QuickStart - OPeNDAP

2017-10-12

Table of Contents

1. Introduction

OPeNDAP is the developer of client/server software, of the same name, that enables scientists to share data more easily over the internet. The OPeNDAP group is also the original developer of the Data Access Protocol (DAP) that the software uses. Many other groups have adopted DAP and provide compatible clients, servers, and SDKs. OPeNDAP's DAP is also a NASA community standard. For the rest of this document, "OPeNDAP" will refer to the software.

With OPeNDAP, you can access data using an OPeNDAP URL of any database server that supports OPeNDAP. You can do this via command-line, Internet browser, or a custom UI. You can also use other NetCFD complaint tools, such asMatlab, R, IDL, IDV, and Panoply.

Note

that OPeNDAP data is, by default, stored and transmitted in binary form. In addition to its native data representation format, OPeNDAP can get data in the following formats: NetCDF, GeoTIFF, JPEG2000, JSON, ASCII.

This quick start guide covers how to use OPeNDAP in a typical web browser, such as Firefox, Chrome, or Safari, to discover information about data that is useful when creating database queries. For additional information, see the OPeNDAP User Guide.

1.1. Key Terms

- **Constraint Expressions:** Criteria that limits what data is returned from a database. Because most databases will have far more data than you want, you'll want to find out something out about the data and then use that information to constrain your queries.
- **DAS (Data Attribute Structure):** Similar to a DDS but contains information about the data, such as units and the name of the variable.
- **DAP (Data Access Protocol):** A data transmission protocol designed by OPeNDAP specifically for science data. The protocol relies on the widely used and stable HTTP and MIME standards and provides data types to accommodate gridded data, relational data, and time series, as well as allowing users to define their own data types.
- **DDS (Dataset Descriptor Structure):** Provides a description of the "shape" of data in a database, using a vaguely C-like syntax.
- **Hyrax:** The server half of OPeNDAP, developed by the OPeNDAP group.
- **OPeNDAP URL** An OPeNDAP URL is a URL that includes "opendap.org" and enables you to access data on a database server on which OPeNDAP is implemented.

2. What to do With an OPeNDAP URL

Suppose someone gives you a hot tip that there is a lot of good data at...

```
http://test.opendap.org/dap/data/nc/sst.mnmean.nc.gz
```

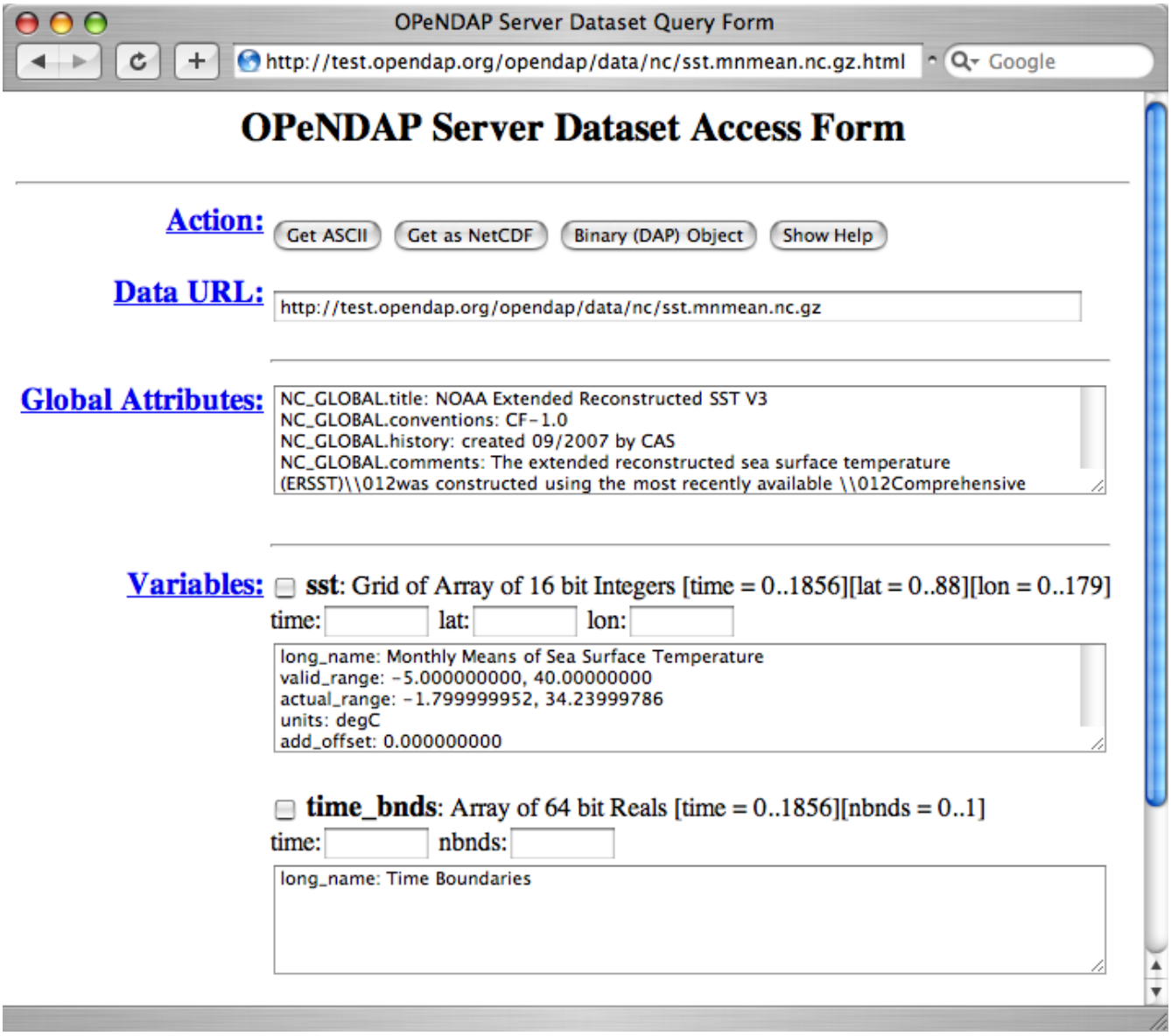
This URL points to monthly means of sea surface temperature, worldwide, compiled by Richard Reynolds of the Climate Modeling branch of NOAA.

The simplest thing you can is to download the data to which the URL points. You could feed it to an OPeNDAP-enabled data analysis package, like Ferret, or you could append **.ascii** and feed the URL to a regular web browser. However, this approach would be less than ideal because there is a large quantity of data that you probably would not want at that URL. Instead, you will want to learn something about the type of data comprised by the database and then use constrained queries to retrieve only the data you need.

NOTE	Because you will rarely want to request an entire archive, OPeNDAP provides sophisticated sub-sampling capabilities, and you need to know something about the data in order to use them.

2.1. An Easy Way: Using the Browser-Based OPeNDAP Server Dataset Access Form

OPeNDAP includes a way to sample data that makes writing a constraint expression somewhat easier than using only a URL to do the same thing. Append **.html** to an OPeNDAP URL, and you will get an OPeNDAP Server Dataset Access Form that simplifies the process for sampling data.



NOTE	As you read this section, it would be useful to have an OPeNDAP Server Dataset Access Form open. Click here for a live OPeNDAP Dataset Access Form

The OPeNDAP Server Dataset Access Form has four main sections:

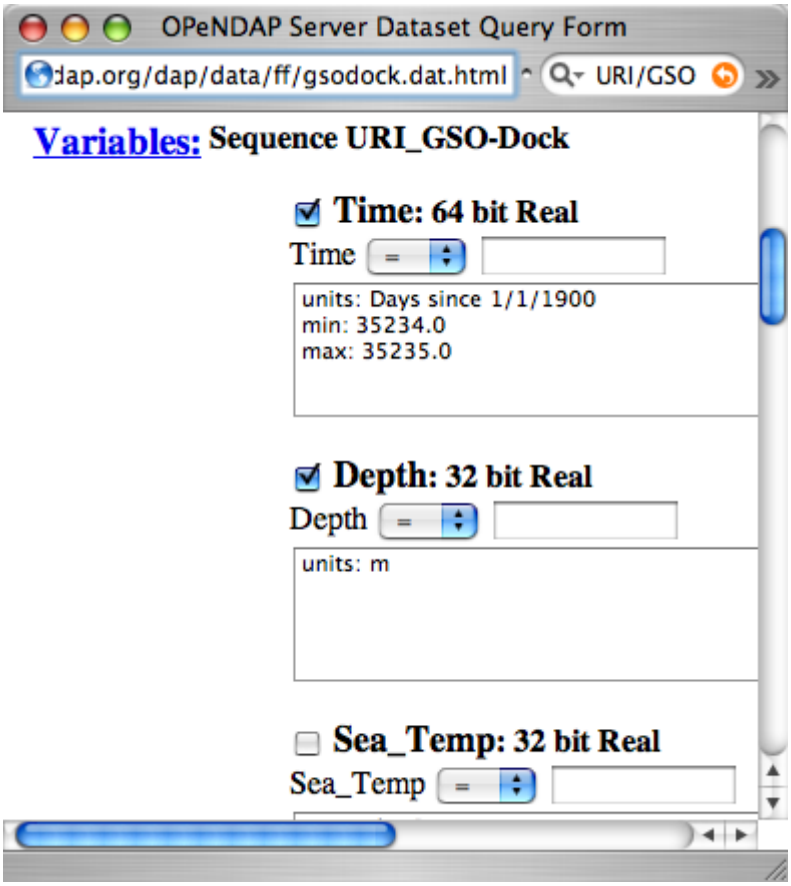
- **Actions:** Use these controls after you have defined your variables:
  - Click **Get ASCII**, and the data request will appear in a browser window, in comma-separated form.
  - Click **Binary (DAP2) Object** to save a binary data file on your local disk.
  - Click **Get as NetCDF 4** to save the file in netCDF format on your local disk. (You can read either of these later with several OPeNDAP clients, by entering the file path instead of a URL.)
- **Data URL:** This is the OPeNDAP URL connected to the data that we are interested in, but it is "unsampled" (that is, there is no constraint expression on it.
- **Global Attributes:** This information is only for reference.
- **Variables:** This is the most important part of the page. For each variable in the dataset, you will see the data description (something like "Array of 32 bit Reals [lat = 0..88]"), a checkbox, a text

input box, and a list of the variable's attributes. If you click the checkbox, you will see the variable's array bounds appear in the text box, and you will see the variable appear in a constraint expression appended to the Data URL at the top of the page. If you edit the array bounds in the text box, press **enter** to place your edits in the Data URL box. Move on down the variable list, editing your request, and experiment by adding and changing variable requests. When you have a request you would like to make, use the Actions buttons at the top of the page.

NOTE

You will see a "stride" mentioned. This is another way to subsample an OPeNDAP array or grid. Asking for **lat[0:4]** gets you the first five members of the **lat** array. Adding a stride value allows you to skip array values. Asking for **lat[0:2:10]** gets you every second array value between 0 and 10: 0, 2, 4, 6, 8, 10.

The OPeNDAP Server Dataset Access Form works for sequence data as well as grids. However, since sequence constraint expressions look different from grid expressions, the form also looks slightly different. The variable selection boxes allow you to enter relational expressions for each variable. Apart from that, the function is exactly the same.

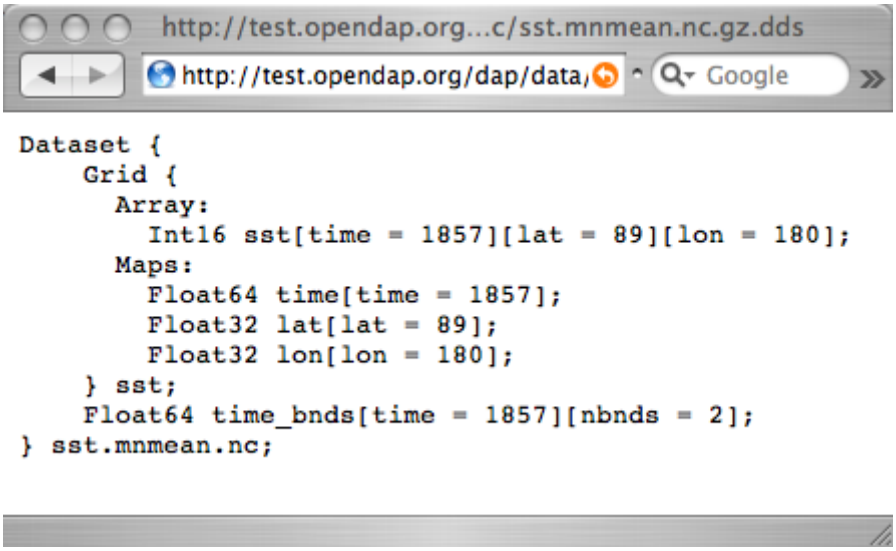


Click here to see a live "sequence" form. Click the checkboxes to choose which data types you want returned, and then add constraint expressions as desired. The data file contains a day's record of changing water properties off a dock in Rhode Island. If you click the *Depth* and *Time* boxes (as in the figure), you will get a record of the tide going in and out twice. You can add conditions by entering values in the text boxes. See what you get when you limit the selection to records where the Depth is greater than 2 meters.

2.2. A More Flexible Way: Using Commands in a Browser

If you would prefer to not use the OPeNDAP Server Data Access Form, you can use just a browser instead. This section describes how to do that.

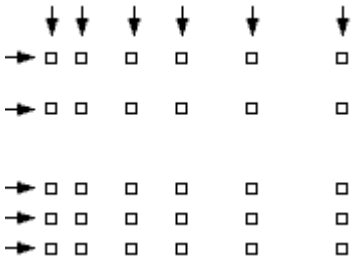
OPeNDAP has sophisticated methods for subsampling data at a remote site, but you need some information about the data first. First, we will look at data's Dataset Descriptor Structure (DDS). This provides a description of the "shape" of the data, using a vaguely C-like syntax. You get a dataset's DDS by appending **.dds** to the URL. Click here to see an example of an OPeNDAP DDS at sst.mnmean.nc.gz.dds. The DDS looks like this:



From the sst.mnmean.nc.gz DDS, you can see that the dataset consists of two different pieces:

- A "Grid" comprising a three-dimensional array of integer values (Int16) called sst; and three "Map" vectors:
  - A 89-element vector called "lat",
  - A 180-element vector called "lon",
  - A 1857-element vector called "time", and
- A 1857 by 2 array called "time_bnds".

The Grid is a special OPeNDAP data type that includes a multidimensional array and map vectors that indicate independent variable values. That is, you can use a Grid to store an array in which rows appear in irregular intervals. Here is a diagram of a simple grid:



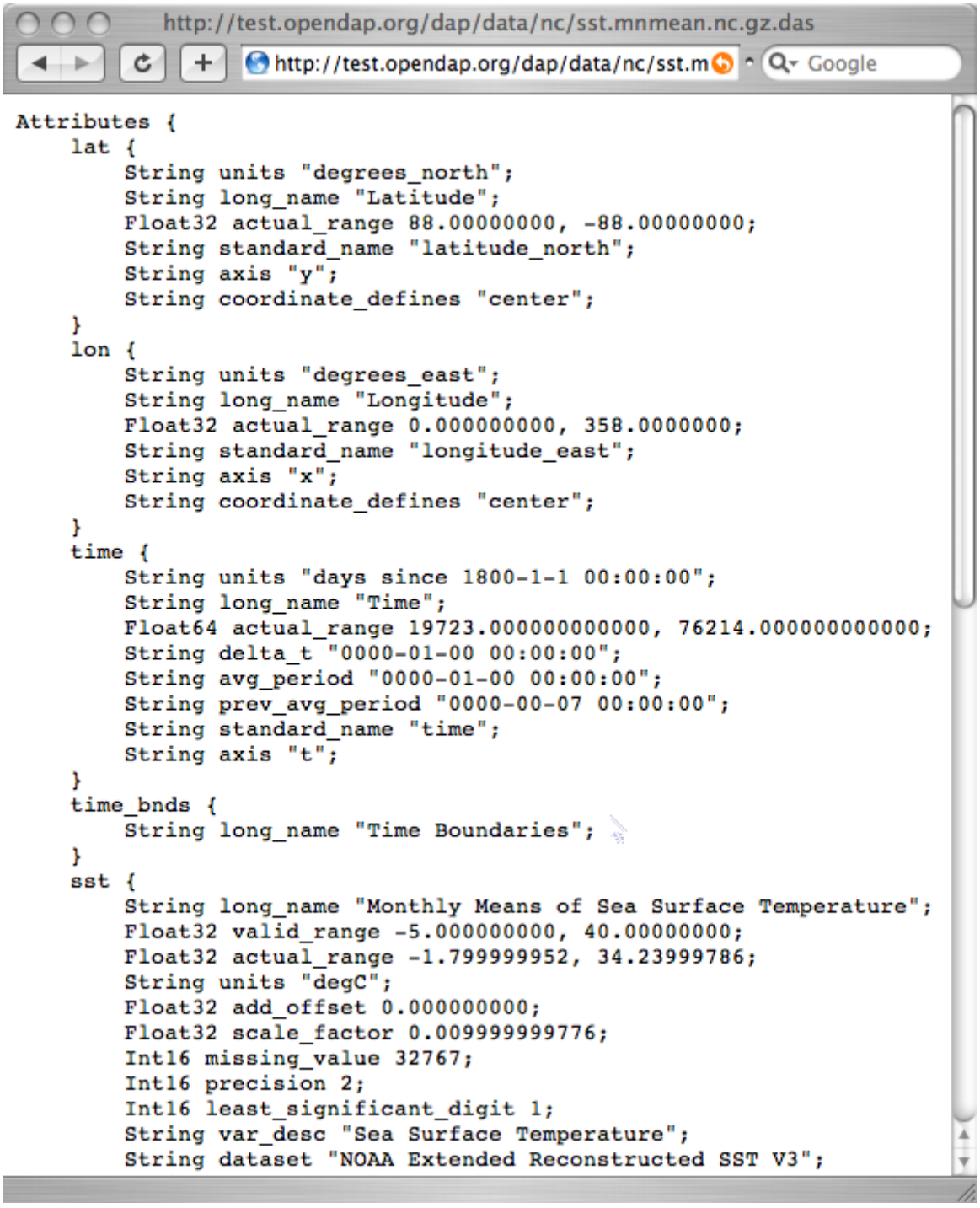
The array part of the grid (like **sst** in the example above) contains the data points measured at each

one of the squares. The X map vector contains the horizontal positions of the columns (like the **lon** vector above). The Y map vector contains the vertical positions of the rows (like the **lat** vector above).

You can also use a grid to store arrays in which the columns and rows occur at regular intervals. You will often see OPeNDAP data stored in this way.

NOTE	The other special OPeNDAP data type worth considering is the *Sequence* . You will see more about this later. There are also *Structures* for representing arbitrary hierarchies.)

You can see from the DDS that the Reynolds data is in a 89x180x1857 element grid, and the dimensions of the Grid are called "lat", "lon", and "time". This is suggestive, but not as helpful as one could wish. To find out more about what the data *is*, you can look at the other important OPeNDAP structure: the Data Attribute Structure (DAS). This structure is somewhat similar to the DDS but contains information about the data, such as units and the name of the variable. Part of the DAS for the Reynolds data that we saw above is shown in the figure below. Click sst.mnmean.nc.gz.das to see the rest of it.



NOTE

Unlike the DDS, the DAS is populated at the data provider's discretion. Consequently, the quality of the data in it (the "metadata") varies widely. The data in the Reynolds dataset used in this example are COARDS compliant. Other metadata standards you may encounter with OPeNDAP data are HDF-EOS, EPIC, FGDC, or no metadata at all.

We can now understand the data better. Apparently the **lat** vector contains latitude, in degrees north, and the range is from 89.5 to -89.5. Since this is a global grid, the latitude values probably go in order. We can check this by asking for just the latitude vector, as in the following:

```
http://test.opendap.org/dap/data/nc/sst.mnmean.nc.gz.ascii?lat
```

What we have done here is to append a constraint expression to the OPeNDAP URL, to indicate how to constrain our request for data. Constraint expressions can take many forms. This guide will only describe a few of them. (You can refer to the OPeNDAP User Guide for more complete information about constraint expressions.) Try requesting the time and longitude vectors to see how this works.

According to the DAS, time is kept in "days since 1800-1-1 00:00:00" in this dataset. This DAS also contains the actual time period recorded in the data (19723 to 76214) which, because of your familiarity with the Julian calendar, you instantly recognize as beginning January 1, 1854.

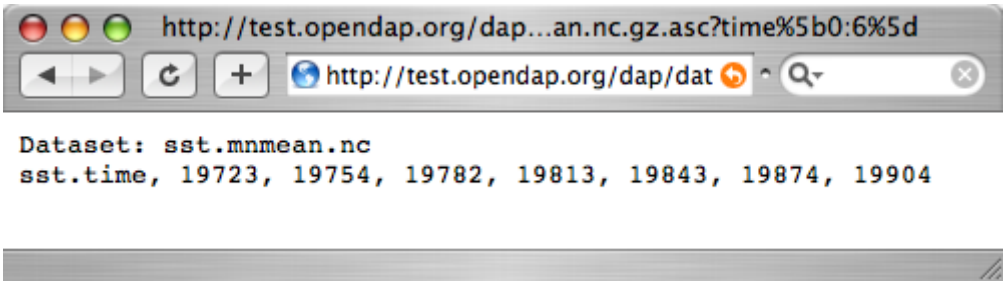
OPeNDAP provides an **info** service that returns, in a single request, all the data we have seen so far. The returned data is also formatted differently, and you can occasionally find server-specific documentation here, as well. Some will find this the easiest way to read the attribute and structure information described above. You can see what information is available by appending **.info** to a URL, like this:

```
http://test.opendap.org/dap/data/nc/sst.mnmean.nc.gz.info
```

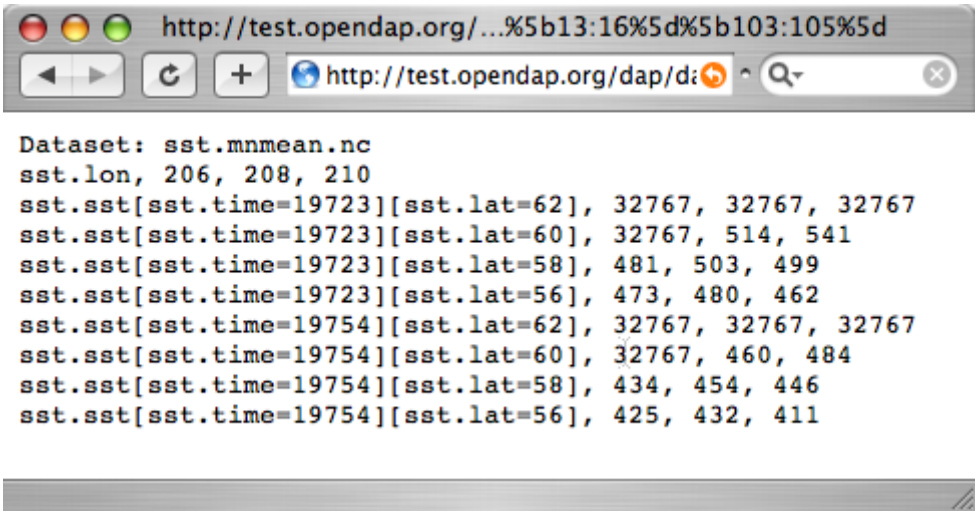
2.2.1. Peeking at Data

Now that we know a little about the shape of the data and the data attributes, we will look at some of the data.

You can request a piece of an array with subscripts, just like in a C program, Matlab, or many other computer languages. Use a colon to indicate a subscript range. For example, http://test.opendap.org/dap/data/nc/sst.mnmean.nc.gz.ascii?time[0:6] will produce a time vecor that looks like this:



If you are interested in the Reynolds dataset, you are probably more interested in the sea surface temperature data than the dependent variable vectors. The temperature data is a three-dimensional grid. To sample the sst grid, you just add a dimension for time: "...sst/mnmean.nc.ascii?sst[0:1][13:16][103:105]". Click here to see this example in action. You will see something like this:



Notice that when you ask for part of an OPeNDAP grid, you get the array part along with the corresponding parts of the map vectors.

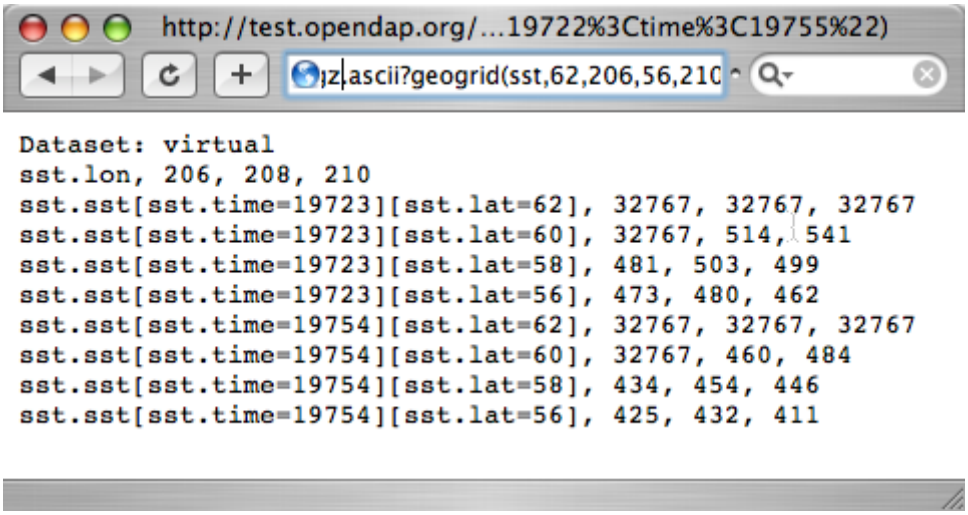
One potentially confusing thing about our request is that we requested the time, latitude, and longitude by their position in the map vectors, but in the returned information they are referenced by their values. That is, we asked for the 0th and 1st time values, but these are 19723 and 19754. We also asked for the 103rd, 104th, and 105th longitude values, but these are 206, 208, and 210 degrees, respectively. The value 434 in the return can be referenced as "... sst/mnmean.nc.ascii?sst[1][15][103]". Click here to see this in action.

Note that the sst values are in Celsius degrees multiplied by 100, as indicated by the **scale_factor** attribute of the DAS. Further, it's important to remember that with this dataset the data were obtained by calculating spatial and temporal means. Consequently, the data points in the **sst** array should be ignored when the value is the missing data flag (32767) as these pixels are probably coincident with land (although there can be other reasons for missing data).

Server Functions: Looking at Geo-Referenced Data Using Hyrax

There are a number of different DAP servers that have been developed by different organizations. Hyrax, the DAP server developed by the OPeNDAP group, supports access to geo-referenced data using lat/lon coordinates. You probably noticed that the array and grid indexes used so far are not very intuitive. You can see the data are global and are indexed by latitude and longitude, but in the previous example we first looked at the lat and lon vectors, saw which indexes corresponded to which real-world locations, and then made our accesses using those indexes.

Hyrax supports a small set of functions which can perform these look-up operations for you. For example, we could rewrite the example above like this: "... sst/mnmean.nc.gz.ascii?geogrid(sst,62,206,56,210,"19722<time<19755")". Click here to see this in action. The results look like this:



The Syntax for `geogrid()` is: "geogrid(grid variable, upper latitude, left longitude, lower latitude, right longitude, *other expressions*)", where *other expressions* must be enclosed in double quotes, and can be one of these forms:

- variable relop value
- value relop variable
- value relop variable relop value

Relop stands for one of the relational operators: <,>,⇐,>=,=,!=. **Value** stands for a numeric constant, and **Variable** must be the name of one of the grid dimensions. You can use multiple clauses by separating them with commas, but each clause must be surrounded by double quotes. For example, the following is yet another way to get the same return data as the above example:

...mnmean.nc.gz.ascii?geogrid(sst,62,206,56,210,"19722<time","time<19755")

You can figure out which functions are supported by Hyrax by calling the server function version(). This will return an XML document that shows each registered function and its version.

To find out how to call each function, you can call it with an empty parameter list and get some documentation for that function. For example, try ...?geogrid().

NOTE	Other severs, such as ERDAP, support alternative ways of doing similar operations.

Creating Server Function Pipelines

Server functions can be composed to form pipelines, feeding the value of one function to another. Since the values in this data set are scaled up by a factor of 100, we can use the *linear_scale()* function to scale the result using...

```
y = mx + b
```

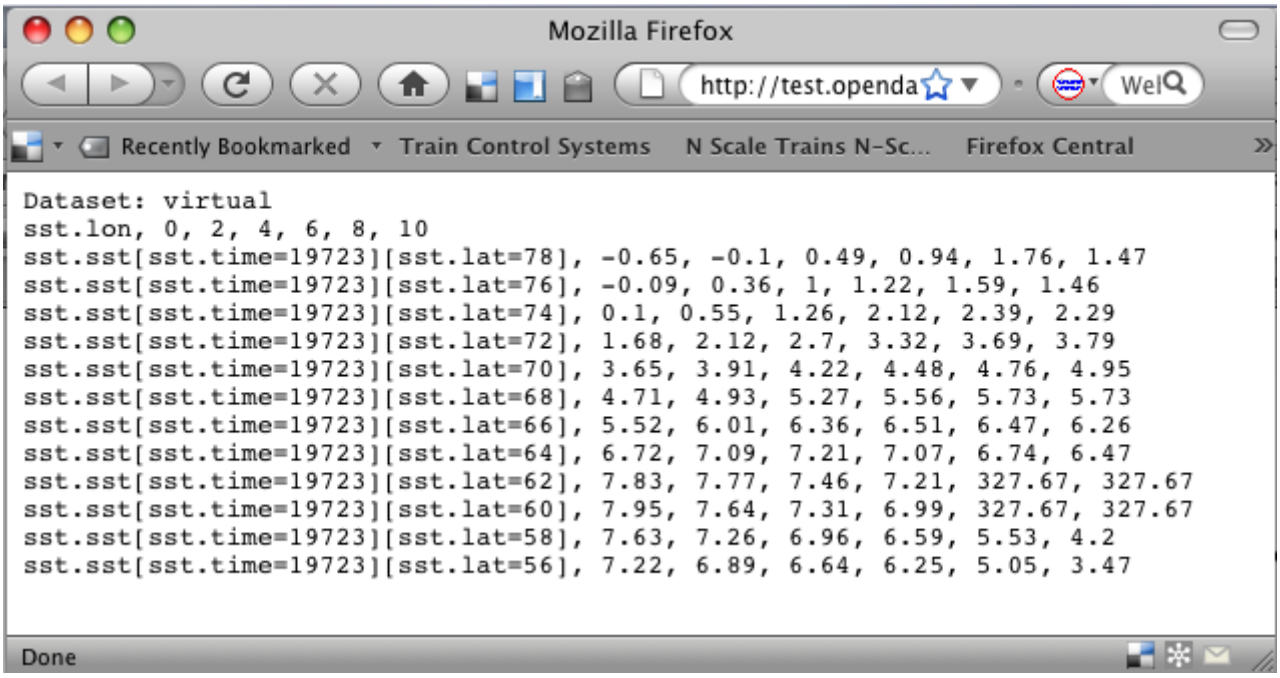
...where **m** is the scale factor and **b** offset. The *linear_scale()* function syntax is:

- linear_scale(variable, scale factor, offset)

- linear_scale(variable)

Use the first form when you want to specify **m** and **b** explicitly or the second form when Hyrax can guess the values using data set metadata. (Note: You will get an error if the server cannot figure out value to use). For example,

…nc.gz.ascii?linear_scale(geogrid(sst,78,0,56,10,"time=19723"),0.01,0) produces the following:



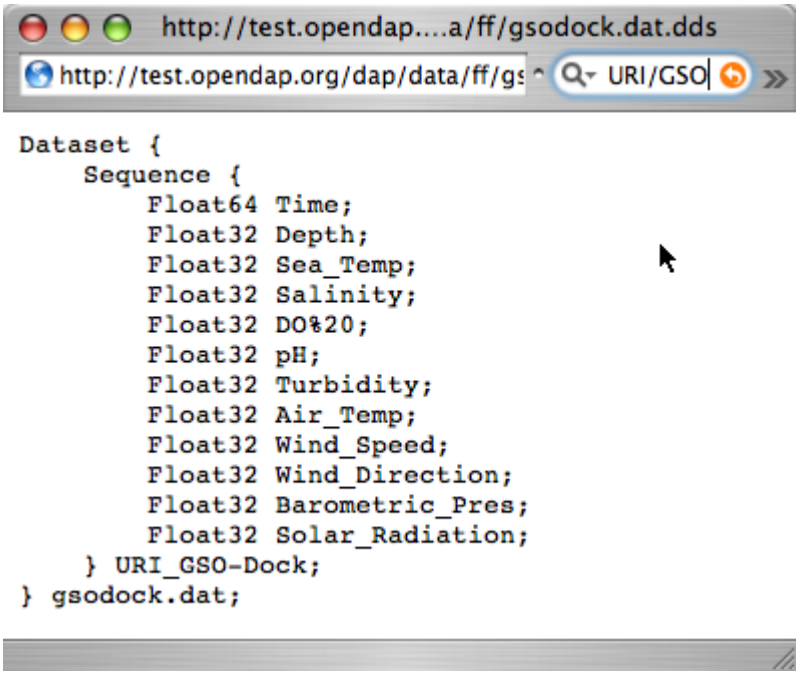
2.2.2. Working With Sequence Data

Gridded data works well for satellite images, model data, and data compilations such as the Reynolds data we have just looked at. Other data, such as data measured at a specific site, are not so readily stored in that form. OPeNDAP provides a data type called a "sequence" to store this kind of data.

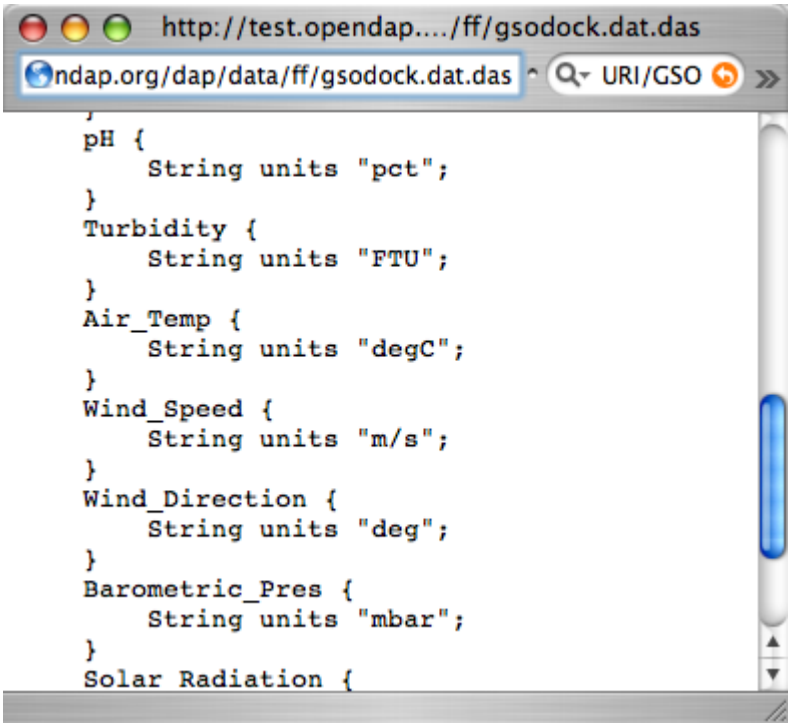
A sequence can be thought of as a relational data table, with each column representing a different data variable, and each row representing a different measurement of a set of values (also called an "instance"). For example, an ocean temperature profile can be stored as a Sequence with two columns: pressure and temperature. Each measurement is a pressure and a temperature and is contained in one row. A weather station's data can be stored as a sequence with time in one column and each weather variable in another column. You can find a good example of a sequence at http://test.opendap.org/dap/data/ff/gsodock.dat

This is a 24-hour record of measurements at a weather station on a dock in Rhode Island. Each record consists of a dozen different variables, including air temperature, wind speed, and direction, as well as depth, temperature, and water salinity The data is arranged into 144 measurements of each of the twelve variables.

Ask for the DDS, and you will see the twelve variables, all contained in a Sequence called URI_GSO-Dock:



The DAS contains the units for each data type and some additional information:



To select the data you want from a server, use a constraint expression, just as you did with the gridded data above. Now, though, the constraint contains two kinds of clauses. One is a list of variables you wish to have returned (the **projection** clause), and the other is the conditions under which they should be returned (the **selection** clause). For example, if you want to see salinity data read after noon that day, try this:

…gsodock.dat.ascii?URI_GSO-Dock.Salinity&URI_GSO-Dock.Time>35234.5

Selection clauses can be stacked endlessly against a projection clause, allowing all the flexibility most people need to sample data files. Here is an example of applying two conditions:

…gsodock.dat.ascii?URI_GSO-Dock.Salinity&URI_GSO-Dock.Time>35234.5&URI_GSO-Dock.Depth>2

Try it yourself with three or four conditions or more.

3. Finding OPeNDAP URLs

Data often comes in the form of a URL enclosed in an email message, and there are several other ways to find data served by OPeNDAP servers.

3.1. Google

Use Google to search for 'OPeNDAP Hyrax' or to search for 'OPeNDAP <terms>' or 'Hyrax <terms>'. For example, Google *OPeNDAP sea surface temperature*.

3.2. GCMD

The Global Change Master Directory provides a huge amount of earth science data and catalogs OPeNDAP URLs for the datasets that have them. You can search on "OPeNDAP" from the main page to find many of these datasets.

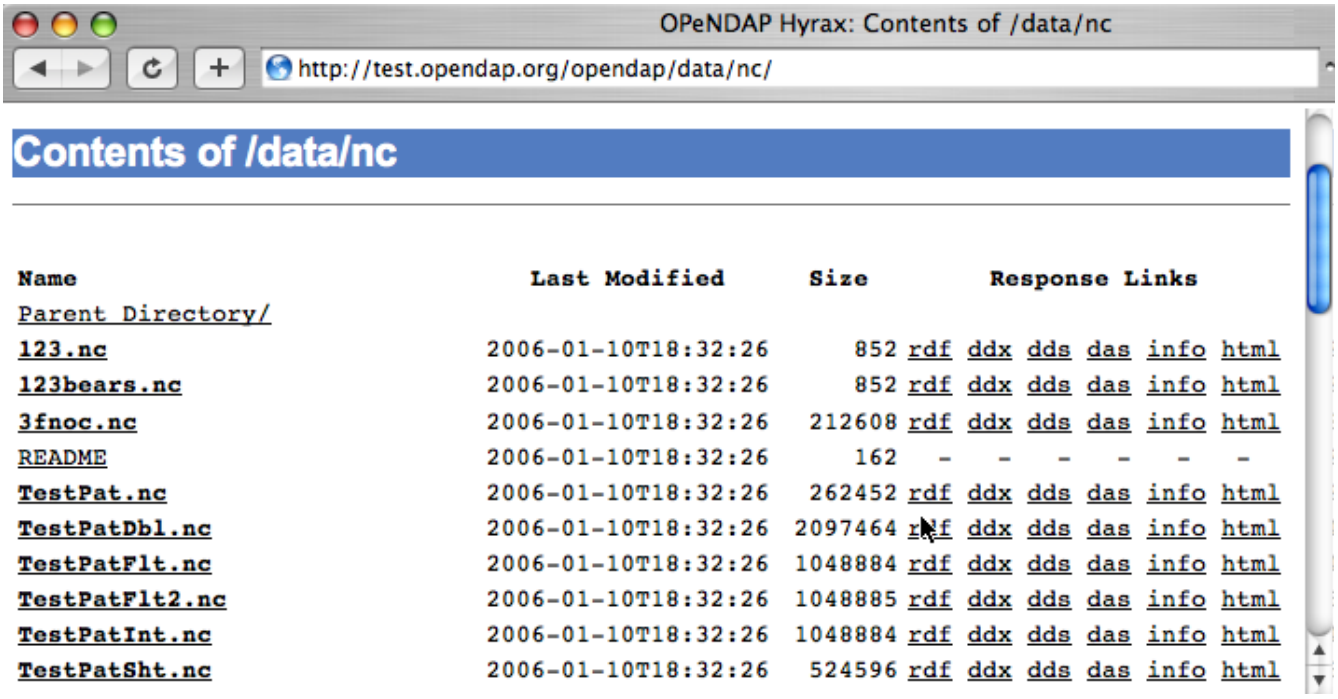
If you make that search, check the list for the Reynolds data; it should be there.

3.3. Web Interface

Many sites that serve one OPeNDAP dataset also serve others. The OPeNDAP web interface (if it is enabled by the site) allows you to check the directory structure for other datasets. For example, we will look at the Reynolds data we saw previously: http://test.opendap.org/dap/data/nc/sst.mnmean.nc.gz.html

If we use the same URL, but without the file name at the end, we can browse the directory of data: http://test.opendap.org/dap/data/nc/

The OPeNDAP server checks to see whether the URL is a directory, and if it is, it generates a directory listing, like this:



You can see from the directory listing that the monthly mean dataset that we have been looking at is accompanied by a host of other datasets. The site you are looking at is our test data sit. We use these datasets to run many of our nightly tests. All of the files in the the */data/nc* directory are stored in NetCDF files; other directories under */data* hold data stored in other file types.

NOTE

In general, this list is produced by an OPeNDAP server and this feature works on all servers. However, it only really understands OPeNDAP data files, so other file types will simply be sent without any interpretation. This can be useful if the 'other file' happens to be a README or other documentation file since this makes it simple to serve data stored in files and documented using plain text files. Essentially, the person or organization providing data does not need to do anything besides installing the server (Hyrax).

4. Further Analysis

This guide is about forming an OPeNDAP URL. After you have figured out how to request the data, there are a variety of things you can do with it. (OPeNDAP software mentioned here is available from the OPeNDAP Home Page.)

- Use a generic web client like **geturl** (a standard part of the OPeNDAP software) or free programs such as wget, curl or **Chrome** to download data into a local data file. To be able to use the data further, you will probably want to download the data using one of alternative response types like the ASCII version (by using the **.ascii** suffix on the URL, as in the examples shown above) or GeoTIFF, NetCDF3, Jpeg2000, etc., using the suffix appropriate for those formats.
- Any tool that uses the Java- or C-NetCDF API will work with OPeNDAP. For example, Matlab has built-in support for OPeNDAP; Matlab supported NetCDF calls can be used with DAP datasets. Other tools that are built on NetCDF API also read data from OPeNDAP servers. A free tool similar to Matlab, GNU Octave, is also supported. The R Project for Statistical Computing can also read data from OPeNDAP servers. The Ferret and GrADS free data analysis packages both

support OPeNDAP. You can use these for down loading OPeNDAP data and for examining it afterwards. (There are limitations. For example, Ferret may not be able to read datasets served as Sequence data.)

NOTE	For information about NetCDF compliant tools, see the NetCDF Compliant Tools in the User Guide.

- Other tools, that are Java NetCDF compliant, also function with OPeNDAP clients; for example, IDV and Panoply.
- If you have written a C NetCDF or Java NetCDF compliant data analysis program, you can probably read data from OPeNDAP servers.

The use of these clients, like the ways in which you can analyze the data you find, is beyond the scope of this document.