

HDF5-Friendly OPeNDAP Client Library Prototype

MuQun Yang and Hyo-Kyung Lee

1 Introduction

Many OPeNDAP visualization clients use the **libnc-dap** client library (or the NetCDF Java library for Java-based clients) to display and analyze Earth Science data via DAP protocol. However, one limitation of such approach is that the **libnc-dap** transforms DAP data model into NetCDF data model. We attribute the **libnc-dap** addiction in visualization tools to the fact that there's no easy-to-use generic DAP client library that can retrieve any DAP data model as is. The **libnc-dap** is too easy to use for developers who are already familiar with NetCDF APIs. The visualization tools that can open local NetCDF files can be adapted easily to open remote DAP data because NetCDF APIs for local files and APIs in **libnc-dap** are almost identical.

Thus, we propose a new easy-to-use OPeNDAP client library called **liboc-dap** which doesn't perform any data model transformation. In addition, to retrieve HDF5-specific information that cannot be accessed by **libnc-dap**, the **liboc-dap** will include extra APIs dedicated for HDF5 and HDF-EOS5 files served by a generic OPeNDAP HDF5 server.

2 Problem of libnc-dap based Visualization Clients

As we demonstrated in the technical note[[1](#)], several OPeNDAP visualization clients failed to display a Grid data provided by a default OPeNDAP HDF5 server. For example, in GrADS, we could identify a set of successive **libnc-dap** API calls that expect shared dimensions with certain fixed names like *lat* and *long*. Since each DAP Grid can have its own dimensions without requiring any global shared dimensions among Grids, such expectation from GrADS via **libnc-dap** is too much NetCDF-model-oriented. Thus, the main problem can be stated as follows:

Can we replace NetCDF API calls with more generic OPeNDAP API calls so that it can work with any DAP Grid data?

In addition, we will have a problem of supporting the full capability of OPeNDAP HDF5 server since HDF5-specific extra information like groups and references will be ignored if there are no special dedicated OPeNDAP client APIs. For example, our current server returns a special attribute called "HDF5_ROOT_GROUP" that keeps all group structure information. Also, a Url type data returned by the current OPeNDAP HDF5 server should be regarded as either an object or a regional reference in HDF5 and must de-referenced correctly. Unless they are handled properly by an OPeNDAP client library, it's hard to support fully any HDF5

files that are served by the current OPeNDAP HDF5 server.

3 Relevant APIs

There are two C/C++ DAP client libraries available from OPeNDAP, Inc. First, **OPeNDAP C API**(OCAPI) is a "generic" C implementation of DAP client library. Second, **libdap** library is a C++ implementation of DAP client(and server). We believe that the **OCAPI** is too complex for novice users who are familiar with easy NetCDF APIs. On the other hand, the **libdap** is not written in C and the visualization tools that are written in C cannot benefit from **libdap**.

To design **liboc-dap**, we investigated two software packages: **libnc-dap** and **H5 Lite**. The **libnc-dap** helped us to understand how a DAP client library can work with libdap and why it is popular among developers. The **H5 Lite** provided a rough idea about the minimal set of essential HDF5-related APIs for retrieving HDF5-specific information.

4 HDF5-Friendly OPeNDAP Client Library API

This is a draft for HDF5-Friendly OPeNDAP Client Library APIs.

4.1 I/O

We don't need mode parameter like read/write in I/O since this client library will need read-only access to a remote OPeNDAP resource.

int oc_open(const char* path, int *id)

opens a remote data specified in *path* URL via OPeNDAP protocol and assigns a unique identifier to *id*.

int oc_close(int id)

closes a remote *id* data via OPeNDAP protocol.

4.2 Query

In this section, we'll use a term *file* for a remote OPeNDAP resource that is opened by *oc_open()*.

int oc_find_var(int id, char* name)

inquires if a *name* variable exists in *id* file. If it exists, it returns the id of the first instance of variable that has *name* in it. Otherwise, it returns *-1*.

int oc_find_att(int id, char* name)

inquires if a *name* attribute exists in *id* file. If it exists, it returns the id of the first instance of attribute that has *name* in it. Otherwise, it returns *-1*.

int oc_find_att(int id, char* name)

inquires if a *name* dimension exists in *id* file. If it exists, it returns the id of the first instance of attribute that has *name* in it. Otherwise, it returns *-1*.

int oc_inq(int id, int *ndimsp, int *nvarsp, int *nattsp)

retrieves the number of dimensions, the number of variables and the number of attributes from the *id* OPeNDAP data file.

The number of dimensions(*ndimsp*) will be the largest possible dimensions that a **Grid** variable has in the data file.

If there's no grid, *ndimsp* will be 0.

int oc_inq_array_ndim(int id, int var_id)

gets the number of dimensions of the *var_id* array variable in *id* file.

int oc_inq_array_size(int id, int var_id)

gets the total size of the *var_id* array variable in *id* file. For example, an array *a[2][3][7]* has the size of 42(= 2 * 3 * 7).

int oc_inq_array_type(int id, int var_id)

gets the OPeNDAP data type of the *var_id* array variable in *id* file.

int oc_inq_att_name(int id, int att_id, char* name)

stores the attribute name of *att_id* from *id* file into *name* character buffer.

int oc_inq_dim_size(int id, int dim_id);

gets the size of *dim_id* dimension from *id* file.

int oc_inq_dim_name(int id, int dim_id, char* name);

stores the dimension name of *dim_id* from *id* file into *name* character buffer.

int oc_inq_grid_ndim(int id, int var_id)

gets the number of dimensions of the *var_id* Grid variable.

int oc_inq_grid_type(int id, int var_id);

gets the OPeNDAP data type of the *var_id* Grid variable in *id* file.

int oc_inq_has_grid(int id)

checks if the *id* file has a Grid data type variable.

int oc_inq_has_var(int id, char* name)

checks if the *id* file has a *name* variable. Unlike *oc_find_var()*, it looks for an exact match instead of sub-string match.

int oc_inq_var_name(int id, int var_id, char* name)

stores the variable name of *var_id* from *id* file into *name* character buffer.

int oc_inq_var_type(int id, int var_id)

obtains the OPeNDAP data type of *var_id* variable.

4.3 Attribute

You won't see many APIs in this subsection. Please read section **6 Limitations** to see why.

`int oc_get_att_text(int id, int att_id, char* str)`
retrieves the values of the **top-level String** type attributes in *att_id* attribute and store it into *str* by concatenation.

4.4 Data

In this subsection, *id* means the remote OPeNDAP resource id returned by **oc_open()** API call.

`int oc_get_var_array(int id, int var_id, char* buf)`
retrieves the entire data from *var_id* OPeNDAP Array and store it into *buf*.

`int oc_get_var_array_ce(int id, int var_id, char* buf, char* ce)`
retrieves the subset data specified in *ce* from *var_id* OPeNDAP Array and store it into *buf*.

`int oc_get_var_array_url(int id, int var_id, char** buf);`
retrieves the array of URLs or Strings from *var_id* OPeNDAP Array and store it into *buf* as an array of character strings.

`int oc_get_var_grid(int id, int var_id, char* buf)`
retrieves the entire Array part of data from *var_id* OPeNDAP Grid and store it into *buf*.

`int oc_get_var_grid_ce(int id, int var_id, char* buf, char* ce)`
retrieves the subset specified in *ce* from the Array part of data from *var_id* OPeNDAP Grid and store it into *buf*.

`int oc_get_var_url(int oid, int varid, char* buf);`
retrieves the string value from *var_id* OPeNDAP Url and store it into *buf*.

4.5 HDF5 Specific

These APIs are useful for only HDF5 files that are served by OPeNDAP HDF5 server. Again, *id* means the remote OPeNDAP resource id returned by **oc_open()** API call.

`int oc_hdf5_find_group(int id, char* name)`
inquires if a *name* group exists. If it exists, it returns the group id of a group that has *name* group. Otherwise, it returns *-1*.

`int oc_hdf5_get_children(int id, int parent_att_id, int* children_atts)`
gets the attribute ids of all objects under a *parent_att_id*. By default, the root group will have an attribute id 0. Since every group and variable is mapped to a DAS attribute, the second argument is expected to be an attribute id of a group. See also **oc_hdf5_get_children_size()**.

`int oc_hdf5_get_children_size(int oid, int parent_att_id)`

gets the size of all children under a parent group id.

int oc_hdf5_get_var_url(int id, char* url, char* buf)

treats URL type as HDF5 reference and retrieves the value that it points to. Treating URL type as the HDF5 reference assumes that a URL refers to a variable **within** the same *id* file. If the *url* refers to other location (i.e. it starts with *http://other_server.com/path/...*) or invalid DAP type like group in HDF5, it will return OC_ENOTVAR error.

int oc_hdf5_is_hdf5(int id)

returns 1 if the *id* file is HDF5 by locating the HDF5_ROOT_GROUP attribute. Otherwise, 0.

int oc_hdf5_is_group(int id, int att_id)

returns 1 if the *att_id* attribute in *id* file is group. Otherwise, 0.

4.6 HDF-EOS5 Specific

The following APIs are useful for only HDF-EOS5 files that are served by OPeNDAP HDF5 server.

Again, *id* means the remote OPeNDAP resource id returned by **oc_open()** API call.

int oc_he5_get_lat_delta(int id)

returns the linearly-interpolated latitude grid delta size for a swath file. (see subsection 4.7 OCSwath class for details)

int oc_he5_get_lat_dim_size(int id)

returns the latitude dimension size. If the *id* file is a swath, it returns the linearly-interpolated latitude grid size (see subsection 4.7 OCSwath class for details). Otherwise, it returns the *YDim* dimension size in the *id* grid file.

int oc_he5_get_lat_min(int id)

returns the linearly-interpolated minimum latitude value for a swath file. (see subsection 4.7 OCSwath class for details)

int oc_he5_get_lon_delta(int id)

returns the linearly-interpolated longitude grid delta size for a swath file. (see subsection 4.7 OCSwath class for details)

int oc_he5_get_lon_dim_size(int id)

returns the longitude dimension size. If the *id* file is a swath, it returns the linearly-interpolated longitude grid size (see subsection 4.7 OCSwath class for details). Otherwise, it returns the *YDim* dimension size in the *id* grid file.

int oc_he5_get_lon_min(int id)

returns the linearly-interpolated minimum longitude value for a swath file. (see subsection 4.7 OCSwath class for details)

int oc_he5_get_lev_dim_size(int id)

returns the vertical level dimension size. If the *id* file is a swath, it returns the size of *Pressure* array if such variable exists(0 if *Pressure* variable doesn't exist). Otherwise, it returns the *ZDim* or *nCandidate* dimension size in a grid file.

int oc_he5_grid_to_swath(int id, int lon, int lat)

returns the index of swath point that is the closest to the give *lon* index and *lat* index.

int oc_he5_has_swath(int id)

checks if the *id* file has swath.

int oc_he5_is_eos5(int id)

checks if the *id* file is HDF-EOS5 file by locating the "HDFEOS INFORMATION" group name.

int oc_he5_swath_to_grid(int id, int var_id, int xstart, int xstop, int ystart, int ystop, int z, float* grid)

returns an interpolated data of *var_id* swath at level *z* in a region (*xstart*, *ystart*) ~ (*xstop*, *ystop*) by storing the data into *grid* from the *id* swath file.

4.7 OCSwath class

In order to facilitate visualization of HDF-EOS5 swath data in GrADS, it is necessary to convert a swath format data into a grid format data because GrADS supports only linear grid dimensions, not swath dimensions. The *OCSwath* class is a part of **liboc-dap** library that is responsible for such automatic conversion. Given an HDF-EOS5 swath file, the *OCSwath* class can compute the min/max latitude and longitude and then determine an appropriate resolution(deltas) of latitude and longitude.

Computing min/max lat/lon information is easy. It can go through the data values in the array stored under *Latitude* and *Longitude* variable in a HDF-EOS5 swath file. Determining the resolution of linear grid dimensions can be computed as follows. Let's assume that the dimension size of lat is *x* and the size of *Longitude* array is *z*. One good approximation will be $z = 2x * x$ since longitude is twice bigger than the size of latitude. Therefore, we can obtain the value of *x* from $\text{sqrt}(z/2)$.

The member function called *initialize()* in *OCSwath* class does all conversion necessary for linear grid mapping from swath dimension data.

5 Example

We tested our prototype library by writing a text-based demo application as well as modifying the open source OPeNDAP client GrADS.

5.1 Example Programs

There are three sample programs that use the **liboc-dap** under *oc_test/* directory.

1) *oc_test.c*

This file tests APIs related to both HDF5 and HDF-EOS5 files.

2) *odump.c*

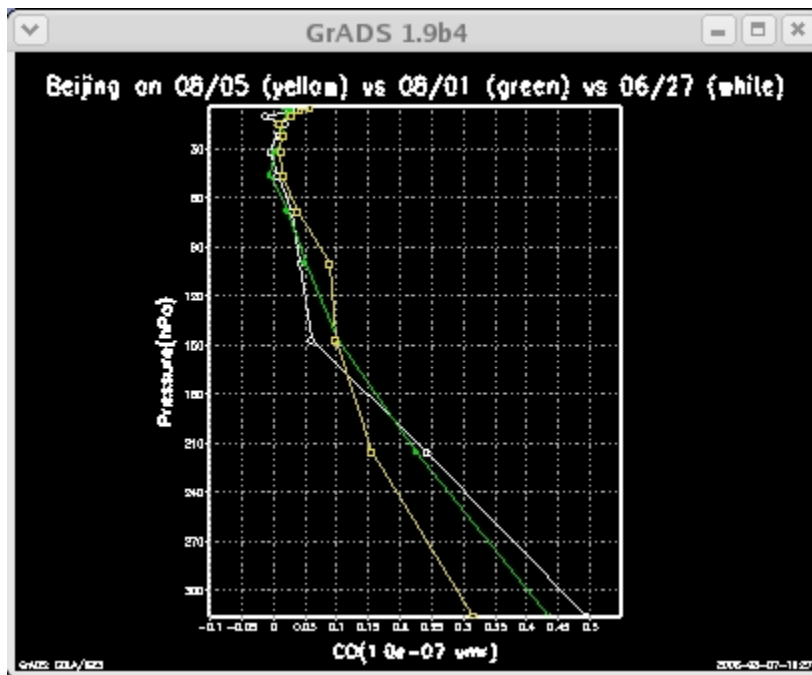
This file fetches a URL and prints out information about the remote HDF5 file.

3) dap2h5.c

This file generates a local HDF5 file from a remote HDF5 file. The original group information will be restored in the local file.

5.2 GrADS Integration

The *gradsdods*, one of executable in GrADS, is linked with **libnc-dap** and displays Grid data over OPeNDAP protocol. We created a new executable called *gradsoc* that is linked with **liboc-dap** that has OPeNDAP client APIs in section 4. By utilizing the HDF-EOS5 specific APIs in section 4.6, we could display OMI/MLS/HIRDLS swath data. We also had a success with MODIS swath data after converting MODIS HDF4 files into HDF5 files using **heconvert**. The below figure demonstrates visualizing vertical profiles of MLS carbon monoxide swath data near Beijing on three different dates using GrADS with **liboc-dap**.



6 Limitations

Due to the prototype nature of **liboc-dap**, there are more limitations than features and users should be careful in using this product.

Due to the current **libdap** error, NASA EOS MLS data cannot be retrieved without turning off DAS retrieval. We suspect that such error occurs because MLS data contains a very long string of attribute which is over the limit of **libdap** string size.

As you can see from the `oc_get_att_text()` API, only the string type of attribute is supported. In addition, nested attributes are not handled at all. The main reason is that we

haven't decided how to represent different types of attributes yet. We're considering XML-style output of all attributes.

References

[1] [Using DAP clients to visualize HDF-EOS5 Grid data](#)

Appendix A: Links

[1] H5 Lite API: http://www.hdfgroup.org/HDF5/hdf5_hl/doc/RM_hdf5lt.html

[2] NetCDF: <http://www.unidata.ucar.edu/software/netcdf/docs/netcdf-c/index.html#Top>