# DODS Catalog Services Preliminary Design

James Gallagher

January 22, 1999

## Contents

# 1 Introduction

The DODS Catalog Services provide three distinct functions. They will provide a way to choose individual URLs from datasets which contain many URLs, provide a standard namespace for datasets and map queries stated in scientific terms into data storage terms. The three problems addressed by the catalog are unified by the fact that they all relate to making datasets more generic and abstract. Thus it makes sense to think of solving them with a single service within DODS.

In DODS a dataset is something that can be referred to using the file part of a URL. Within that dataset, DODS' constraint expressions can extract data very flexibly. However, real-world datasets are often composed of many files. DODS has currently has no way to select among those files; to it those datasets look like collections of datasets. The first part of the catalog services addresses this problem by providing a way to choose files in a multi-file dataset.[1] See Section 2.

Using the File Server, users will typically have to access multi-file datasets twice to get data values. This is cumbersome and not the ideal that DODS is designed to achieve. The problem lies in the way DODS sees files; the current design treats each file as a separate dataset. To address this, the File Server design will evolve into a more universal service which can map queries among files which comprise one logical dataset and combine the separate return values into a single cohesive object. Hopefully this solution will, in turn, evolve into something that can map queries over several logically distinct data sets.

DODS' guiding philosophy is that creators of datasets know best how to organize those data. Thus DODS does not impose a name standard requiring, for example, that time be represented a certain way. Unfortunately, this is that it makes comparison between datasets hard and requires users learn something about the internal organization of each dataset. The second part of the catalog services provides a standard namespace which may be layered over a dataset's existing names. Creators of datasets (or people who install DODS servers after the fact) may choose to provide a mapping between the names used in a given dataset and one, or more, of the DODS namspaces. See Section 3.

[1] In the past this was called a 'Catalog Server'. At the same time that phrase was used to mean other things. In this paper I named this piece of software 'File Server' to reduce confusion.

The third part of the catalog services will map queries expressed in scientific terms into the correct syntax and namespace of a particular dataset. For most queries this will work only when basic DODS servers support the catalog services, particularly the standard names service. See Section 3

Development of the catalog services will proceed by first building servers that solve problems one and two. That is, we will build servers that can map queries of multi-file datasets onto a list of URLs into that dataset and we will provide a simple, standard, name space for oceanographic data. Once these parts are complete we will proceed to the third problem and generalization of the first problem.

# 2 Building File Servers

The File Server (FS) provides a way for clients to choose individual files from datasets composed of many files. The DODS design views each file as a distinct dataset. Thus the constraint mechanism cannot be used to choose between files in a multi-file datasets since, to DODS, that dataset appears to be a collection of datasets.

A FS is a DODS dataset which describes another dataset. Typically a FS contains a Sequence[2] that has columns for date, time, location, etc. along with a column for a URL which matches those values. When a client gets data from a FS it is getting a list of URLs and matching information which characterizes those URLs.

A typical multifile dataset is shown in Figure 1. The dataset contains about 25,000 files. A single satellite image is stored in each file. The files are arranged in a directory tree where each year's images are stored in a directory named for that year (1979, 1980, ...) and within that directory each month's images are stored in a directory name for that month (1, 2, ..., 12).

If a user or client already knows the pathname to a specific file, it can get that file using a DODS server. For example, the DODS URL to the first file in month 2 of year 1989 is <...>. However, if the user or client does not know the full pathname to the file they want, then they must understand how this particular dataset organizes its files and must build up the DODS URL themselves. With only one or two datasets this would not be hard to

[2] A constructor data-type similar to a relational table.

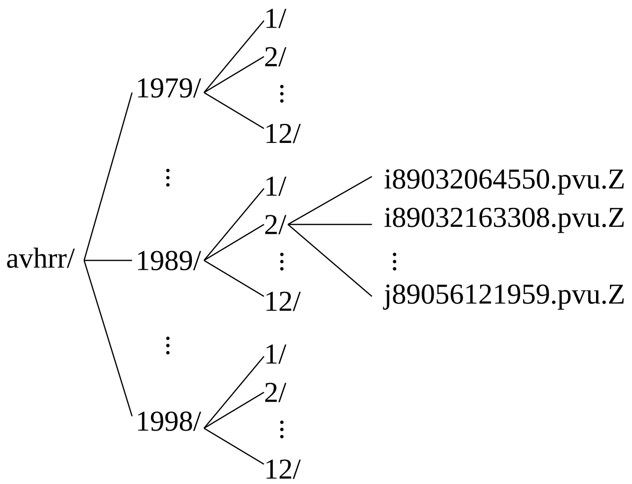


Figure 1: The Directory structure of a multifile datasets.

do, but with hundreds or more it is practically impossible to build clients which can carry with them the information to build URLs.

If a second (single file) dataset is created which lists the parameter values matched with URLs, then it is possible for a client to query first that dataset, get a URL (or a list of URLs) and then retrieve data. This is the basic File Server design. File Servers will be built using a DODS server to access a table of dates, times, etc. matched with URLs. The following sections describe this design and some refinements which address lingering problems in the basic design.

## 2.1 Primitive file servers

The simplest file server maps date and/or time onto URLs. For example, Figures 2 and 3 show a simple file server for the JPL Pathfinder dataset. It is *primitive* because no attempt is made to hide the internal organization of time. The client must understand how time is represented by the server and must be able to use that representation. In this example, date is expressed using year (four digits) and day number (three digits).[3] The constraint says "send the `day`, `hours` and `DODS_URL` variables for the sequence elements which have `year` == 1985 and 0 < `day` < 10". From the DDS in Figure 2 it is clear that `day`, `hours` and `DODS_URL` are all variables in the dataset.

[3]Note that Figure 3 shows output from the new ASCII feature of the DODS servers.

```
[dcz:] jpl=http://dcz/test/nph-jg/pathfinder
[dcz:] geturl -d $jpl %$
Dataset {
    Sequence {
        String year;
        Sequence {
            String day;
            String hours;
            String minutes;
            String seconds;
            String DODS_URL;
        } Level_1;
     } Level_0;
} pathfinder;
```

Figure 2: DDS of the JPL Pathfinder Primitive File Server.

Primitive File Servers are hard to use because without standardization, each server presents a special case. That is, the File Server architecture makes it possible to query a multifile dataset and determine which URLs map to certain parameter values (i.e., certain dates, times, ...) but if the users of File Servers must still master different variable names and different parameter organizations (e.g., dates as years, months and days on one server and year and year-day numbers on another) the gains are minimal since users must still know *a priori* which organization is used by each File Server; only a modest improvement over not having File Servers at all.

## 2.2 Hiding data organization for selections

To make file servers more uniform we can hide certain types of common queries behind functional interfaces.[4] The functional interfaces present a standard way to frame a query by defining a set number of, and representation for, parameters. The functions are responsible for translating this standard representation into a form that can be used by a dataset. In theory, each

[4] The primitive access method would still work; these functions serve to augment those methods, not replace them.

```
[dcz:] geturl "${jpl}.asc?day,hours,DODS_URL&year=1985&day>0&day<10" %$
day, hours, DODS_URL
004, 12, http://podaac.jpl.nasa.gov/.../1985004h09da-gdm.hdf
004, 00, http://podaac.jpl.nasa.gov/.../1985004h09dd-gdm.hdf
005, 12, http://podaac.jpl.nasa.gov/.../1985005h09da-gdm.hdf
005, 00, http://podaac.jpl.nasa.gov/.../1985005h09dd-gdm.hdf
006, 12, http://podaac.jpl.nasa.gov/.../1985006h09da-gdm.hdf
006, 00, http://podaac.jpl.nasa.gov/.../1985006h09dd-gdm.hdf
007, 12, http://podaac.jpl.nasa.gov/.../1985007h09da-gdm.hdf
007, 00, http://podaac.jpl.nasa.gov/.../1985007h09dd-gdm.hdf
008, 12, http://podaac.jpl.nasa.gov/.../1985008h09da-gdm.hdf
008, 00, http://podaac.jpl.nasa.gov/.../1985008h09dd-gdm.hdf
009, 12, http://podaac.jpl.nasa.gov/.../1985009h09da-gdm.hdf
009, 00, http://podaac.jpl.nasa.gov/.../1985009h09dd-gdm.hdf
```

Figure 3: Output from the JPL Pathfinder Primitive File Server.

interface must be custom-implemented for each dataset. In practice most datasets are similar and an interface may be implemented once and configured with a configuration file. The configuration file can be created much more simply than a complete implementation of that interface. For those datasets that encode values in ways that will not succumb to this scheme, the interface will have to be re-implemented.

There are three classes of functions for which standard interfaces have been, or will be, developed: date/time, geospatial and parameter. Since this is a working paper and since The most immediate need is for time-based queries, I'll describe only the time and date functions and how they might be used. Other functions will be described in the appendices of this paper.

For date, time and date+time queries three functions will be defined, one for each type of comparison. Each function will take either one or two arguments. If one argument is given then the function returns true in a CE for all dates (time or dates and times)[5] equal to the argument given. If two arguments are given then the function returns true for all dates between the two arguments.

[5]In the rest of this section I'll just say 'date' or 'dates'; the equivalent discussion holds for the time and date+time functions.

**date(*$arg$*):** date $==$ $arg$

**date(*$arg_1$*, *$arg_2$*):** $arg_1$ $<=$ date $<=$ $arg_2$

All the arguments will be passed as strings. The full set of date/time functions are described in Appendix B.

The constraint for the previous example (see Figures 2 and 3) could be written:

```
day,hours,DODS_URL&date("1985/1/1", "1985/1/10")
```

The client no longer needs to know how htis dataset represents dates to make queries. Furthermore, the syntax and semantics of `date(...)` are the same for all datasets that support this function. Once a client is programmed to use this function, it can make date queries for all such datasets.

## 2.3 Abstracting return variables

Note that the `date` function described above does not affect how time is encoded in the returned sequence. Thus it is only half a solution. While the client does not need to understand how dates are encoded in the dataset to *make* a query, it *does* need to know how they are encoded when interpreting the results of that query!

One way to return values using a standard representation is to synthesize new variables and add them to the dataset. These new variables would compute their values based on the values of other variables in the dataset. Since it is possible to synthesize the new variables using functions, we can control the ways these variables represent various types of information.

However, once the *projection* part of a constraint expression has been parsed the collection of variables in the dataset cannot be changed easily. Unfortunately, function were not previously supported in the projection part of a CE. To work around this limitation, functions are now supported in the projection part of the CE. The functions can take any action the function implementor chooses. Projection Functions are written to take three argument, an integer argument count, a vector of pointers to BaseType instances and the DDS for the dataset. This is the same type signature as functions for the selection part of the CE. In addition, projection functions are registered with the DDS using the same method call as function for the selection part of the CE.

Once the projection function is written and registered with the DDS, a server can process a constraint such as:

```
DODS_Date(),DODS_URL&date("1985/1/1", "1985/1/5")
```

Where the projection function `DODS_Date()` causes the CE parser to synthesize a new variable and add it to the dataset. The variable is added to the DDS while the projection part of the CE is being parsed; the parser actually evaluates the function which in turn synthesizes the variable and adds it to the DDS.[6] At that time an additional selection clause is also added to the expression which contains a function call similar to the `date()` function described in section 2.2. This function, which users never see, computes a value for the new variable. Thus, the single function seen in the constraint expression is a façade. The real work is done by a function in the selection part of the expression, the function seen by the user in the projection part is there to establish and initialize the new variables the unseen selection function. Figure 4 shows output from the above query. Note that the return values for the date use a standard string representation although the dataset uses two integer values to represent the same information.

The one potential complication in this design is that the `DODS_Date()` *projection function* must know where in the DDS to add the new variable. In many cases this might be simple since, by looking at the other variables in the projections you could divine what the user wanted. However, doing this in general might be very hard. Users wanting the new variable added at a particular point the DDS' hierarchy will need to supply the name of the new variable's container. With no name given `DODS_Date()` should assume that the user wants the new variable to be at the top level of the DDS.

# 3 Adding Standard Information About Variables and Datasets

Standard namespaces were initially introduced into DODS to make graphical user interfaces simpler to build and scalable. However, it seems certain that such a service will be used by many different types of clients, not just user

[6] A new method synthesized_p() has been added to the class BaseType. The projection function must be sure to set send_p() and synthesized_p() for a synthesized variable.

```
DODS_Time, DODS_URL
1985/1/4, http://podaac.../1985004h09da-gdm.hdf
1985/1/4, http://podaac.../1985004h09dd-gdm.hdf
1985/1/5, http://podaac.../1985005h09da-gdm.hdf
1985/1/5, http://podaac.../1985005h09dd-gdm.hdf
```

Figure 4: Output of the JPL Pathfinder File Server Using a Standard Interface.

interfaces. Thus, the interface must present a range of information in a machine readable fashion.

It is possible to use an 'ancillary DAS' (which is a text file) to augment the DAS object for any given dataset. See Figure 5. Extra information may be added about each variable using the ancillary DAS. For each variable in a blessed dataset (one where the ancillary DAS object has been defined) there should be attribute information that gives the standard name for that variable.

> A useful add-on to the current stable of server features would provide a way for database users to look at the DAS information. Think of the standards we're talking about here as a schema and each DAS as a record in a highly distributed database. Towards that end, to what degree can the Sequoia 2K schema be CO-opted by DODS?

Clients will use this information by requesting it. It will be up to the client to know to look for standard names, substitute the real name of the variable for the standard name, etc. In the future we can make the system more automatic. See the DODS User's Guide for more information about DAS objects, attributes and ancillary attributes.

Figures 6 and 7 show information for one of the image files in the DSP image archive at GSO. The DAS shown (Figure 7) is the *ancillary* DAS file. In other words that is the information were adding to the DAS so that when a client requests the DAS of one of the images it will see the information shown there *in addition to* the information that the DSP server extracts from the file.

A more complicated situation is presented by Figures 8 and 9. These figures show the DDS and an ancillary DAS for the JPL Pathfinder file server

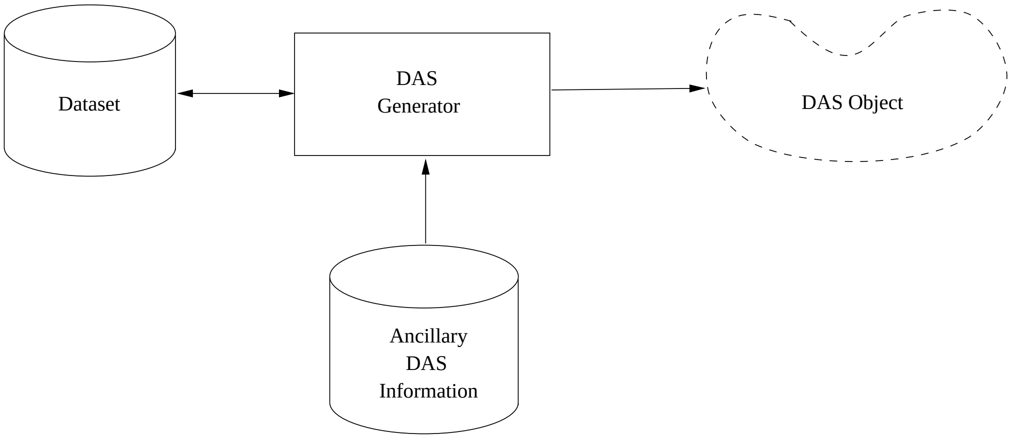


Figure 5: Information from the dataset along with and ancillary DAS object are combined into a single DAS object.

```
[dcz:/usr/local/DODS/src/dap-2.17] geturl -d $dsp
Dataset {
    Grid {
     ARRAY:
        Byte dsp_band_1[lat = 1024][lon = 1024];
     MAPS:
        Float64 lat[lat = 1024];
        Float64 lon[lon = 1024];
    } dsp_band_1;
    Float64 lat[lat = 1024];
    Float64 lon[lon = 1024];
} h97197120242;
```

Figure 6: DDS of a DSP Image file.

```
[dcz:/usr/local/DODS/src/dap-2.17] geturl -a $dsp
Attributes {
    DODS_GLOBAL {
        String DODS_Interfaces "";   # No interfaces supported
        Int32 DODS_Standards 2;      # Class of this dataset/server
    }
    lat {
        String DODS_Name "Latitude"; # New information
        Alias DODS_Units units;      # Alias to existing attribute
        DODS_Parameter "...";        # Whatever the GCMD param. name is.
        DODS_Extent {
            Float64 DODS_Min_Max 34.7, 62.5;
        }
    }
    .
    .
    .
    dsp_band_1 {
        String DODS_Name "Sea Surface Temperature";
        String DODS_Units "counts";
        DODS_Parameter "...";        # Whatever the GCMD param. name is.
        DODS_Extent {
            Float64 DODS_Min_Max 0, 255;
            # Vertices of containing polygon (a box in this case)
            Float64 DODS_Containment 34.7, -70.0, 34.7, -50.0,
                                     62.5, -70.0, 62.5, -50.0;
        }
        DODS_Mapping_0 {
            String DODS_Unit "degC";
            Int32 DODS_Null 0, 1, 2, 3;
            Float64 DODS_Coefs 0.125, 2.5;
        }
    }
}
```

Figure 7: Possible entries into an ancillary DAS for the DSP Image in Figure 6. Parts of the DAS have been removed to make it more readable.

```
[dcz:] geturl -d $jpl
Dataset {
    Sequence {
        String year;
        Sequence {
            String day;
            String hours;
            String minutes;
            String seconds;
            String DODS_URL;
        } Level_1;
    } Level_0;
} pathfinder;
```

Figure 8: DDS for the JPL Pathfinder File server

discussed in Section 2. The server supports the date and time interfaces which includes (unless we decide to change them) two functions which return a standardized date and/or time value. Effectively those interfaces synthesize a variable. How do we represent that synthesized variable in the ancillary DAS? One way (shown) is to add a new attribute container for the variable and include various 'standard' names for it (`DODS_Min`, `DODS_Max`, ...).

## 3.1 Discovering a server's class

There is a need to support multiple classes[7] of datasets within DODS. One class of server provides the lowest level of services. They return, via the DAS, DDS and Data objects, exactly what was encoded with the data. In the case of some APIs (e.g., NetCDF) that *may* be a lot information while in the case of others (e.g., Matlab) it will be next to nothing. At the other end of the spectrum will be servers which had extensive amounts of additional information about the dataset. We should not plan on accessing this information using a functional interface because, by default, DODS servers have *no* functions. They are added on a per server basis. However, the DAS object was

[7]Note to be confused with classes in OOP; really, 'levels' would be a better word, but that is loaded with obscure meanings in the science data systems world.

```
[dcz:] geturl -a $jpl
Attributes {
    DODS_Global {
        # This server supports the date and time interfaces.
        String DODS_Interfaces "date", "time";
    }
    DODS_Time {
    .
    .
    .
    }
    .
    .
    .
    DODS_URL {
    }
}
```

Figure 9: Ancillary DAS for the JPL Pathfinder File Server. How do we include information about date/time?

specifically designed to hold additional information about the dataset and its variables *and* to be extensible (without changing the underlying data).

Clients wanting information about a dataset's/server's class can examine the attribute container `DODS_GLOBAL` for the attribute `DODS_Interfaces`. If present, it will give the class of the server/dataset. If absent, the client should assume the server/dataset is class 0 (the lowest level of a dataset/server). See Appendix A.

### 3.2 Discovering a server's interfaces

To provide a way for clients to learn about various interfaces available from a given server, we can describe those interfaces in an attribute called `DODS_Interfaces`. Each interface will have a standard set of functions which will be listed in an appendix of this paper. For example, one interface previously described is the date/time interface. The DODS_interfaces attribute would only need to name the interface, not list all of its functions, etc. It will be up to the clients to know which interfaces do what. See Appendix A.

## 4 Using Scientific Units for Queries

Users should be able to ask for data using Earth-science units. In the current incarnation of DODS, if a user wants to read information from an array or grid they must either read the entire data object or use array indices to constrain the object. This is often hard to do since the mapping between indices and Earth units can be complex. DODS needs to provide a simple way for users to frame queries using the units in which the variables are registered.

## 5 Going Further

Here are features which should be added to the base-line catalog services described in this paper:

- Extend the catalog server so that it can use attributes of subordinate datasets as variables in its own dataset. For example, suppose that a catalog for the AVHRR dataset lists Date, Time and URL. Each file in the dataset is referenced by one URL and the Date and Time columns of the catalog provide a way to choose a URL based on date and time. Now suppose that in each file (which is also a dataset) there is a variable called 'lon' with DODS:Minimum and DODS:Maximum attributes. The 'lon' variable from the files could be promoted to the catalog and, in the catalog, it would assume the values given by the Minimum and Maximum attributes.

In order to promote an attribute up from a dataset to a catalog for that dataset, the attribute must be marked as *promotable*. For a multi-file dataset, an attribute is promotable if it is preset and has the same value in all files. All attributes in single file datasets are promotable.

Attributes promoted would be the DODS standard ones. The DODS:Name attribute of a dataset would be used to name the variable in the catalog.

- Add an additional service that can hold an object, or reference to an object, without returning it. This service would allow clients to ask questions about the referenced object *before* retrieving it.

- Extend the catalog so that it can dereference URLs automatically and return complex objects.

# A Standard Attributes

These attributes must/should be present for a class 1 or 2 dataset. Next to each attribute name is the lowest class of server that is required to have that standard name in the DAS object. A class 2 server must have both class 1 and 2 attributes, while a class 1 server needed only the class 1 attributes. A server that supplies none of these attributes is a class 0 server.

**DODS_GLOBAL: 1** This is an attribute container that holds other standard attributes which apply to the entire dataset and server.

- **Interfaces: 1** This attribute contains a vector of strings, each element of which names a standard interface supported by this server.
- **Standards: 1** An integer describes the class of this server and dataset. 0 == some class 1 attributes missing, 1 == no class 1 attributes missing, some or all class 2 attributes missing, 3 ...[8]

**DODS_Time: 1** This attribute container holds information used by the DODS_Time interface. These attributes provide a simple way to configure the DODS_Time interface so that time data may be read from datasets without data providers writing C++ code. The information contained here will be expanded to cover a broader range of storage formats as dictated by demand.

- **hours_variable: 1** This holds the name of the variable in the dataset which contains hours information. If the variable is part of a Sequence, then it is assumed that that for each instance of the Sequence, the named variable contains hours information for that instance. The same applies to the following two attributes.
- **minutes_variable: 1** This holds the name of the variable which contains minutes information.
- **seconds_variable: 1** This holds the name of the variable which contains seconds information.
- **gmt_time: 1** This holds either the string 'true' or 'false' depending on whether the dataset's times are GMT/UTC times or not.

[8] Yes, there is a bit of a catch 22 here. If this attribute is missing then the client is on its own.

**DODS_Date: 1** This attribute container holds information used by the DODS_Date interface that used in a way analogous to that held by the DODS_Time attribute container.

**year_variable: 1** The name of the variable which contains the year information for a date. If this variable is part of a sequence, then it is assumed that it contains the year information for that sequence element.

**year_base: 1** If present, this Int32 attributes contains the base year for dates. For example, if a dataset stores the year 1998 as 98, year_base would be 1900. If this is not present DODS will assume that a year 98 refers to 98 A.D.[9]

**month_variable: 1** The name of the variable which contains information about the month.

**day_variable: 1** The name of the variables which contains information about the month. If this attribute is used the *year_day_variable* attribute should not be present. The results of defining both *day_variable* and *year_day_variable* are undefined.

**year_day_variable: 1** The name of the variable which contains year-day date information. For example, some datasets store year information in one variable and a day offset within the year in another. See *day_variable* for warnings about defining both that attribute and this one.

The DODS_Date_Time interface provides a way to compute with combined date and time values. The interface uses the DODS_Date and DODS_Time attributes. That is, the DODS_Date_Time_Factory class will read from the DODS_Date and DODS_Time attributes. See Appendix B for more information about date and times in constraint expressions.

**Variable name: 0** For each variable in a dataset, there is one attribute container with that variable's name. This is not part of the standard—there is always an attribute container for each variable name. The

[9]This will almost certainly be wrong, but it is also better in the long run since 'base 1900' dates are a bad idea.

attributes listed below are *per variable* attributes. They appear in each variable's attribute container.

**DODS_Name: 1** The name of this variable. We will need to build a dictionary of blessed names. Examples include *Latitude*, ...

**DODS_Parameter: 2** The GCMD classification of this variable.

**DODS_Unit: 2** In what units are the variable's values.

**DODS_Extent: 2** This container attribute

- **DODS_Min_Max: 2** A two element vector which contains the minimum and maximum values for this variable. Present only for scalars.
- **DODS_Enum: 2** An enumeration of possible values for this variable. Present only for scalars. DODS_Min/DODS_Max and DODS_Enum are mutually exclusive attributes
- **DODS_Containment: 2** A vector of vertices of a polygon enclosing the data. The vector stores latitude/longitude information in the vector as $lat_0, lon_0, \cdots, lat_n, lon_n$.[10]

**DODS_Mapping_*name*: 3** Provides information on mapping the values of this variable from DODS_Unit to some other unit system. There can be more that more that one set of mapping attributes if there are more than one set of values.

- **DODS_Coefs: 3** A vector containing $a_0, a_1, \cdots, a_n$ where those values can be used in $y = a_n x^n + \cdots + a_1 x + a_0$ to map values of the variable $x$ into new values $y$.
- **DODS_Unit: 3** The Units of $y$ in the above are given by this attribute.
- **DODS_Null: 3** If any values of the variable are used to denote null or missing values they should be given here. If more than one value is used to indicate missing data, this attribute should be a vector of those values.

[10] Should this be made more general so as to include other types of containment?

# B Date/Time

The date/time interface provides a standard way to query a dataset about time-dependent information. The interface provides a set of functions which can be used to select data using date and/or time and to retrieve date and/or time information in a standard way. In addition to the functions, the interface defines a standard representation for dates and times. These representations are used by the functions and the return values. The intent of this interface is to present a standardized *view* of a dataset's contents, not to standardize the data values themselves. Thus the functions must be customized to a particular dataset; they must know how to read date/time information using the local conventions of the dataset as well as transforming the standard representations into the local ones for comparison.

## B.1 Representation of dates and times

Dates will be represented as a string formated in one of two ways: either using a year, month, day format or using a year and day-number format. See Table 1. Time will be represented as a string using hours, minutes and seconds. See Table 2. Combined date and time values will use a colon to separate the two parts of the string (e.g., $< date >:< time >$).

*How should we handle smaller times?*

Examples dates, times and date-time strings:

- 1 January 1998: `1998/1/1`
- 4 February 982: `982/2/4`
- 4:57pm: `16:57:00`
- 1:23:45am: `1:23:45`
- 1 January 1998 at 4:57pm: `1998/1/1:16:57:00`

## B.2 Functions for projection and selection

Three functions are defined for selection by date and/or time. Arguments enclosed in [] are optional.

Table 1: Grammar for dates

| | |
|---|---|
| $< date >$ | $< year > / < month > / < day >$ $\mid < year > / < day - number >$ |
| $< year >$ | $0 \cdots 9999$ |
| $< month >$ | $1 \cdots 12$ |
| $< day >$ | $1 \cdots 31$ |
| $< day - number >$ | $1 \cdots 366$ |

Table 2: Grammar for times

| | |
|---|---|
| $< time >$ | $< hours >:< minutes >:< seconds >$ |
| $< hours >$ | $0 \cdots 23$ |
| $< minutes >$ | $0 \cdots 59$ |
| $< seconds >$ | $0 \cdots 59$ |

**date**($arg_0$, [$arg_1$]) Return true for all values whose dates equal or fall between $arg_0$ or $arg_0$ and $arg_1$. In the later case the range is inclusive.

**time**($arg_0$, [$arg_1$]) Return true for all values whose times equal or fall between $arg_0$ or $arg_0$ and $arg_1$. In the later case the range is inclusive.

**date_time**($arg_0$, [$arg_1$]) Return true for all values whose date-times equal or fall between $arg_0$ or $arg_0$ and $arg_1$. In the later case the range is inclusive.

Matching these selection functions are three projection functions. These are used in the projection part of a constraint expression to ask for date/time information as a variable in the values returned as a result of the query.

*There are two schemes described earlier in this paper—this is one of them.*

**DODS_Date**([*container*]) Add the variable `DODS_Date` to the set of variables returned as a result of this query. If *container* is given, add `DODS_Date` not at the top-level of the DDS but to the named container.

**DODS_Time**([*container*]) Add the variable `DODS_Time` to the set of variables returned as a result of this query. If *container* is given, add `DODS_Time` not at the top-level of the DDS but to the named container.

**DODS_Date_time(**[*container*]**)** Add the variable `DODS_Date_Time` to the set of variables returned as a result of this query. If *container* is given, add `DODS_Date_Time` not at the top-level of the DDS but to the named container.

## B.3 Customizing the Date/Time interfaces for a particular dataset

There are three ways to customize the Date/Time interfaces for a particular dataset. Different customization options are provided since there is a wide variety of storage forms and formats. The simplest option can be done without any software development at all (see Section B.3.1) while others require that C++ software be written (See Sections B.3.2 and B.3.3) .

### B.3.1 Using attributes to customize the Date/Time interfaces

If a dataset stores year, month, day or year and year-day information in integer variables, then the Date interface may be set up using attributes. See Section A for information. Similarly, the Time and Date_Time interfaces may also be configured using attributes for datasets which have the required information in Integer variables.

### B.3.2 Writing new factory classes

The attributes described in Section A are read by *factory*[11] classes which configure themselves to read the appropriate type object from the dataset (e.g., See the source files `DODS_Date_Factory.h` and `DODS_Date_Factory.cc`). For each class which implements one of the standard interfaces (DODS_Time, DODS_Date and DODS_Date_Time) there is a matching factory class (DODS_Date_Factory, ...).

For any dataset which does not store information in a way anticipated by the current set of factory classes, users may modify or extend those classes or replace one or more with new factory classes.

[11] I'm using the term as it appears in the Java programming language documentation.

### B.3.3 Writing new interfaces

For some datasets the concept of a factory object which reads values may not be appropriate. In this case, a new interface class may be written. In this case the new class must be meshed with the constraint expression evaluator. To do this three new functions (not member functions) must be written. The source file `ce_functions.cc` contains examples of these three functions.

In order to enable the CE evaluator to compare values using a standard representation, you must register a comparison function. in the source file `ce_function.cc`, look at the definition of `bool func_date(...)` for an example. To enable the return of values using your new standard interface class, you must define two new functions; one to run during the evaluation of the projection part of the CE and one that is run during the evaluation of the selection part of the CE. Look at the definitions of `void proj_dods_date(...)` and `bool sel_dods_date(...)` in `ce_functions.cc`, respectively, for examples for these. Note that the `bool func_date(...)` and `void proj_dods_date(...)` functions must be registered with the DDS using `add_function(...)`. To see an example of this, look at `ff_dods.cc`.