GCC 3.4.6 to build the libraries and packages. For PyHDF, Python 2.5.2 was used.

4 Sample Files

One HDF4 file and two HDF-EOS2 files were used to demonstrate how to access or visualize HDF data with these packages. These files are used to store data from the NASA Special Sensor Microwave/Imager (SSMI) Sensor (14) and the Advanced Microwave Scanning Radiometer for the EOS (AMSR-E) satellite system (15). Table 1 shows the instrument names, formats and other information about these HDF4/HDF-EOS2 files. We also converted HDF-EOS2 files to netCDF-4 classic

model–compliant and netCDF-4-compliant HDF5 files. In this document, we mainly refer to the instrument and the description of physical variables when specifying a file or a field in a file. However, the file names may be referred to in the code examples that appear throughout this document.

Instrument	SSMI	AMSR-E	AMSR-E
Description of Physical Variables	Ocean Wind Fields	Rainfall Accumulations	Brightness Temperature, Sea Ice Concentration, Snow Depth over Sea Ice
Data Format	HDF4	HDF-EOS2 Grid	HDF-EOS2 Grid
Variations		1. netCDF-4-compliant HDF5 2. netCDF-4 classic model–compliant HDF5	1. netCDF-4 classic model–compliant HDF5
Projection	Geographic	Geographic	Polar Stereographic

Table 1. Data files

4.1 SSMI – Ocean Wind Fields

The first file stores the NASA SSMI ocean wind fields data. In this document, we will use the two variables `u10m` and `v10m` that represent the U component and the V component of the wind field. Both variables have three dimensions: time (from January 2005 to December 2005), longitude, and latitude.

The file name is `atlas.ssmi.ver02.level3.5_5day.s950103.hdf`. The original file cannot be obtained. Users can download a similar HDF4 file from ftp://podaac.jpl.nasa.gov/pub/ocean_wind/ssmi/atlas_ssmi_ver10/data/level3.5_5day/1995/atlas.ssmi.ver10.level3.5_5day.s19950101.hdf.

4.2 AMSR-E – Rainfall Accumulations

The second file (AE_RnGd) stores the NASA AMSR-E data that describes the monthly average rainfall accumulation over ocean and land in July 2007.

4.2.1 HDF-EOS2

The format of the original file is HDF-EOS2. The original file name is `AMSR_E_L3_RainGrid_B05_200707.hdf`. Users can download this file from ftp://n4ftl01u.ecs.nasa.gov/SAN/AMSA/AE_RnGd.001/2007.07.01/AMSR_E_L3_RainGrid_B05_200707.hdf.

We changed the extension from `hdf` to `he2` when we accessed this file via NCL.

4.2.2 NetCDF-4-compliant HDF5

We used the HDF4-to-HDF5 Conversion tool (16) to convert the AMSR-E HDF-EOS2 file to a netCDF-4-compliant HDF5 file. The converted file was renamed to either `AMSR_E_L3_RainGrid_B05_200707.h5` or `AMSR_E_L3_RainGrid_B05_200707.nc`, as required by the software package processing it.

4.2.3 NetCDF-4 Classic Model–compliant HDF5

Since the netCDF-4 classic model (17) has more restrictions than the general netCDF-4 model, two steps are needed to create a netCDF-4 classic model–compliant HDF5 file. The first step is to create a

netCDF-4-compliant HDF5 file as addressed in the previous section. The second step is to follow the procedure explained in Appendix 11.1.1. The file is called `AMSR_E_L3_RainGrid_B05_200707_flatten.nc`.

4.3 AMSR-E – Brightness Temperature, Sea Ice Concentration, Snow Depth over Sea Ice

The third file (AE_SI12) contains several fields that represent brightness temperatures, sea ice concentration, and snow depth over sea ice. This file is also from AMSR-E (18). Unlike the other two files, the polar stereographic projection is used.

4.3.1 HDF-EOS2

The original file is an HDF-EOS2 file and the filename is `AMSR_E_L3_SeaIce12km_B02_20020619.hdf`. Users can download this file from ftp://n4ftl01u.ecs.nasa.gov/SAN/AMSA/AE_SI12.001/2002.06.19/AMSR_E_L3_SeaIce12km_B02_20020619.hdf.

4.3.2 NetCDF-4 Classic Model–compliant HDF5

To create the netCDF-4 classic model–compliant HDF5 file, one needs to follow the same procedure described in Section 4.2.3. The converted file name is `AMSR_E_L3_SeaIce12km_B02_20020619_flatten.nc`.

5 NCL

5.1 Overview

The NCAR Command Language (NCL) is an interpreted language designed for scientific data analysis and visualization. In this section, we explain how to install NCL, read HDF4 and HDF5 files, and visualize them.

5.2 Installation

Although NCL is free, registration is required to download it. One can find information regarding registration, downloading, and installation from http://www.ncl.ucar.edu/Download/. We used NCL 5.0.0.

NCAR distributes precompiled NCL binaries for several widely used platforms including AIX, IRIX, Linux, Mac OSX, Solaris, and cygwin. We used their Linux distribution, and it worked without any problems. Source code is also available, and building NCL from source code is documented at http://www.ncl.ucar.edu/Download/build_from_src.shtml.

5.3 How to Use

5.3.1 Introduction

The following code shows a typical way to use NCL to visualize a variable in a file. This code needs to be typed at the prompt that NCL shows.

```
load "$NCARG_ROOT/lib/ncarg/nclex/gsun/gsn_code.ncl"
```

```
begin
  file1 = addfile("filename", "r")
  variable1 = file1->var1(:,:)
  variable2 = file1->var2(0,:,:)

  variable1@_FillValue = -1

  xwks1 = gsn_open_wks("pdf","outputfile")

  resources1 = True
  resources1@tiMainFont = 21

  plot1 = gsn_csm_vector_map_ce(xwks1,variable1,variable2,resources1)
end
```

Figure 1. A typical NCL code to read and plot data

The first line starting with `load` loads an NCL module that defines functions used in this NCL code.

An NCL code starts with the keyword `begin` and ends with the keyword `end`. The `addfile()` function is used to open a file. Two arguments need to be provided for this function. The first argument provides the file name and the second argument provides the file access mode, in this case, `r`, which represents read-only mode. This function returns the file descriptor, in this case, `file1`.

The file descriptor can be used to refer to a variable in the file. When referring to a variable, one can use NCL's subscript feature to select a specific portion of a variable. The explanation of subscripts used in NCL can be found in the "Subscript" section of *NCL Language Reference Guide: Variables* (19).

As shown in Figure 1, the entire data of data field `var1`, denoted as `var1(:,:)`, are represented as an NCL variable, `variable1`. A subset of a data field `var2`, denoted as `var2(0,:,:)`, is represented as NCL variable, `variable2`. The subsection of the first dimension of `var2` is one element, the first element of this dimension. The subsections of the second and third dimensions include the whole sections of these dimensions.

NCL tries to read dimensions, units, and fill values from variable attributes. When a variable does not have attributes, users need to manually provide the fill value for better visualization. To set the fill value -1 for `variable1`, for example, one can write a statement as `variable1@_FillValue = -1`. When two-dimensional dimension scales are associated with a variable, users need to write additional statements such as `variable1@lon2d = lon`, assuming that a variable, `lon`, contains longitude values.

To draw a plot, one needs to call `gsn_open_wks()` first to get a workstation descriptor. A workstation is an instance of an output device such as a screen or a file. The name of the descriptor in this example is `xwks1` as shown in Figure 1. One of its arguments determines whether NCL draws a plot on the screen, or creates a file such as PDF or PostScript.

Users can customize a plot by setting attributes of a resource object created by assigning `True`. A resource object in NCL means configuration settings of a plot such as vector shapes, font sizes and colors of a plot. For example, the statement `resources1@tiMainFont = 21` specifies that the font size of the main title is 21. For more information, refer to http://www.ncl.ucar.edu/Document/Graphics/Resources/index.shtml.

With a workstation and a resource object, one can draw a plot of variables by calling NCL APIs such as `gsn_csm_vector_map_ce()` or `gsn_csm_contour_map_ce()`. In this example, `gsn_csm_vector_map_ce()` is called. Since this example generates a vector plot with `varible1` as one component of the vector and `variable2` as another component of the vector, both `variable1` and `variable2` should be passed as parameters to `gsn_csm_vector_map_ce()`. For a contour plot, only one variable should be passed to the function `gsn_csm_contour_map_ce()`.

5.3.2 Handle an HDF4 or HDF5 file

The extension of the file name passed to `addfile()` function is important because NCL detects the file format based on the extension. To open an HDF-EOS2 file whose extension is `.hdf`, for example, one needs to rename the actual file name or append `.he2` to the argument.

When reading HDF-EOS2 fields, users need to be aware of the name mangling that occurs in NCL. Suppose that `RrLandRain` is a data field defined under a grid `MonthlyRainTotal_GeoGrid`; NCL flattens this structure and appends the grid name to the field name. As a result, the variable name becomes `RrLandRain_MonthlyRainTotal_GeoGrid`. To access this data field, users need to refer to this mangled name, not the pure data field name, `RrLandRain`.

5.3.3 Examples

In this section, we will explain how to visualize an HDF4 SDS (5.3.3.1), HDF-EOS2 (5.3.3.2, 5.3.3.4) and netCDF-4 classic model–compliant HDF5 (5.3.3.3, 5.3.3.5).

5.3.3.1 Visualize an HDF4 SDS

Figure 2 shows how to use NCL to read vectors from an HDF4 file and to draw a vector plot. We used the ocean wind fields data from the SSMI instrument.

```
load "$NCARG_ROOT/lib/ncarg/nclex/gsun/gsn_code.ncl"
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_csm.ncl"

begin
  cdf_file = addfile("atlas.ssmi.ver02.level3.5_5day.s950103.hdf", "r")
  u = cdf_file->u10m(0,:,:)
  v = cdf_file->v10m(0,:,:)

  xwks = gsn_open_wks("pdf","ssmi")
  resources = True
  plot = gsn_csm_vector_map_ce(xwks,u,v,resources)
end
```

Figure 2. NCL code to read and plot data from an HDF4 SDS

The above code reads part of two HDF4 SDSs (`u10m` and `v10m`) and draws a vector plot over a cylindrical equidistant map.

With some additional settings, the plot shown in Figure 3 can be generated. The unabridged code with the full `resources` variable setting is provided in Appendix 11.2.1.

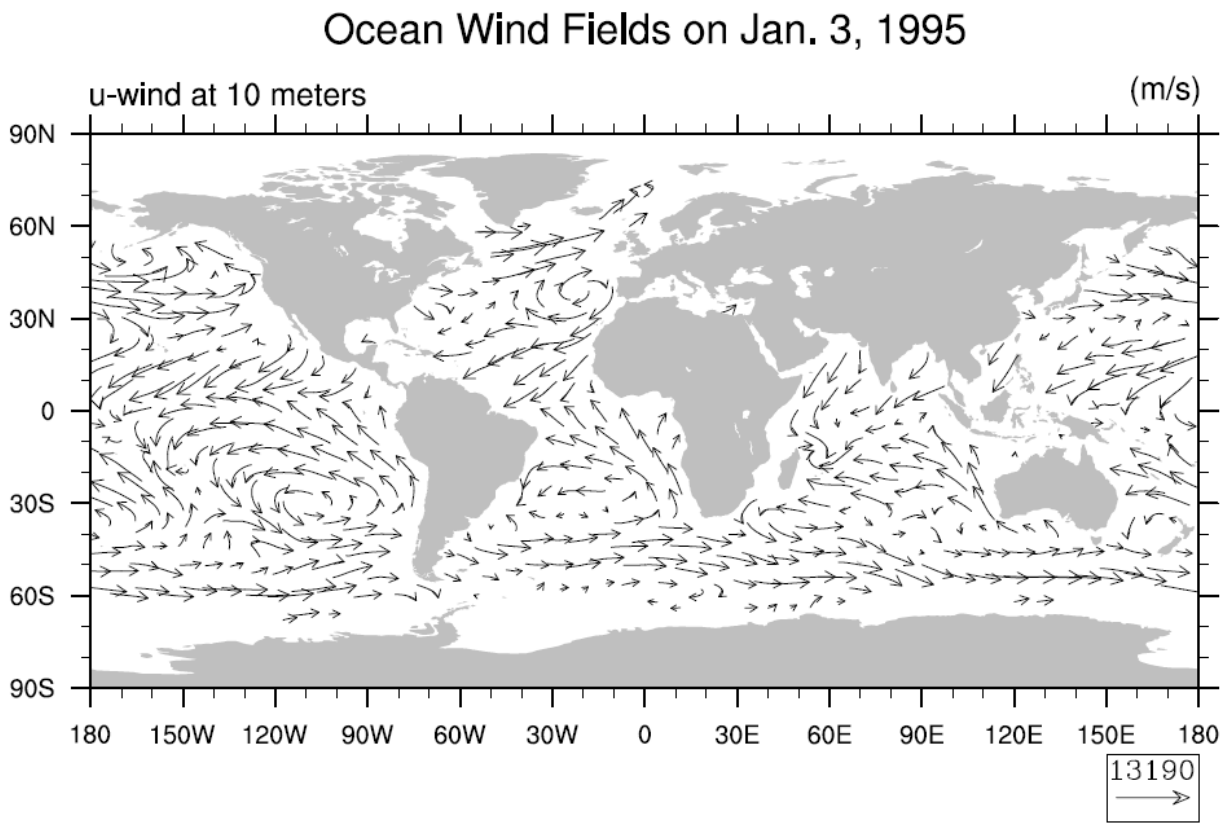


Figure 3. NCL vector plot from an HDF4 SDS

5.3.3.2 Visualize an HDF-EOS2 File that has 1-D Coordinate Variables

Figure 4 shows code to read an HDF-EOS2 field and draw a contour plot.

```
load "$NCARG_ROOT/lib/ncarg/nclex/gsun/gsn_code.ncl"
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_csm.ncl"

begin
  cdf_file = addfile("AMSR_E_L3_RainGrid_B05_200707.he2","r")
  rrland = cdf_file->RrLandRain_MonthlyRainTotal_GeoGrid(:,:)
  rrland@_FillValue = -1

  resources = True
  xwks = gsn_open_wks("pdf","AE_RnGd.hdfeos2")
  plot = gsn_csm_contour_map_ce(xwks,rrland,resources)
end
```

Figure 4. NCL code to read and plot grid data from an HDF-EOS2 file

Due to name mangling, this code uses the mangled name, `RrLandRain_MonthlyRainTotal_GeoGrid` to access the data field `RrLandRain` in the grid `MonthlyRainTotal_GeoGrid`. Since this field does not have an attribute that specifies the fill value, this code informs the fill value to draw a more meaningful plot.

`gsn_csm_contour_map_ce()` draws a contour plot over a cylindrical equidistant map. With several additional resource settings to specify NCL plot options, we could get the result shown in Figure 5. The full code is provided in Appendix 11.2.2.

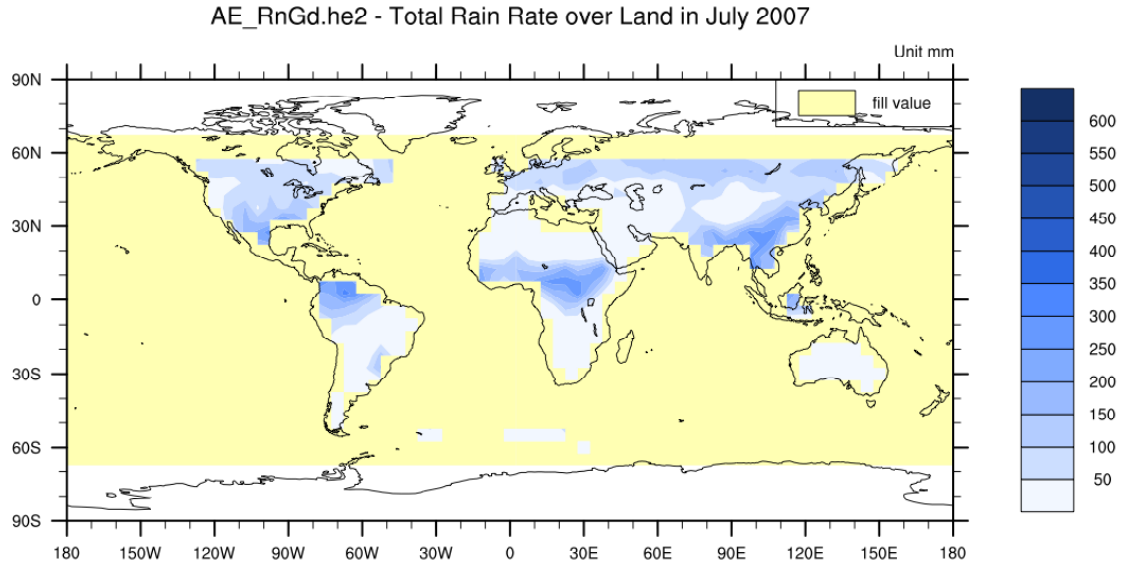


Figure 5. NCL contour plot from an HDF-EOS2 Grid field

5.3.3.3 Visualize a NetCDF-4 Classic Model–compliant HDF5 File that has 1-D Coordinate Variables

If an HDF5 file is netCDF-4 classic model–compliant, one can use NCL to visualize data.

```
load "$NCARG_ROOT/lib/ncarg/nclex/gsun/gsn_code.ncl"
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_csm.ncl"

begin
  cdf_file = addfile("AMSR_E_L3_RainGrid_B05_200707_flatten.nc","r")
  rrland = cdf_file->RrLandRain(:,:)
  rrland@_FillValue = -1

  resources = True
  xwks = gsn_open_wks("pdf","AE_RnGd.netcdf4")
  plot = gsn_csm_contour_map_ce(xwks,rrland,resources)
end
```

Figure 6. NCL code to read and plot an HDF5 dataset from a netCDF-4 classic model–compliant HDF5 file

Figure 6 is almost the same as Figure 4 regardless of the file formats. The result is the same as Figure 5. For more information about the netCDF-4 classic model–compliant HDF5 file, refer to Appendix 11.1.1.

5.3.3.4 Visualize an HDF-EOS2 File that has 2-D Coordinate Variables

As we mentioned in Section 4.3, this file uses the north polar stereographic projection that requires two-dimensional longitude and latitude. This projection requires additional effort.

```
load “$NCARG_ROOT/lib/ncarg/nclex/gsun/gsn_code.ncl"
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_csm.ncl"

begin
  cdf_file = addfile("AMSR_E_L3_SeaIce12km_B02_20020619.he2", "r")
  nh18vday = cdf_file->SI_12km_NH_18V_DAY_NpPolarGrid12km(:,:)
  nh18vday@lon2d = cdf_file->GridLon_NpPolarGrid12km
  nh18vday@lat2d = cdf_file->GridLat_NpPolarGrid12km

  xwks = gsn_open_wks("pdf","AE_SI12.north.dailyavgt")
  plot = gsn_csm_contour_map_polar(xwks,nh18vday,resources)
end
```

Figure 7. NCL code using two-dimensional longitude and latitude in an HDF-EOS2 file

`SI_12km_NH_18V_DAY` is an HDF-EOS2 data field defined in this file. `GridLon_NpPolarGrid12km` and `GridLat_NpPolarGrid12km` don’t exist in the file, but NCL generates both of these two-dimensional geolocation fields, which represent longitude and latitude. Since NCL does not automatically associate `SI_12km_NH_18V_DAY` with these two geolocation fields, users should specify the associations with statements, starting with `nh18vday@lat2d` and `nh18vday@lon2d`. NCL will recognize the `lat2d` and `lon2d` attributes and associate the data variable with geolocation variables.

`gsn_csm_contour_map_polar()` draws a plot over a polar stereographic map, as shown in Figure 8. The unabridged code is explained in Appendix 11.2.3.

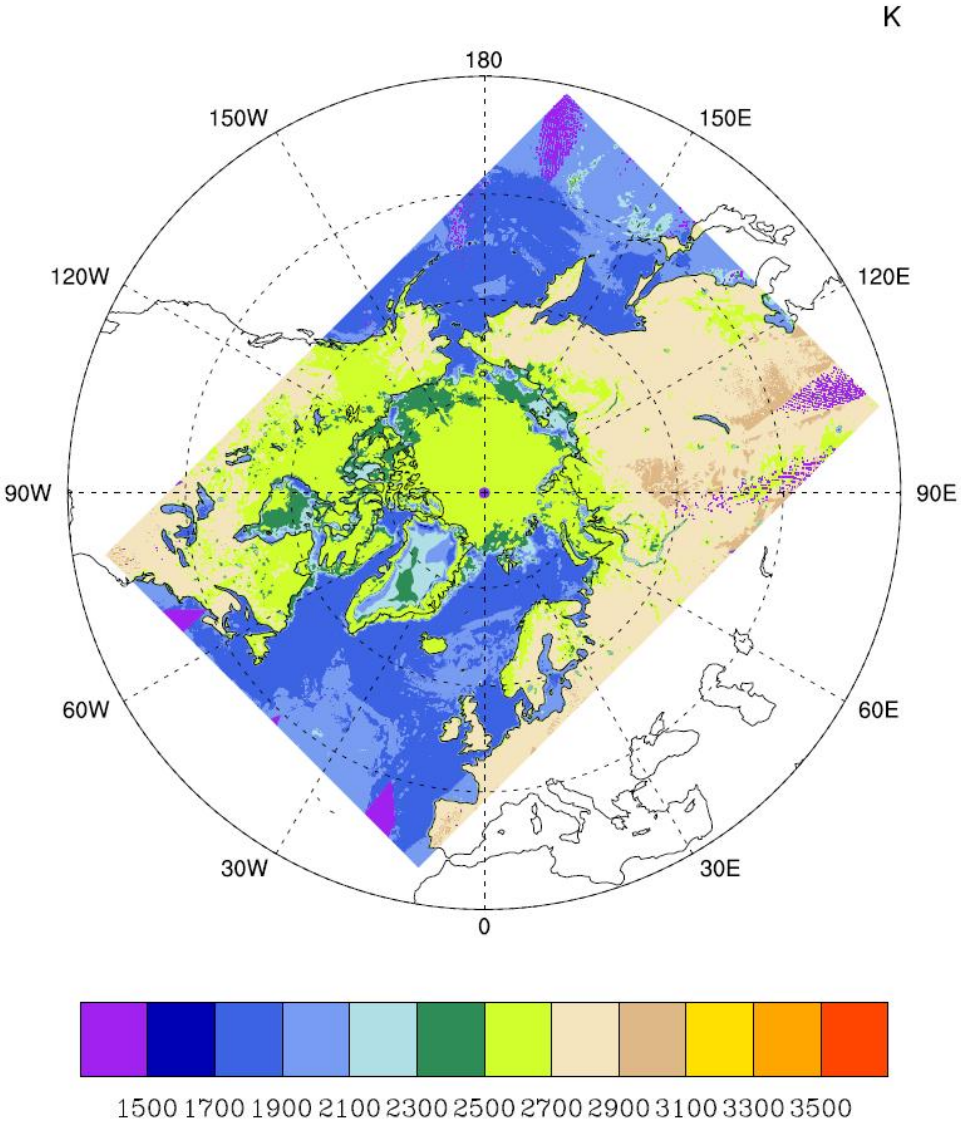


Figure 8. NCL contour plot over a polar stereographic map

5.3.3.5 Visualize a NetCDF-4 Classic Model–compliant HDF5 File that has 2-D Coordinate Variables

This section explains how to draw a plot when an HDF5 file has two-dimensional longitude and latitude. To accomplish this, the file must have two additional datasets representing longitude and latitude. In Figure 9, `lon` and `lat` are variables containing real coordinate values.

```
load “$NCARG_ROOT/lib/ncarg/nclex/gsun/gsn_code.ncl"
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_csm.ncl"

begin
  cdf_file = addfile("AMSR_E_L3_SeaIce12km_B02_20020619_flatten.nc", "r")
  nh18vday = cdf_file->SI_12km_NH_18V_DAY(:,:)
```

```
  nh18vday@lat2d = cdf_file->lat
  nh18vday@lon2d = cdf_file->lon

  xwks = gsn_open_wks("pdf","AE_SI12.north.dailyavgt")
  plot = gsn_csm_contour_map_polar(xwks,nh18vday,resources)
end
```

Figure 9. NCL code using two-dimensional longitude and latitude in a netCDF-4 classic model–compliant HDF5 file

Basically, Figure 9 is the same as Figure 7 except that the code in Figure 9 uses both `lon` and `lat`, which exist in the HDF5 file. The data variable and the two coordinate variables should have the same number of elements because coordinate variables provide longitude and latitude for each element in the data variable. Since we can provide arbitrary longitude and latitude values with this method, we believe that NCL can draw a plot from all kinds of projections.

6 GrADS

6.1 Overview

The Grid Analysis and Display System (GrADS) is an interactive desktop tool that is used for easy access, manipulation, and visualization of earth science data. In this section we explain how to read and visualize HDF4 and HDF5 files.

6.2 Installation

Since GrADS does not support netCDF-4, GrADS cannot read netCDF-4-compliant HDF5 files. With some undocumented hacks, users can make GrADS link with netCDF-4 rather than netCDF-3. For this netCDF-4 hack, users cannot use the precompiled binary. This hack is explained in Appendix 11.1.2.

6.3 How to Use

6.3.1 Introduction

Three methods are provided to read netCDF or HDF4 SDS files: `sdfopen`, `xdfopen`, and `open`.

After reading a file, data can be manipulated and visualized in many ways using the `display` command. One can change visualization settings using `set lon`, `set lat`, `set gxout`, and so on. Like NCL, GrADS can both draw plots on the screen and export them to a file.

6.3.1.1 sdfopen

The `sdfopen` command accepts the actual file name. Then, GrADS recognizes longitude and latitude from attributes and reads all values from the data fields. GrADS can recognize missing values from attributes and can detect date and time from human-readable strings such as "`hours since 1995-01-01 00:00:00`".

This method is the most convenient, but the target file should conform to COARDS conventions (20). For example, HDF-EOS2 grid files cannot be opened using `sdfopen` because the values of longitude and latitude are not explicitly defined.

6.3.1.2 xdfopen

When a file does not conform to COARDS conventions, one can use the `xdfopen` command. This command requires a user-defined description file. The `xdfopen` command receives the description file name as an argument. Note that `xdfopen` does not receive the actual data file name because that is specified in the description file.

Figure 10 shows part of a typical description file used by the `xdfopen` command to open an HDF-EOS2 grid file.

```
DSET filename
XDEF XDim:grid1 72 LINEAR 2.5 5
YDEF YDim:grid1 28 LINEAR -67.5 5
VARS 1
var1=>variable1
ENDVARS
```

Figure 10. A typical description file to read an HDF-EOS2 grid file

The first line of the description file specifies the actual HDF-EOS2 file name. Then, `XDEF` and `YDEF` define longitude and latitude, respectively. Both `XDEF` and `YDEF` receive five arguments. The first argument is the actual dimension name in the file. The HDF-EOS2 grid file does not store latitude and longitude as variables. It only defines dimension names, which are normally `XDim` and `YDim` followed by a colon and the enclosing grid name. These names can be easily fetched with *hdp*, a command-line dumper utility of HDF4, or *HDFView* (21). In this example, `XDim:grid1` is the actual dimension name defined in the file. The second argument specifies the size of the dimension, and users can easily get this information from hdp or HDFView. This is explained in Appendix 11.1.2.2.

The remaining three arguments, `LINEAR 2.5 5`, mean that the longitude starts from 2.5 degrees and the values increase by a step size of 5. Since these arguments are not explicitly defined in the file, users need to get these values from the projection code and related attributes inside the HDF-EOS2 file.

If longitude and latitude values are not linear, one needs to use `LEVELS` followed by a list of actual values instead of `LINEAR`. In addition, using `XDEF` and `YDEF` is not proper if longitude and latitude are two-dimensional. This is explained in Section 6.3.2.4.

The remaining part defines variables. The line starting with `var1` defines one variable, `variable1`, from a field, `var1`, in the file. One should be aware that defining `XDEF` and `YDEF` correctly is crucial to drawing correct plots because GrADS retrieves each element of `variable1` from *user-defined* `XDEF` and `YDEF` options.

6.3.1.3 open

If a file does not conform to COARDS conventions and `xdfopen` fails, one can use the `open` command. Like `xdfopen`, `open` also requires a description file. The description file is very similar although the description file for `open` requires more settings. Since the `open` command supports the `PDEF` option, two-dimensional longitude and latitude can be handled. The detailed explanation and an example are provided in Section 6.3.2.4.1.

6.3.2 Examples

In this section, we explain how to visualize an HDF4 SDS (6.3.2.1), HDF-EOS2 (6.3.2.2, 6.3.2.4) and netCDF-4 classic model–compliant HDF5 (6.3.2.3, 6.3.2.5).

6.3.2.1 Visualize an HDF4 SDS

The variable for ocean wind fields from the SSMI instrument has enough metadata for GrADS to recognize it; it has longitude and latitude dimension scales, unit, and fill value. To read this file, `sdfopen` is sufficient.

Since this file has time dimension as well as longitude and latitude, several plots, based on the time dimension, can be animated. The following is a full list of commands that read the file and draw animated plots. When GrADS shows the prompt, type these statements:

```
$ grads
ga> sdfopen atlas.ssmi.ver02.level3.5_5day.s950103.hdf
ga> set lon -180 180
ga> set t 1 73
ga> set looping on
ga> display u10m ; v10m ; sqrt(u10m * u10m + v10m * v10m)
```

Figure 11. GrADS code to plot an HDF4 SDS

The statements `set t 1 73` and `set looping on` are used for animation. The `t` implies the time dimension, and the output will display an animation containing 73 plots. The `display` command actually draws an animation. Both `u10m` and `v10m` are variable names defined in the file. The last expression, `sqrt(…)`, is optional; GrADS colorizes the vector according to the value of this field. Figure 12 shows the first frame of the animation.

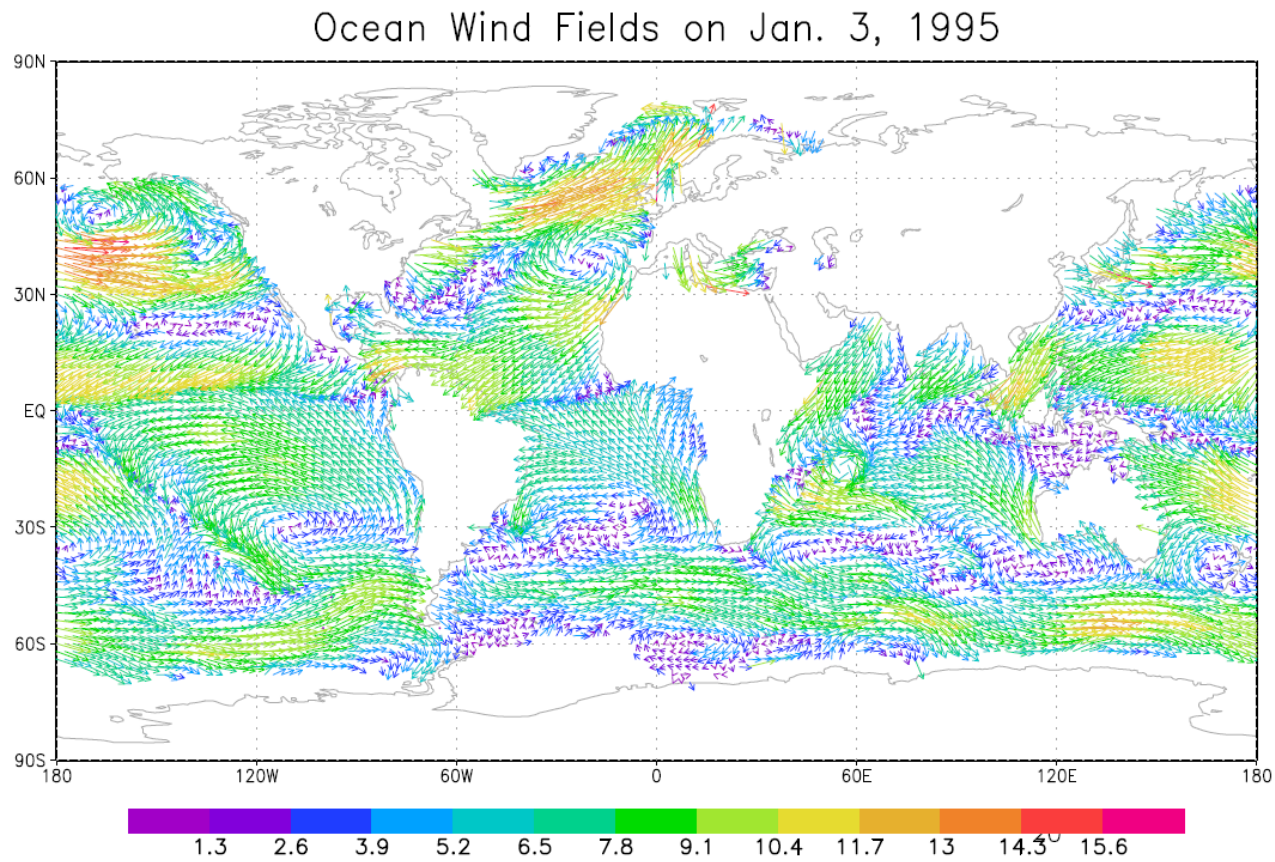


Figure 12. GrADS plot of ocean wind fields on January 3, 1995

6.3.2.2 Visualize an HDF-EOS2 File that has 1-D Coordinate Variables

As explained in Section 6.3.1.1, `sdfopen` cannot be used to visualize HDF-EOS2 grid data. Users can use the `xdfopen` command instead. Figure 13 shows an example of the description file.

```
DSET AMSR_E_L3_RainGrid_B05_200707.hdf
TITLE AE_RnGd
OPTIONS YREV
XDEF XDim:MonthlyRainTotal_GeoGrid 72 LINEAR 2.5 5
YDEF YDim:MonthlyRainTotal_GeoGrid 28 LINEAR -67.5 5
VARS 1
RrLandRain=>RrLandRain Rain rate derived monthly rain total over land.
ENDVARS
```

Figure 13. Description file for rainfall accumulation over land in July 2007

As explained in Section 6.3.1.2, HDF-EOS2 files define dimension names. In this example, `XDim:MonthlyRainTotal_GeoGrid` is one actual dimension name defined in the file.

`OPTIONS YREV` indicates that the latitude has the reverse order. `LINEAR -67.5 5` in the `YDEF` statement means the first data was measured at -67.5 and the next data was measured at -62.5, which means the location goes from south to north. However, the actual data in the file were measured from north to south, and this option provides this information.

We named this description file `AE_RnGd.xdf`. Under the GrADS environment, that file name is passed as the argument to `xdfopen`, as in the following example.

```
$ grads
ga> xdfopen AE_RnGd.xdf
ga> set lon -180 180
ga> set gxout shaded
ga> display RrLandRain
ga> draw title Total Rain Rate over Land in July 2007
```

Figure 14. GrADS code to plot grid data from an HDF-EOS2 file

`set gxout shaded` makes GrADS draw a shaded plot. The `display` command shows a visualization, which is represented in Figure 15.

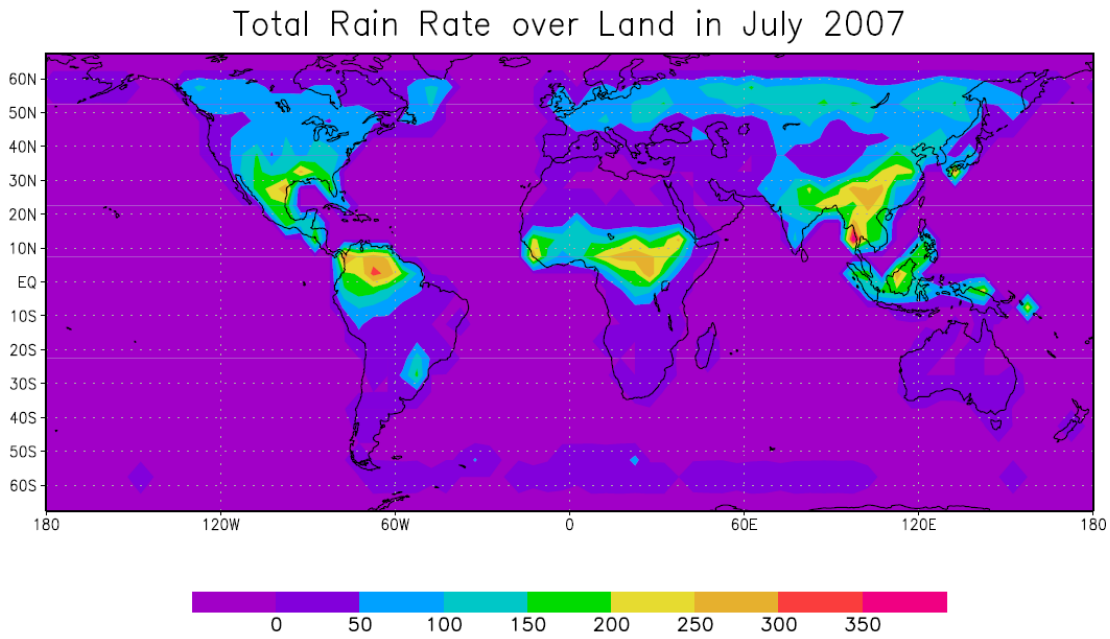


Figure 15. GrADS plot of rain rate over land in July 2007

6.3.2.3 Visualize a NetCDF-4 Classic Mode–compliant HDF5 File that has 1-D Coordinate Variables

Since the HDF4-to-HDF5 converter calculates and writes longitude and latitude from HDF-EOS2 projection code and related attributes, `sdfopen` can be used, and tedious tasks specifying `XDEF` and `YDEF` can be avoided.

However, the generated file does not conform to the netCDF-4 classic model due to groups. To conform to the classic model, the generated file should be manually flattened as we show in Appendix 11.1.1. After the modification, `sdfopen` can directly open the netCDF-4-compliant HDF5 file. Be aware that the current GrADS release cannot handle a netCDF-4 classic model–compliant HDF5 file. Check Appendix 11.1.2 for more details.

```
$ grads
ga> sdfopen AMSR_E_L3_RainGrid_B05_200707_flatten.nc
ga> set lon -180 180
```

```
ga> set gxout shaded
ga> display RrLandRain
ga> draw title Total Rain Rate over Land in July 2007
```

Figure 16. GrADS code to plot an HDF5 dataset from a netCDF-4 classic model–compliant HDF5 file

The visualized plot is the same as in Figure 15.

6.3.2.4 Visualize an HDF-EOS2 File that has 2-D Coordinate Variables

If longitude and latitude values cannot be represented as one-dimensional arrays, XDEF and YDEF are not enough. Two-dimensional longitude and latitude are supported through the PDEF command.

6.3.2.4.1 Using PDEF BILIN

For the PDEF option, open should be used instead of sdfopen or xdfopen as Section 6.3.1.3 explained. Figure 17 shows an example of a description file.

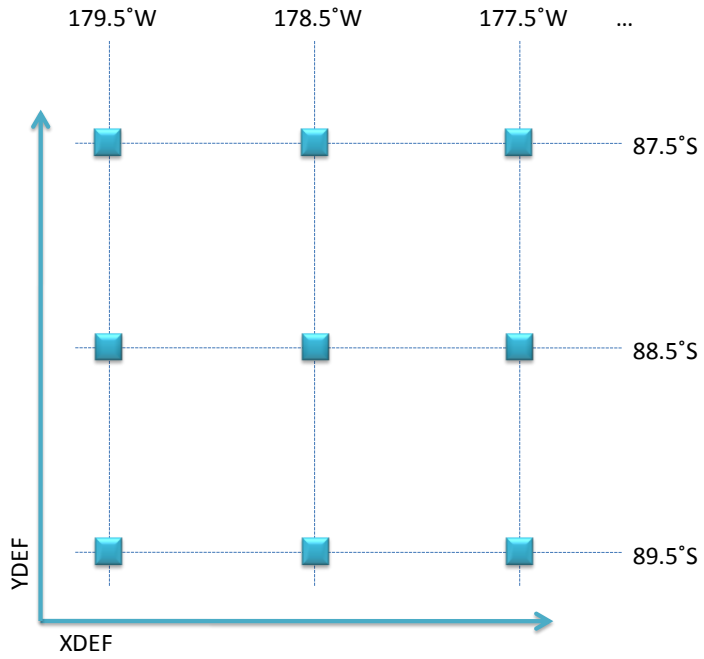
```
DSET AMSR_E_L3_SeaIce12km_B02_20020619.hdf
DTYPE hdfsds
UNDEF 0 _FillValue
PDEF 608 896 BILIN STREAM BINARY bilin_file
XDEF 360 linear -179.5 1
YDEF 180 linear -89.5 1
ZDEF 1 levels 0
TDEF 1 linear 00Z19jun2002 1mo
VARS 1
SI_12km_NH_SNOWDEPTH_5DAY=>snow 0 y,x SI_12 Snow Depth
ENDVARS
```

Figure 17. Description file for the polar stereographic projection

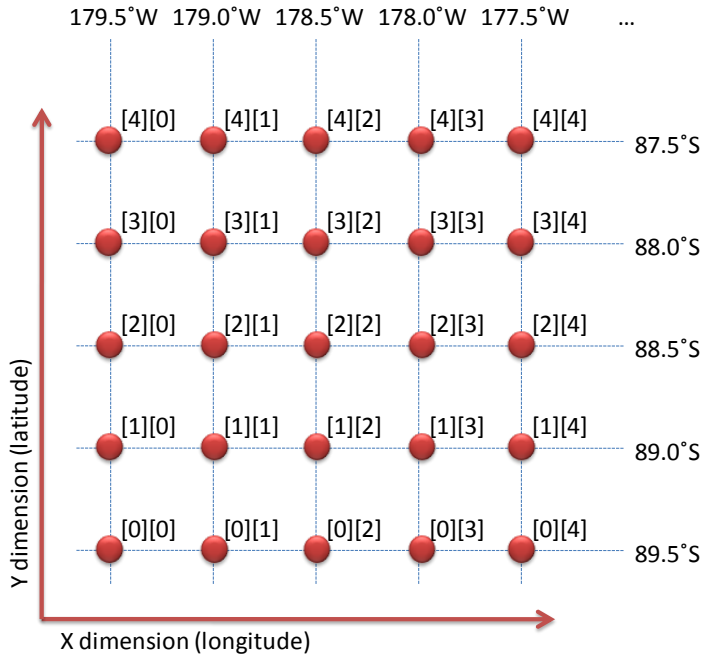
The description file has UNDEF, ZDEF, TDEF that represent undefined values, Z-axis, and time axis, respectively, because the open command requires these definitions. In this example, ZDEF and TDEF are meaningless because the actual data do not have Z-axis and time axis. The fill value is undefined in the original file. To generate a plot that can correctly describe the distribution the physical field, fill value is set to 0.

PDEF BILIN provides a way for GrADS to obtain geolocation information as lon2d and lat2d do in NCL (Section 5.3.3.4). However, the geolocation information should be stored in an additional file, and the file name is passed as an argument to PDEF BILIN. In this example, bilin_file is the name of the additional file. This additional file specifies the horizontal array indices of the data field measured at each grid point defined by XDEF and YDEF.

In Figure 17, XDEF and YDEF define a 360 × 180 horizontal grid. Shown in Figure 18(a), each grid point is represented as a rectangle; the latitude and longitude at each grid point are also specified. Figure 18(b) shows the data array index with the corresponding latitude and longitude, which can be obtained from the EOS2 file. The data array index for each grid point shown in Figure 18(a) needs to be mapped from the information shown in Figure 18(b). For example, the rectangle at the third column and the second row (88.5˚S, 177.5˚W) represents the location of the data element of which the index is [2][4](Figure 18(b)). The data array index for each grid point defined by XDEF and YDEF is then stored in the bilin_file. GrADS draws the plot based on the array index.



(a) Part of geolocations defined by XDEF and YDEF



(b) Part of geolocations where elements of the data field were measured

Figure 18. Geolocations defined by XDEF and YDEF, and geolocations where data were measured

Calculating array indices may not be straightforward for some projections. If the file is an HDF-EOS2 grid file, however, users can exploit the HDF-EOS2 API function `GDll2ij()`. This function accepts the projection code, related parameters, and longitude and latitude, and it returns indices for the X dimension and Y dimension corresponding to the given longitude and latitude. Among the API inputs, the projection code and related parameters are defined in the HDF-EOS2 file. Longitude and latitude

are defined by XDEF and YDEF in the description file. Given the XDEF and YDEF defined in Figure 17, Figure 19 shows the pseudo-code used to generate the file that BILIN accepts.

```
i = array[180 * 360]
j = array[180 * 360]
for lon = -89.5 (step 1, iterates 180 times)
  for lat = -179.5 (step 1, iterates 360 times)
    (ival, jval) = GDll2ij(projection, related params, lon, lat)
    i.add(ival)
    j.add(jval)
write_to_file i
write_to_file j
write_to_file 0 /* wind rotation */
```

Figure 19. Pseudo-code generating the file that BILIN requires

This generated file is the bilin file described previously. With the generated file, the description file can correctly associate scientific data with coordinate variables. Type the following commands at the GrADS prompt in order to draw a shaded plot with the description file for a polar stereographic projection.

```
$ grads
ga> open AE_SI12.bilin.xdf
ga> set lon -180 180
ga> set lat 30 90
ga> set mproj nps
ga> set gxout shaded
ga> display snow
ga> draw title Five-day Snow Depth at June 19 2002
```

Figure 20. GrADS code using two-dimensional longitude and latitude

`set mproj nps` indicates that the plot is drawn over a north polar stereographic map. GrADS 2.0a2 gave an error saying that HDF SDS was not fully implemented. This problem disappeared in GrADS 2.0a3, and we could draw the Snow Depth image shown in Figure 21.

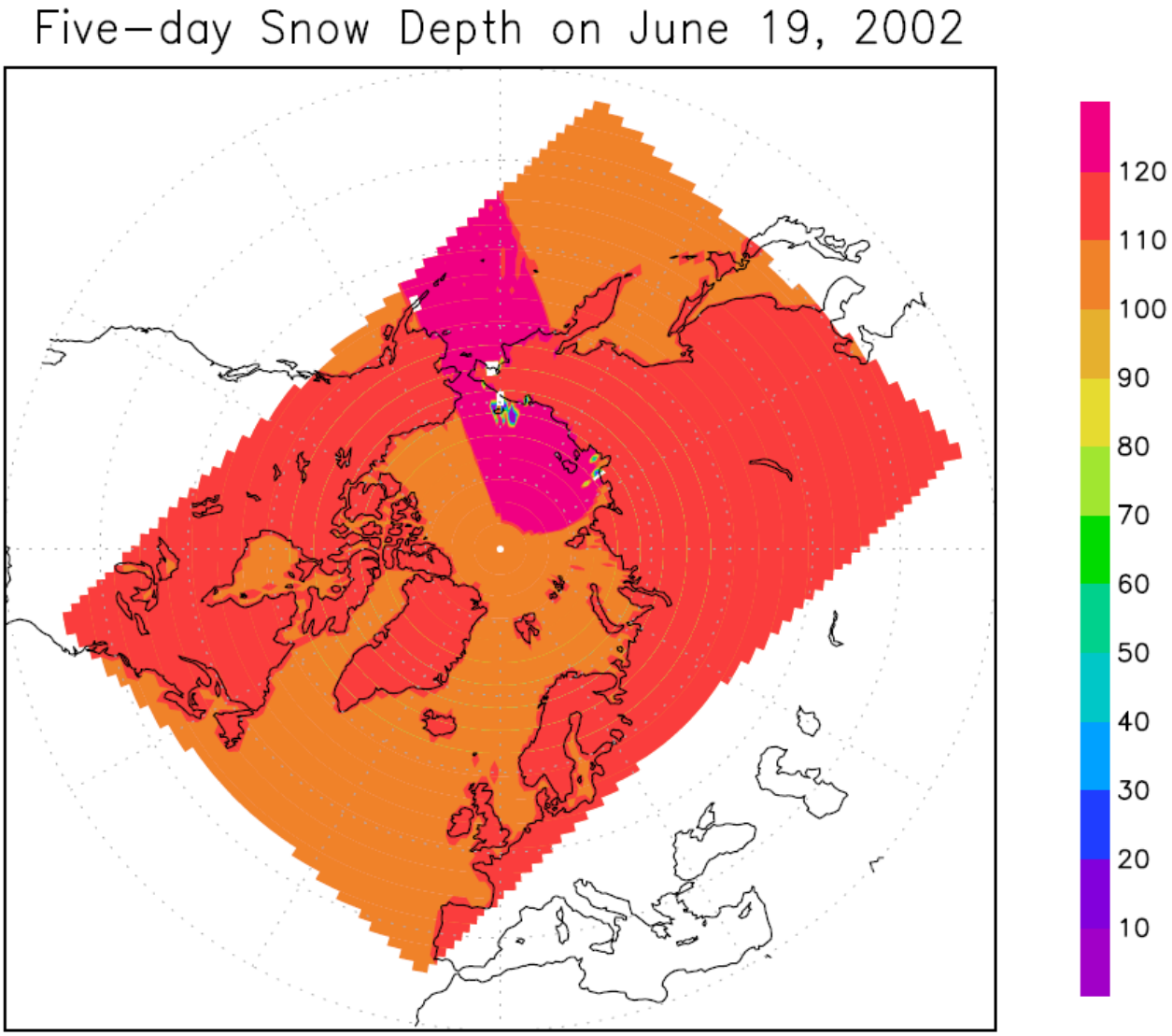


Figure 21. GrADS plot over a polar stereographic map

We believe GrADS can handle all types of projections in HDF-EOS2 grid, as long as the user provides the file that `BILIN` needs. For swath, this can be more difficult because HDF-EOS2 does not have a function similar to `GDll2ij()` to calculate the index for swath data. Users need to interpolate EOS Swath to regular gridded data to obtain the array index for each grid point. The discussion of this topic is beyond the scope of this document.

For more information about using the `BILIN` option to draw a plot with GrADS, please read Appendix 11.1.2.2.

6.3.2.4.2 A Possible Alternative Method

Besides `BILIN`, GrADS provides a convenient method to generate plots for files having the north polar stereographic projection or the south polar stereographic projection. Several parameters are required by GrADS to generate plots for files having these projections.

However, this method may not be straightforward for handling HDF-EOS2 files because HDF-EOS2 and GrADS use polar stereographic projection differently. As of this writing, the authors still cannot draw the correct plot with this method.

6.3.2.5 Visualize a NetCDF-4 Classic Model–compliant HDF5 File that has 2-D Coordinate Variables

This case is very similar to the case described in Section 6.3.2.4.1 because `PDEF BILIN` is not format-dependent. The file that the `BILIN` option requires is exactly the same as the one used in Section 6.3.2.4.1. However, the description file needs to be slightly changed as shown in Figure 22.

```
DSET AMSR_E_L3_SeaIce12km_B02_20020619.hdf
DTYPE netcdf
UNDEF 0 _FillValue
PDEF 608 896 BILIN STREAM BINARY bilin_file
XDEF 360 linear -179.5 1
YDEF 180 linear -89.5 1
ZDEF 1 levels 0
TDEF 1 linear 00Z19jun2002 1mo
VARS 1
SI_12km_NH_SNOWDEPTH_5DAY=>snow 0 y,x SI_12 Snow Depth
ENDVARS
```

Figure 22. Description file for the polar stereographic projection

Compared with Figure 17, only `DTYPE` is different. Opening this description file with the `open` command produces the same plot as shown in Figure 21.

7 PyHDF

7.1 Overview

PyHDF is a Python interface to the HDF4 library. It covers most functions of Scientific Data Sets (SD API), Vdatas (VS API), and Vgroups (V API). PyHDF is not merely a wrapper of HDF4 C API. PyHDF exploits Python features such as the OOP concept and exception handling to make it more convenient.

At the time this document was written, the latest version was 0.7-3 released in July 2005, and it was built with HDF4.2r1. However, it worked with HDF4.2r3.

7.2 Installation

Like other Python libraries, this library comes with the `setup.py` script. Users may need to set `include_dirs` and `library_dirs` for HDF4. Although PyHDF was developed for HDF4.2r1, we could not find any problems with HDF4.2r3. If HDF4 was not built with SZIP, the libraries option needs to be changed.

PyHDF 0.7-3 requires the *Numeric* package that the *numpy* package (22) replaced. PyHDF was successfully built with *Numeric-24-2*.

7.3 How to Use

PyHDF is similar to HDF4 C API in that most functions have similar names and functionality. Although most function names are the same as or similar to corresponding C APIs, they are categorized into a few classes. For example, the SD API is divided into five Python classes, including *SD*, *SDS*, *SDim* and *SDAttr*. Figure 23 shows part of a program that creates a Scientific Data Set.

```
from pyhdf.SD import *
import Numeric

data = Numeric.array(((1, 2, 3),
                      (4, 5, 6)), Numeric.Int16)

# Create an HDF file
sd = SD("hello.hdf", SDC.WRITE | SDC.CREATE)

# Create a dataset
sds = sd.create("sds1", SDC.INT16, (2, 3))

# Fill the dataset with a fill value
sds.setfillvalue(0)

# Set dimension names
dim1 = sds.dim(0)
dim1.setname("row")
dim2 = sds.dim(1)
dim2.setname("col")

# Assign an attribute to the dataset
sds.units = "miles"

# Write data
sds[:] = data

# Close the dataset
sds.endaccess()

# Flush and close the HDF file
sd.end()
```

Figure 23. Python code creating an HDF4 SDS with PyHDF interface

The code in Figure 23 creates an HDF4 file and an SDS object in it. This code is straightforward to those who are familiar with HDF4. As Table 2 shows, many PyHDF interfaces are equivalent to HDF4 C interfaces.

Table 2. PyHDF API and equivalent HDF4 C API

PyHDF API	Equivalent HDF4 C API
SD (constructor)	SDstart
SD.create	SDcreate

SDS.setfillvalue	SDsetfillvalue
SDS.dim	SDgetdimid
SDim.setname	SDsetdimname
SDS.endaccess	SDendaccess
SD.end	SDend

The statement starting with `sd = SD()` creates an SD instance, and it is equivalent to the `SDstart()` function. The SD class implements functions applied to a file such as creating a file and a global attribute. The SD interface identifier that the `SDstart()` API returns does not exist because the SD class of PyHDF encapsulates the data and possible operations.

The statement starting with `sds.units` sets an attribute to the specific SDS object. This is equivalent to the `SDsetattr()` C function. The next statement, `sds[:] = data`, writes the actual values to the file as `SDwritedata()` does.

Both V API and VS API are divided into a few classes and are encapsulated like SD API. This eliminates the use of an identifier, and may improve the readability.

8 GDL

8.1 Overview

GNU Data Language (GDL) is a free clone of Interactive Data Language (IDL), which is an interpreted language used to manipulate scientific data and draw plots. GDL partially supports both HDF4 and HDF5. Additionally, it supports netCDF.

8.2 Installation

Building GDL from source code requires PLplot (23) and GNU Scientific Library (GSL) (24). In particular, building PLplot was not easy, and it generated two errors that required manual fixes. First, it could not detect correct paths for Python executables, libraries and include files. We had to manually specify them. Second, when installing PLplot, a missing `.mod` files error occurred. We had to manually copy three `.mod` files from the `bindings/f95` directory.

Installing GDL requires special attention if HDF4 or HDF5 is built with SZIP. In this case, the `configure` script will fail. Users have to patch the `configure` script to link the HDF4 or HDF5 library with SZIP.

If netCDF-4 built with HDF5 is used, the `configure` script should be fixed further because GDL assumes that netCDF does not depend on HDF5, which is no longer true if netCDF-4 is built with HDF5. In the `configure.in` file, netCDF-4 rule should be located after HDF5 rule. Also, the user should add `-lhdf5_hl` to `LIBS` for HDF5 rule because netCDF-4 uses them.

Since GDL uses HDF5 1.6 API, `H5_USE_16_API` should be defined for the preprocessor if it is inked with HDF5 1.8 or later.

8.3 How to Use

Since GDL has only a thin abstraction layer, it exposes format-specific differences to users. For example, all HDF4-related function names start with `HDF` while all HDF5-related function names start with `H5`. Both HDF4 and HDF5 are partially supported.

8.3.1 Read an HDF4 File

Figure 24 is an example of code that reads data from an HDF4 SDS and stores all values in the `tbocean` variable. These statements can be typed under the GDL environment.

```
FILE_NAME="AMSR_E_L3_RainGrid_B05_200707.hdf"
SDS_NAME="TbOceanRain"

// Open an HDF4 file and an SDS in it
sd_id = HDF_SD_START(FILE_NAME, /read)
sds_index = HDF_SD_NAMETOINDEX(sd_id, SDS_NAME)
sds_id = HDF_SD_SELECT(sd_id, sds_index)

// Read data from the SDS
HDF_SD_GETDATA, sds_id, tbocean

// Close the SDS and the file
HDF_SD_ENDACCESS, sds_id
HDF_SD_END, sd_id
```

Figure 24. GDL code reading data from an SDS in an HDF4 file

One GDL function is mapped to one HDF4 C API as the above example shows. For example, `HDF_SD_START()` is equivalent to `SDstart()`. For more detailed information, refer to the HDF4 reference manual.

8.3.2 Read an HDF5 File

To read an HDF5 file, a different set of functions that resemble HDF5 C API should be used. Figure 25 shows code that opens an HDF5 file and reads all values in a dataset. Although this is equivalent to Figure 24, the code is very different because the file formats are different.

```
FILE_NAME="AMSR_E_L3_RainGrid_B05_200707.h5"
DATASET_NAME="/MonthlyRainTotal_GeoGrid/Data Fields/TbOceanRain"

// Open an HDF5 file and a dataset in it
file_id = H5F_OPEN(FILE_NAME)
dset_id = H5D_OPEN(file_id, DATASET_NAME)

// Read data from the dataset
tbocean = H5D_READ(dset_id)
space_id = H5D_GET_SPACE(dset_id)
dimensions = H5S_GET_SIMPLE_EXTENT_DIMS(space_id)

// Close the dataset and the file
H5S_CLOSE, space_id
H5D_CLOSE, dset_id
H5F_CLOSE, file_id
```

Figure 25. GDL code reading data from a dataset in an HDF5 file

Similar to the HDF4 interface, one GDL function is mapped to one HDF5 C API, which means that users need to know how to use HDF5 C API.

8.3.3 Draw a Plot

IDL provides functions such as `MAP_SET` for mapping points on the earth's surface, but GDL does not implement this, as of June 2008. GDL can draw contours and surfaces, but it lacks the ability to shade surfaces.

After reading data from a file by using the code shown in either Figure 24 or Figure 25, a contour can be drawn by the following command:

```
contour,tbocean
```

Figure 26 shows the result of the above command.

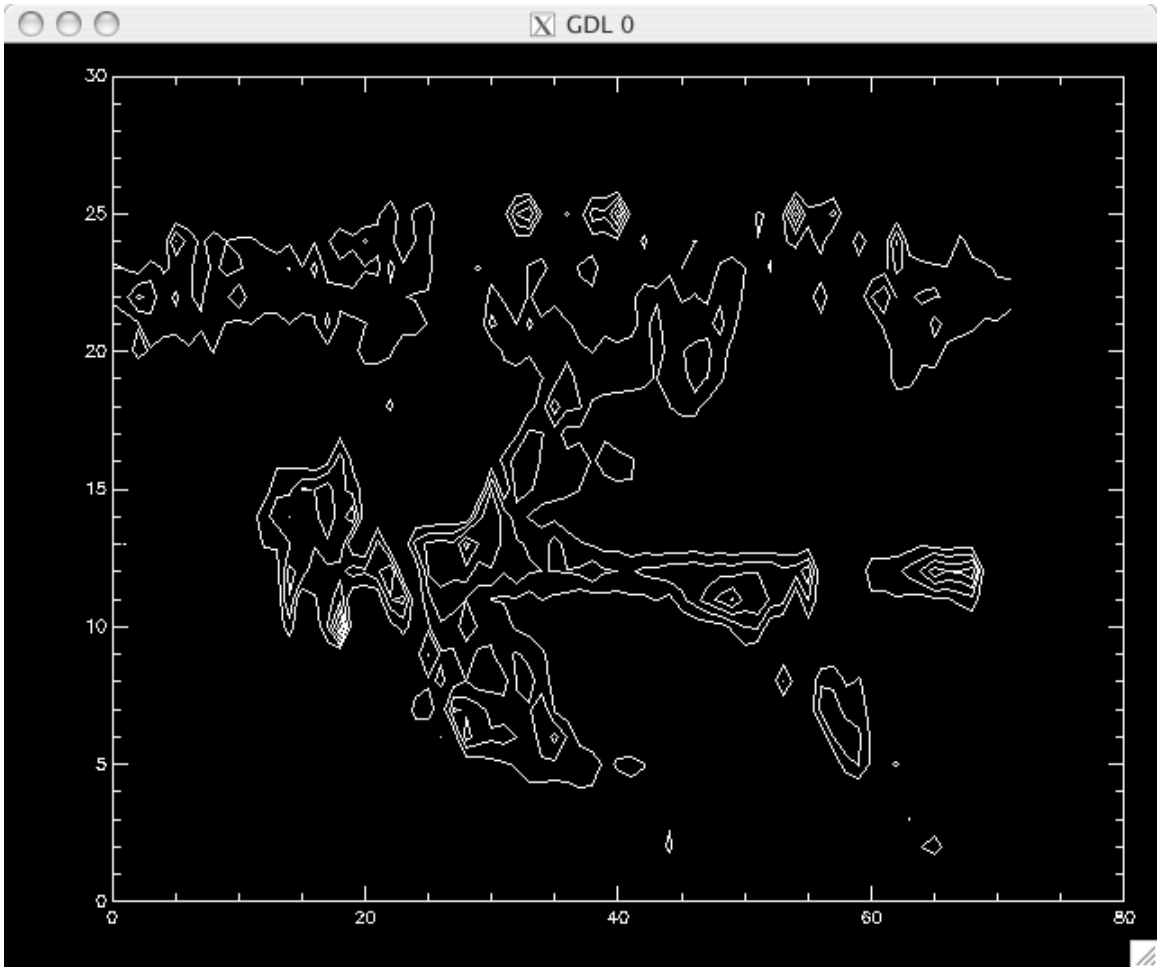


Figure 26. GDL contour plot of rain rate over land in July 2007

The result is not very readable due to the lack of the earth's surface and shading.

9 GDAL

9.1 Overview

Geospatial Data Abstraction Library (GDAL) is a translator library for raster geospatial data formats. As the name implies, GDAL has an abstraction layer that hides format-specific details, which means there is only one GDAL API regardless of file format. For example, one of the following — an HDF4 Vdata, an HDF4 SDS, an HDF4 Vgroup, an HDF5 dataset or an HDF5 group — is mapped to a `GDALDataset` object.

9.2 Installation

A compile error occurred while building GDAL 1.5.2, but this is a known problem. A patch (25) is available, and, we hope this error will not occur in the next release.

GDAL 1.5.2 assumes HDF5 1.6 API; so, `H5_USE_16_API` should be defined if HDF5 1.8 or later is used.

9.3 How to Use

A `GDALDataset` object represents one HDF4 or HDF5 object. Although the term may cause confusion for HDF4 and HDF5 users, it represents HDF4 Vgroups and HDF5 groups as well as HDF4 Vdata, HDF4 SDS and HDF5 datasets.

Since GDAL is written in C++, information provided here will be explained in C++. This section explains how to call some important functions. A complete program showing all information in a file is available in Appendix 11.2.4.

Although examples below show how to use GDAL to handle an HDF5 file, handling HDF4 files is very similar due to GDAL's abstraction layer.

9.3.1 Open a File or an Object

The `GDALOpen()` function opens a file as the following example shows.

```
GDALDataset *ds = (GDALDataset *)
     GDALOpen("hdffile1.h5", GA_ReadOnly);
```

The first argument specifies an HDF4 or HDF5 file, but it can also represent a specific dataset. For instance, if the following string is passed as the first argument, GDAL opens the HDF5 dataset `dataset1` in the HDF5 group `grid1` in the HDF5 file `hdffile1.h5`. HDF5:\ specifies that this is an HDF5 file.

```
"HDF5:\" hdffile1.h5\"://grid1/dataset1"
```

The above string can be fetched by calling the `GetMetadata()` method as Section 9.3.3 explains.

9.3.2 Retrieve Attributes

`GDALDataset` can have metadata as an HDF5 object can have attributes. The `GetMetadata()` method returns a list of metadata. Unlike HDF4 and HDF5, GDAL categorizes metadata, and `GetMetadata()`

takes one argument that specifies the *domain*. Since all attributes in the file are stored in the default domain, an empty string can be passed as the argument.

```
char **metadata = poDataset->GetMetadata("");
```

Each string is formatted as "Name=Value" and null-terminated. If an attribute is not of string type, values are converted into a string.

9.3.3 Open Child Objects

In GDAL, the list of child objects is merely one category of metadata. `GetMetadata()`, introduced in Section 9.3.2, can be used to retrieve the list of child objects. The only difference is that the first argument should be "`SUBDATASETS`". This is a predefined domain for child objects.

```
char **metadata = poDataset->GetMetadata("SUBDATASETS");
```

Assuming that `hdffile1.h5` has two datasets in the `grid1` group, the returned list looks like the following.

```
SUBDATASET_0_NAME=HDF5:"hdffile1.h5"://grid1/dataset1
SUBDATASET_0_DESC=[28x72] //grid1/dataset1 (32-bit floating-point)
SUBDATASET_1_NAME=HDF5:"hdffile1.h5"://grid1/dataset2
SUBDATASET_1_DESC=[28x72] //grid1/dataset2 (32-bit floating-point)
```

As explained in Section 9.3.1, `GDALOpen()` can open a specific object if the first argument indicates one dataset. A value whose name is `SUBDATASET_*_NAME` can be the first argument.

9.3.4 Read Data

A `GDALDataset` object corresponding to an HDF4 SDS or an HDF5 dataset contains one `GDALRasterBand` object. The `GetRasterBand()` method returns one `GDALRasterBand` object.

After getting a `GDALRasterBand` object, one can read or write values by using the `RasterIO()` method. The following is a routine that reads all values in the dataset given by the `poDataset` variable. After executing the following code, the buffer will obtain the data stored in the HDF file.

```
int xsize = poDataset->GetRasterXSize();
int ysize = poDataset->GetRasterYSize();
GDALRasterBand *rb = poDataset->GetRasterBand(1);
float *buffer = new float[xsize * ysize];
rb->RasterIO(GF_Read, 0, 0, xsize, ysize, buffer,
             xsize, ysize, GDT_Float32, 0, 0);
```

10 Conclusion

We explained three applications and two libraries. For NCL and GrADS, we explained how to draw a plot from an HDF4 file, an HDF-EOS2 file, and a netCDF-4-compliant HDF5 file. Both tools could handle two-dimensional coordinate variables. GDL can draw a plot, but it has limitations.

PyHDF provides a Python interface for the HDF4 library. We showed the similarity between PyHDF API and HDF4 C API. GDAL has a thick abstraction layer and provides one unified API for both HDF4 and HDF5 files. We showed how to read files, objects and data, and how to retrieve attributes.

11 Appendix

11.1 Additional explanations

11.1.1 NetCDF-4 Classic Model–compliant HDF5 File

Although NCL does not support HDF5, it does support the netCDF-4 classic model. This subsection explains what the netCDF-4 classic model–compliant HDF5 file is and how to get it from an HDF5 file.

A netCDF-4 classic model–compliant HDF5 file strictly follows the netCDF-3 data model. It has three requirements. The first requirement is that each dimension of an HDF5 dataset associated with a dimension dataset and the dimension should be located under the same or ancestor groups of the HDF5 dataset it is associated with. For this reason, a netCDF-4 classic model–compliant HDF5 file is a netCDF-4-compliant HDF5 file. The second requirement is that all HDF5 datasets should be defined only under the HDF5 root group. The third one is that all HDF5 datatypes should be HDF5 atomic datatypes.

Most netCDF-4-compliant HDF5 files contain several groups, which causes them not to conform to the netCDF-4 classic model. Although it is tedious, it is not impossible to manually convert a netCDF-4-compliant HDF5 file with a non-classic model into one with a classic model.

One way to accomplish this is to use *ncdump* and *ncgen*, which are part of the netCDF-4 package. One can use ncdump to generate a text file from a netCDF-4 file, which is shown in Figure 27. With this text file, one can use any text editor to remove groups, as shown in Figure 28. One may also need to rename some variables if their names are used in multiple groups. Then, the modified text file can be used as input to ncgen, which generates a netCDF-4 file. The output of ncgen can be read by NCL.

```
netcdf AMSR_E_L3_RainGrid_B05_200707 {
  // global attributes:
                  :HDFEOSVersion_GLOSDS = "HDFEOS_V2.13" ;
  group: MonthlyRainTotal_GeoGrid {
  ...
    group: Data\ Fields {
      dimensions:
          lon = 72 ;
          lat = 28 ;
      variables:
          float TbOceanRain(lat, lon) ;
                  TbOceanRain:HDF4_OBJECT_TYPE = "SDS" ;
                  TbOceanRain:HDF4_OBJECT_NAME = "TbOceanRain" ;
          double lon(lon) ;
                  lon:long_name = "longitude" ;
                  lon:units = "degrees_east" ;
          double lat(lat) ;
                  lat:long_name = "latitude" ;
                  lat:units = "degrees_north" ;
          float RrLandRain(lat, lon) ;
```

```
                RrLandRain:HDF4_OBJECT_TYPE = "SDS" ;
                RrLandRain:HDF4_OBJECT_NAME = "RrLandRain" ;
  ...
    }
  }
```

Figure 27. Textual representation of a netCDF-4 file generated by ncdump

```
netcdf AMSR_E_L3_RainGrid_B05_200707_flatten {
  dimensions:
       lon = 72 ;
       lat = 28 ;
  variables:
       float TbOceanRain(lat, lon) ;
              TbOceanRain:HDF4_OBJECT_TYPE = "SDS" ;
              TbOceanRain:HDF4_OBJECT_NAME = "TbOceanRain" ;
       double lon(lon) ;
              lon:long_name = "longitude" ;
              lon:units = "degrees_east" ;
       double lat(lat) ;
              lat:long_name = "latitude" ;
              lat:units = "degrees_north" ;
       float RrLandRain(lat, lon) ;
              RrLandRain:HDF4_OBJECT_TYPE = "SDS" ;
              RrLandRain:HDF4_OBJECT_NAME = "RrLandRain" ;
  data:

   TbOceanRain =
  ...
  }
```

Figure 28. Edited textual representation of a netCDF-4 file

11.1.2 GrADS-related topics

11.1.2.1 Link GrADS with NetCDF-4

This section is only for those who want to build GrADS with netCDF-4. Since GrADS assumes netCDF-3, linking with netCDF-4 requires a few modifications.

Both netCDF-4 and GrADS define `find_dim()` and `find_var()` functions. To resolve the conflict, we renamed `find_dim()` and `find_var()` in the GrADS code.

Another problem is that GrADS assumes that netCDF does not depend on HDF5. While this is true for netCDF-3, netCDF-4 needs HDF5. The clean solution for this is to edit the `configure.in` file and regenerate the `configure` script. In the `configure.in` file, we put `hdf5_hl` and `hdf5` in the dependency list for `nc_libs`.

11.1.2.2 Retrieve Dimension Names and Sizes using HDFView

This section explains how users can retrieve dimensions associated with the given variable using HDFView. Both dimension names and dimension sizes are necessary to write a description file used by `xdfopen` command.

Opening an HDF-EOS2 file shows the hierarchy of the opened file. For example, users will see Figure 29 when the AMSR-E file that contains the rainfall accumulation data is opened.

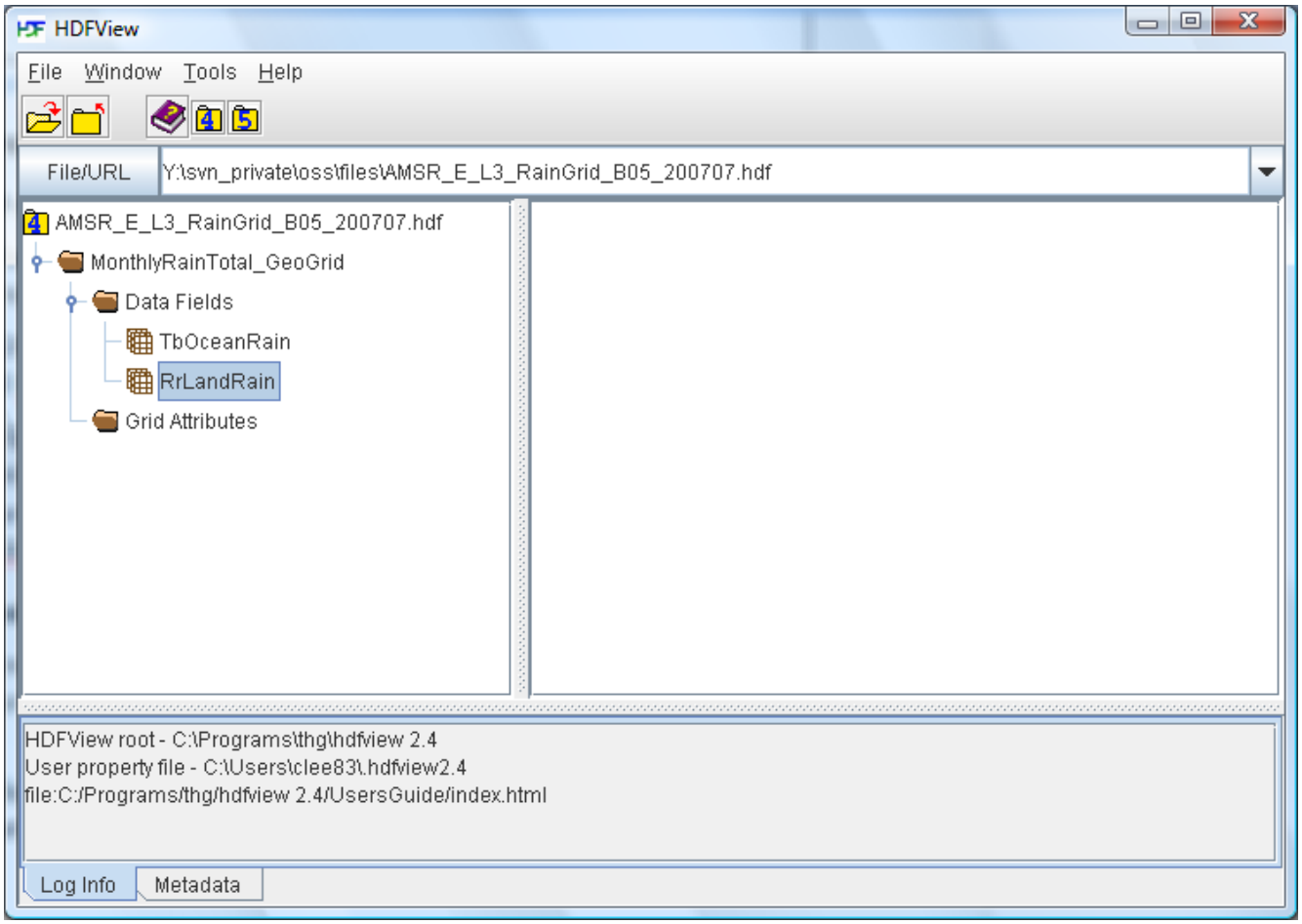


Figure 29. HDFView when an HDF-EOS2 is opened

To retrieve dimensions associated with `RrLandRain` variable, the user can click the right button on `RrLandRain` variable in the tree and choose “Show Properties”. That command shows detailed properties of `RrLandRain` variable as shown in Figure 30.

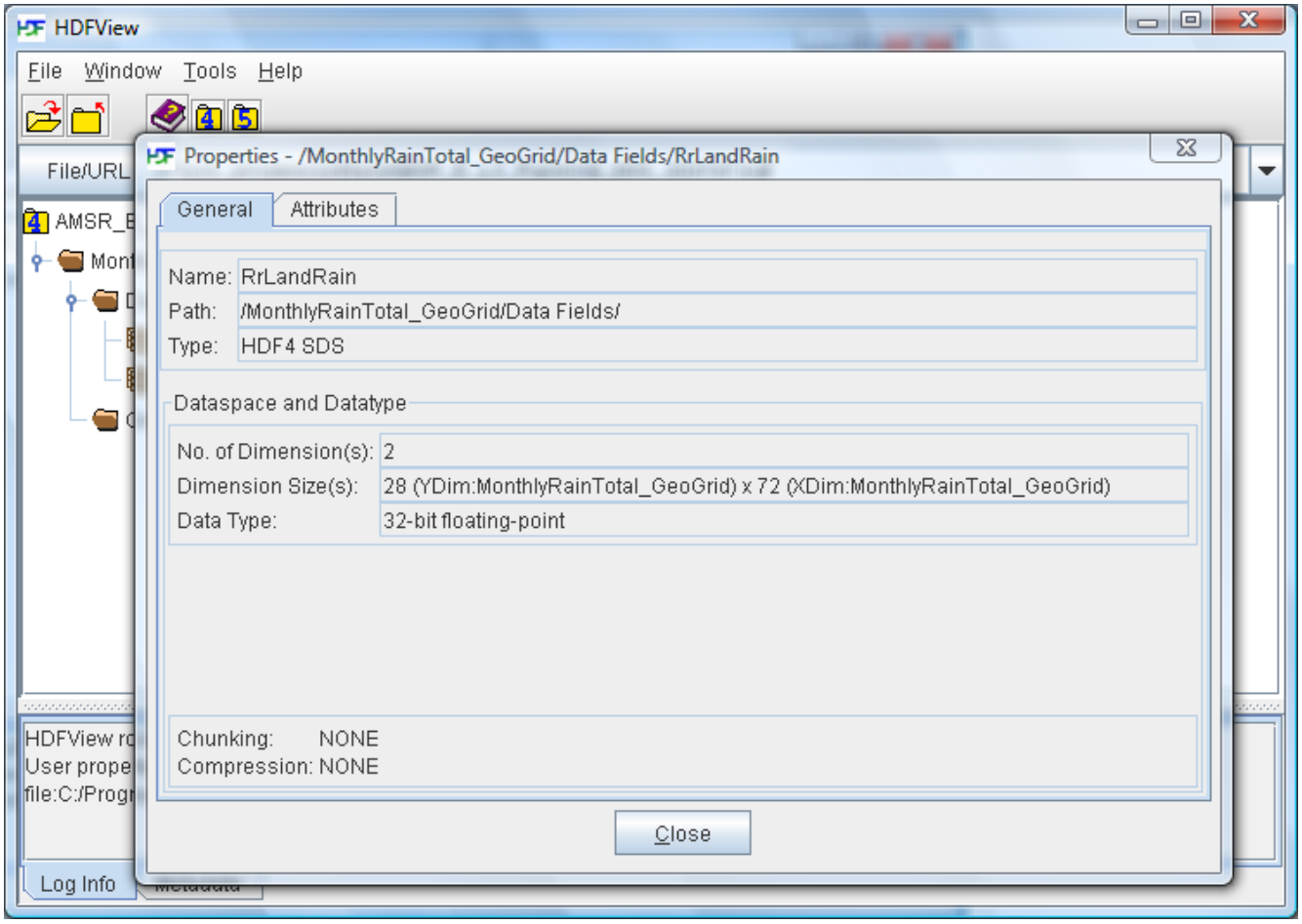


Figure 30. "Show Properties" on a variable

The user can find dimension information from the row marked as "`Dimension Size(s)`" located in the middle of the window. The string can be interpreted as follows:

- The size of the first dimension is 28, and its name is `YDim:MonthlyRainTotal_GeoGrid`.
- The size of the second dimension is 72, and its name is `XDim:MonthlyRainTotal_GeoGrid`.

Use the above information in the first and the second arguments of `XDEF` and `YDEF` in the description file.

```
DSET AMSR_E_L3_RainGrid_B05_200707.hdf
...
XDEF XDim:MonthlyRainTotal_GeoGrid 72 LINEAR 2.5 5
YDEF YDim:MonthlyRainTotal_GeoGrid 28 LINEAR -67.5 5
...
```

Figure 31. Description file for rainfall accumulation over land in July 2007

11.1.2.3 PDEF BILIN Option in GrADS

This section provides more information about the `PDEF BILIN` option with a concrete example.

Suppose that a data field, `Field1`, has 3×3 values, and they are measured at (20˚S, 30˚W), (20˚S, 0˚), (20˚S, 30˚E); (0˚, 30˚W), (0˚, 0˚), (0˚, 30˚E); (20˚N, 30˚W), (20˚N, 0˚), (20˚N, 30˚E).

For this data field, one can write a description file containing the following statements:

```
PDEF 3 3 BILIN STREAM BINARY bilin_file
XDEF 3 LEVELS -30 0 30
YDEF 2 LEVELS -20 20
```

XDEF and YDEF specify that the rectilinear latitude/longitude grid has 3×2 locations: (20˚S, 30˚W), (20˚S, 0˚), (20˚S, 30˚E); (20˚N, 30˚W), (20˚N, 0˚), (20˚N, 30˚E). Figure 32 shows this configuration; nine circles describe where nine elements of Field1 were measured, and six rectangles describe a rectilinear latitude/longitude grid defined by XDEF and YDEF.

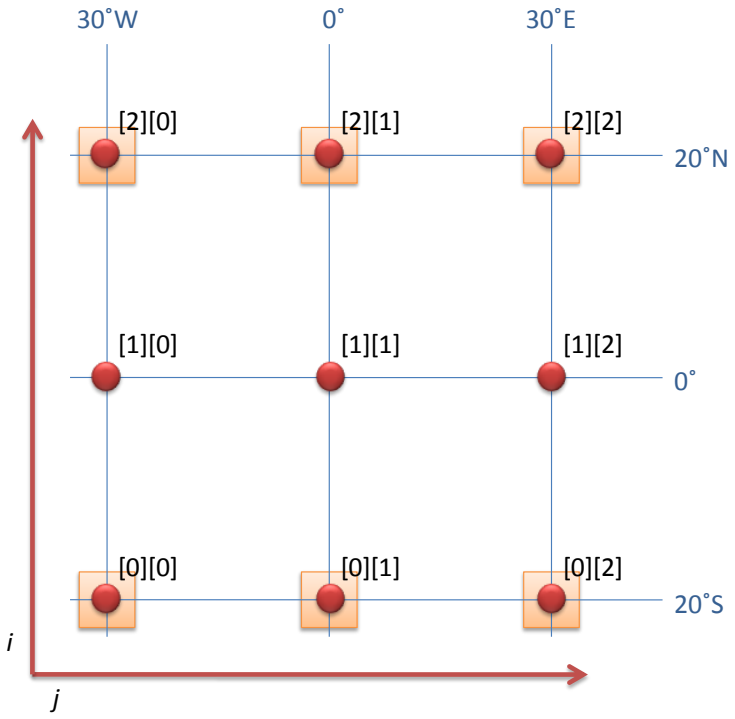


Figure 32. A data field with nine elements and six geolocations defined by XDEF and YDEF

Note that the setting of a rectilinear grid is up to the users, and it may have nothing to do with the data field.

For each location of the rectilinear latitude/longitude grid, GrADS must know which element of the data field is related to this location. The related element is specified by its indices for each dimension: *i* and *j*. For this example, the following table shows the values of *i* and *j*.

Location	i	J	Reason
20˚S, 30˚W	0	0	Field1[0][0] is related to this location
20˚S, 0˚	0	1	Field1[0][1] is related to this location
20˚S, 30˚E	0	2	Field1[0][2] is related to this location
20˚N, 30˚W	2	0	Field1[2][0] is related to this location
20˚N, 0˚	2	1	Field1[2][1] is related to this location
20˚N, 30˚E	2	2	Field1[2][2] is related to this location

Then, the file that `BILIN` needs will contain both six values of *i* and six values of *j*, sequentially; that is, the file will start with 0, 0, 0, 2, 2, 2, 0, 1, 2, 0, 1, 2. After these values, this file should have additional data for wind rotation values. For more information, refer to http://www.iges.org/grads/gadoc/pdef.html.

If the rectilinear latitude/longitude grid has a location such as (10˚S, 15˚E), the values of *i* and *j* will be 0.5 and 1.5, respectively. For this *i* and *j* pair, GrADS does the interpolation using data values from `Field1[0][1]`, `Field1[0][2]`, `Field1[1][1]` and `Field1[1][2]`.

Since calculating the value of *i* and *j* from a location is difficult in most cases, the user needs to consider using an HDF-EOS2 API, `GDll2ij()`, if the file is an HDF-EOS2 grid file. This function returns *i* and *j* based on a location, a projection code and related attributes. One can easily read the projection code and related attributes using HDF-EOS2 API (26) (27).

11.2 Unabridged code

11.2.1 Visualize an HDF4 SDS with NCL

This code is an unabridged version of Figure 2. It reads and visualizes ocean wind fields in an HDF4 SDS.

```
load "$NCARG_ROOT/lib/ncarg/nclex/gsun/gsn_code.ncl"
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_csm.ncl"

begin
  cdf_file = addfile("atlas.ssmi.ver02.level3.5_5day.s950103.hdf", "r")
  u = cdf_file->u10m(0,:,:)
  v = cdf_file->v10m(0,:,:)

  xwks = gsn_open_wks("pdf","ssmi")

  resources = True
  resources@vcGlyphStyle = "CurlyVector"
  resources@tiMainString = "Ocean Wind Fields at Jan 3, 1995"

  ; skip some data to prevent too dense vector
  resources@vcRefLengthF = 0.05
  resources@vcMinDistanceF = 0.015

  resources@tiMainFont = 21
  resources@tiXAxisFont = 21
  resources@tiYAxisFont = 21
  resources@lbLabelFont = 21
  resources@tmXBLabelFont = 21
  resources@tmYLLabelFont = 21
  resources@tmXTLabelFont = 21
  resources@gsnStringFont = 21

  plot = gsn_csm_vector_map_ce(xwks,u,v,resources)

  delete(plot)
  delete(u)
  delete(v)
```

```
  delete(resources)
end
```

11.2.2 Visualize an HDF-EOS2 File with NCL that has 1-D Coordinate Variables

The following code is an *unabridged* version of Figure 4.

```
load "$NCARG_ROOT/lib/ncarg/nclex/gsun/gsn_code.ncl"
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_csm.ncl"

begin
  cdf_file = addfile("AMSR_E_L3_RainGrid_B05_200707.he2","r")

  tbocean = cdf_file->TbOceanRain_MonthlyRainTotal_GeoGrid(:,:)
  tbocean@units = "Unit mm"
  tbocean@_FillValue = -1
  rrland = cdf_file->RrLandRain_MonthlyRainTotal_GeoGrid(:,:)
  rrland@units = "Unit mm"
  rrland@_FillValue = -1

  xwks = gsn_open_wks("pdf","AE_RnGd.hdfeos2")

  resources = True
  resources@gsnDraw = False
  resources@gsmFrame = False
  resources@cnLinesOn = False
  resources@cnFillOn = True
  resources@cnMonoFillPattern = True
  resources@cnMonoFillColor = False
  resources@cnInfoLabelOn = False
  resources@mpFillOn = False

  resources@pmLabelBarDisplayMode = "Always"
  resources@lbOrientation = "vertical"

  resources@tiMainFont = 21
  resources@tiXAxisFont = 21
  resources@tiYAxisFont = 21
  resources@lbLabelFont = 21
  resources@tmXBLabelFont = 21
  resources@tmYLLabelFont = 21
  resources@tmXTLabelFont = 21
  resources@gsnStringFont = 21
  resources@tiMainFontHeightF = 0.015
  resources@tiXAxisFontHeightF = 0.01
  resources@tiYAxisFontHeightF = 0.01
  resources@lbLabelFontHeightF = 0.01
  resources@tmXBLabelFontHeightF = 0.01
  resources@tmYLLabelFontHeightF = 0.01
  resources@tmXTLabelFontHeightF = 0.01
  resources@gsnStringFontHeightF = 0.01

  resources@cnLevelSelectionMode = "ManualLevels"
  resources@cnLevelSpacingF = 50
```

```
  resources@cnMinLevelValF = 50
  resources@cnMaxLevelValF = 600

  resources@cnMissingValFillPattern = 0
  resources@cnMissingValFillColor = 17

  ; define color map
  cmap = (/ (/360.,0.,1./),(/360.,0.,0./), \
  (/220, 0.05, 1.0/), (/220, 0.2, 1.0/), \
  (/220, 0.3, 1.0/), (/220, 0.4, 1.0/), \
  (/220, 0.5, 1.0/), (/220, 0.6, 1.0/), \
  (/220, 0.7, 1.0/), (/220, 0.8, 0.9/), \
  (/220, 0.8, 0.8/), (/220, 0.8, 0.7/), \
  (/220, 0.8, 0.6/), (/220, 0.8, 0.5/), \
  (/220, 0.8, 0.4/), (/220, 0.8, 0.3/), \
  (/220, 0.9, 0.2/), (/60, 0.3, 1.0/) /)
  rgbcmap = hsvrgb(cmap)
  gsn_define_colormap(xwks,rgbcmap)

  ; Brightness temperature derived monthly rain total over ocean
  resources@tiMainString = "AE_RnGd.he2 - Total Rain Rate over Ocean in July
2007"
  plot = gsn_csm_contour_map_ce(xwks,tbocean,resources)

  ; create label bar for fill value
  lbres = True
  lbBoxCount = 1
  lbres@vpWidthF = 0.15
  lbres@vpHeightF = 0.04
  lbres@lbBoxMajorExtentF = 0.60
  lbres@lbFillColors = hsvrgb((/ (/60, 0.3, 1.0/), (/60, 0.3, 1.0/) /))
  lbres@lbMonoFillPattern = True
  lbres@lbLabelFont = 21
  lbres@lbLabelFontHeightF = 0.04
  lbres@lbLabelJust = "CenterLeft"
  lbid = gsn_create_labelbar(xwks,1,(/"fill value"/),lbres)

  ; draw annotation containing label bar
  amres = True
  amres@amJust = "TopRight"
  amres@amParallelPosF =  0.5
  amres@amOrthogonalPosF = -0.5
  annoid = gsn_add_annotation(plot,lbid,amres)

  draw(plot)
  frame(xwks)

  ; Rain rate derived monthly rain total over land
  resources@tiMainString = "AE_RnGd.he2 - Total Rain Rate over Land in July
2007"
  plot = gsn_csm_contour_map_ce(xwks,rrland,resources)

  annoid = gsn_add_annotation(plot,lbid,amres)

  draw(plot)
```

```
  frame(xwks)

  delete(plot)
  delete(tbocean)
  delete(rrland)
  delete(resources)
end
```

11.2.3 Visualize an HDF-EOS2 File with NCL that has 2-D Coordinate Variables

The following code is an unabridged version of Figure 7.

```
load "$NCARG_ROOT/lib/ncarg/nclex/gsun/gsn_code.ncl"
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_csm.ncl"

begin

  cdf_file = addfile("AMSR_E_L3_SeaIce12km_B02_20020619.he2", "r")

  nh18vday = cdf_file->SI_12km_NH_18V_DAY_NpPolarGrid12km(:,:)
  nh18vday@lon2d = cdf_file->GridLon_NpPolarGrid12km
  nh18vday@lat2d = cdf_file->GridLat_NpPolarGrid12km
  nh18vday@unit = "K"

  xwks = gsn_open_wks("pdf","AE_SI12.north.dailyavgt.hdfeos2")

  setvalues NhlGetWorkspaceObjectId()
      "wsMaximumSize" : 500000000
  end setvalues

  resources = True

  gsn_define_colormap(xwks,"wgne15")

  resources@gsnPolar = "NH"
  resources@mpMinLatF = 30
  resources@mpFillOn = False
  resources@cnFillOn = True
  resources@cnLinesOn = False
  resources@gsnSpreadColors = True
  resources@gsnSpreadColorStart = 2
  resources@gsnSpreadColorEnd   = -3

  resources@tiMainString = "18.7 GHz vertical, daily average Tb (x10) at June
19 2002"

  ; [1625, ..., 3058] 4186 fillers
  resources@cnLevelSelectionMode = "ManualLevels"
  resources@cnLevelSpacingF = 200
  resources@cnMinLevelValF = 1500
  resources@cnMaxLevelValF = 3500
  plot = gsn_csm_contour_map_polar(xwks,nh18vday,resources)

  delete(plot)
```

```
  delete(nh18vday)
  delete(resources)
end
```

11.2.4 A Complete Program Built on GDAL

This program recursively dumps all datasets in a file. Although this program is not very practical, this sample shows how users can call GDAL APIs.

```
#include "gdal_priv.h"
#include <iostream>
#include <cstdlib>
#include <string>

#define suicide()  _suicide(__FILE__, __LINE__)

#define DUMP(expr)  std::cout << dump::indent() << expr << std::endl
#define  DUMP_PUSH(expr)    do  {  std::cout  <<  dump::indent()  <<  expr;
dump::push(); } while (false)
#define DUMP_POP()  dump::pop()
namespace dump
{
  static int level = 0;
  static std::string indentation;
  static const char *indent()
  {
    return indentation.c_str();
  }
  static void increase(bool positive)
  {
    level += positive ? 1 : -1;
    indentation = "";
    for (int i = 0; i < level; ++i)
      indentation += "  ";
  }
  static void push()
  {
    std::cout << " {{{" << std::endl;
    increase(true);
  }
  static void pop()
  {
    increase(false);
    DUMP("}}}");
  }
}

static void _suicide(const char *fname, int line)
{
  std::cerr << "suicide at " << fname << ":" << line << std::endl;
  _exit(1);
}

// print all metadata in the given domain
```

```
static void print_metadata(GDALDataset *poDataset, const char *domain)
{
  char **metadata = poDataset->GetMetadata(domain);
  if (metadata) {
    DUMP_PUSH("Metadata " << domain);
    for (char **meta = metadata; *meta; ++meta)
      DUMP(*meta);
    DUMP_POP();
  }
}

// recursively dump elements, attributes and child datasets
static void doit(const char *filename)
{
  GDALDataset *poDataset;

  poDataset = (GDALDataset *)GDALOpen(filename, GA_ReadOnly);
  if (!poDataset) suicide();
  DUMP_PUSH(filename);

  // global description
  {
    DUMP("Driver " << poDataset->GetDriver()->GetDescription());
  }

  // x, y
  int xsize = poDataset->GetRasterXSize();
  int ysize = poDataset->GetRasterYSize();
  DUMP("(X, Y) " << xsize << ", " << ysize);

  // projection
  {
    const char *proj = poDataset->GetProjectionRef();
    if (proj) {
      DUMP("Projection " << proj);
    }
    const char *gcpproj = poDataset->GetGCPProjection();
    if (gcpproj) {
      DUMP("GCPProjection " << proj);
    }
    int gcpcount = poDataset->GetGCPCount();
    for (int i = 0; i < gcpcount; ++i) {
      const GDAL_GCP *gcp = poDataset->GetGCPs() + i;
      DUMP("GCP[" << i << "] " << gcp->pszId << ", " << gcp->pszInfo);
    }
  }
  {
    double geotransform[6];
    if (poDataset->GetGeoTransform(geotransform) == CE_None) {
      DUMP("GeoTransform " << geotransform[0] << ", " << geotransform[1] << ",
" << geotransform[2]);
      DUMP("GeoTransform " << geotransform[3] << ", " << geotransform[4] << ",
" << geotransform[5]);
    }
  }
```

```
  // elements
  {
    int count = poDataset->GetRasterCount();
    for (int i = 0; i < count; ++i) {
      GDALRasterBand *rb = poDataset->GetRasterBand(i + 1);
      int blockx, blocky;
      rb->GetBlockSize(&blockx, &blocky);
      GDALDataType dtype = rb->GetRasterDataType();
      const char *desc = rb->GetDescription();
      DUMP_PUSH("RasterBand[" << i << "] (" << blockx << ", " << blocky << ")
" << GDALGetDataTypeName(dtype) << " : " << desc);
      int hasfill = false;
      double fill = rb->GetNoDataValue(&hasfill);
      if (hasfill) DUMP("Fill " << fill);
      float *buffer = new float[xsize * ysize];
      rb->RasterIO(GF_Read,  0,  0,  xsize,  ysize,  buffer,  xsize,  ysize,
GDT_Float32, 0, 0);
      for (int j = 0; j < ysize; ++j) {
       for (int k = 0; k < xsize; ++k) {
         std::cout << buffer[j * xsize + k] << " ";
       }
       std::cout << std::endl;
      }
      delete [] buffer;
      DUMP_POP();
    }
  }

  // metadata
  print_metadata(poDataset, "");
  print_metadata(poDataset, "GEOLOCATION");
  print_metadata(poDataset, "IMAGE_STRUCTURE");
  print_metadata(poDataset, "SUBDATASETS");

  // recursively dump all child objects
  {
    char **metadata = poDataset->GetMetadata("SUBDATASETS");
    if (metadata) {
      // need to skip SUBDATASET_?_DESC;
      // we only need SUBDATASET_?_NAME here
      for (char **meta = metadata; *meta; meta += 2) {
       const char *name = strstr(*meta, "=");
       if (!name) suicide();
       doit(name + 1);
      }
    }
  }

  GDALClose(poDataset);
  DUMP_POP();
}

int main(int argc, char **argv)
{
```

```
  if (argc != 2) return 1;

  GDALAllRegister();
  doit(argv[1]);
  return 0;
}
```

12 Bibliography

1. *HDF4.* [Online] The HDF Group. http://www.hdfgroup.org/products/hdf4/.

2. *HDF5.* [Online] The HDF Group. http://www.hdfgroup.org/HDF5/.

3. HDF-EOS2. *HDF-EOS.* [Online] http://www.hdfeos.org/software.php#HDF-EOS2.

4. *NetCDF-4.* [Online] Unidata. http://www.unidata.ucar.edu/software/netcdf/netcdf-4/.

5. *NCAR Common Language (NCL).* [Online] http://www.ncl.ucar.edu/.

6. *GrADS.* [Online] http://www.iges.org/grads/grads.html.

7. *PyHDF.* [Online] http://pysclint.sourceforge.net/pyhdf/.

8. *GNU Data Language (GDL).* [Online] http://gnudatalanguage.sourceforge.net/.

9. *Geospatial Data Abstraction Library (GDAL).* [Online] http://www.gdal.org/.

10. NCL Documentation. *NCL.* [Online] http://www.ncl.ucar.edu/Document/index.shtml.

11. GrADS User's Guide. [Online] http://www.iges.org/grads/gadoc/users.html.

12. PyHDF Documentation. [Online] http://pysclint.sourceforge.net/pyhdf/documentation.html.

13. *An Evaluation of HDF Support for NCL, GrADS, PyHDF, GDL and GDAL.* [Online]

14. SSM/I derived global ocean surface-wind components '87-'96 (Atlas et al.). [Online] http://podaac.jpl.nasa.gov/order/order_ocnwind.html#Product079.

15. AMSR-E/Aqua Monthly L3 5x5 deg Rainfall Accumulations. [Online] http://nsidc.org/data/ae_rngd.html.

16. *H4H5TOOLS.* [Online] http://www.hdfgroup.org/h4toh5/.

17. The NetCDF-4 Classic Model Format. [Online] http://www.unidata.ucar.edu/software/netcdf/docs/netcdf/NetCDF_002d4-Classic-Model-Format.html#NetCDF_002d4-Classic-Model-Format.

18. AMSR-E/Aqua Daily L3 12.5 km Tb, Sea Ice Conc., & Snow Depth Polar Grids. [Online] http://nsidc.org/data/ae_si12.html.

19. *NCL Language Reference Guide: Variables.* [Online] http://www.ncl.ucar.edu/Document/Manuals/Ref_Manual/NclVariables.shtml#Subscripts.

20. COARDS Conventions. [Online] http://ferret.wrc.noaa.gov/noaa_coop/coop_cdf_profile.html.

21. *HDFView.* [Online] http://www.hdfgroup.org/hdf-java-html/hdfview/.

22. *NumPy.* [Online] http://numpy.scipy.org/.

23. *PLplot.* [Online] http://plplot.sourceforge.net/.

24. *GNU Scientific Library (GSL).* [Online] http://www.gnu.org/software/gsl/.

25. Ticket #2296. *GDAL.* [Online] http://trac.osgeo.org/gdal/attachment/ticket/2296/gdal-hdf4-UNKNOWN.diff.

26. GDgridinfo 2-143. *HDF-EOS Library Users Guide for the EMD Project Volume 2.* Upper Marlboro, Maryland : Ratheon Company, 2003.

27. GDprojinfo 2-154. *HDF-EOS Library Users Guide for the EMD Project Volume 2.* Upper Marlboro, Maryland : Ratheon Company, 2003.