DODS Server Installation Guide
Version 1.12

Tom Sgouros

February 25, 2003

Contents

List of Figures

List of Tables

1 The DODS Server

The Distributed Oceanographic Data System (DODS) provides a way to access data over the internet, from programs that weren't originally designed for that purpose, as well as some that were. A *DODS server* is a program that sends data in a standard transmission format to a client that has requested it. A *DODS client* is a program, running on a networked computer somewhere in the world, that requests and receives data. A client can be a specialized DODS client, or a standard web browser.

Though originally developed to deal with oceanographic data, DODS has found wide use outside that community, and is now used for several different kinds of science data.

Nothing limits the use of DODS to science data; its framework will support many different data types. But DODS has facilities for accommodating large arrays, relational tables, irregular grids, and many other otherwise anomalous data types. DODS servers can be adapted to serve data from any kind of storage format, and versions that support several popular data storage formats are readily available.

A DODS server is just an ordinary WWW server (`httpd`) equipped with a set of CGI programs that enable it to respond to requests for data from DODS client programs. Web servers and CGI programs are standard parts of the Web, and the details of their operation and installation are beyond the scope of this guide. (That is, there are too many different varieties of web servers out there for us to help you install them.) Once you have one installed, this guide will explain how to use it to serve DODS data.

A DODS server is just a http server with special CGI programs.

Entire books are written about the operation of the Internet and the WWW, and about client/server systems. This is not one of them. To understand the DODS architecture, you need only understand the following:

- A Web server is a process that runs on a computer (the host machine) connected to the Internet. When it receives a URL from some Web client,

such as a user somewhere operating Netscape (or a specialized DODS client) it packages and returns the data specified by the URL to that client. The data can be text, as in a web page, but it may also be images, sounds, a program to be executed on the client machine, or some other data.

- A properly specified URL can cause a Web server to invoke a *CGI program* on its host machine, accepting as input a part of the URL, or some other data, and returning the output of that program to the client that sent the URL in the first place. The CGI is executed on the server.

The CGI programs you need depend on the data you want to serve.

The CGI programs you need depend on the storage format of the data you intend to serve. The DODS project supports a variety of storage formats, including NetCDF, HDF, Matlab, and DSP. DODS can also read data using the FreeForm ND and JGOFS libraries, which can be configured to read files with nearly any data format.

1.1 Server Architecture

A DODS server consists of a set of programs, and a CGI *dispatch script* used to decide which program can handle whatever request is at hand.

You need to know the shape of the data before you request the data.

A user can make three different sorts of requests to the server. The first request is for the "shape" of the data, and consists of the *data descriptor structure* (DDS). The second request is for the *data attribute structure* (DAS) of the data types described in the DDS. (*The DODS Quick Start Guide* and *The DODS User Guide* contain more information about these structures.) Both of these requests return plain text data, readable in a web browser like Netscape Navigator. You can see a typical DDS in figure 1.1.

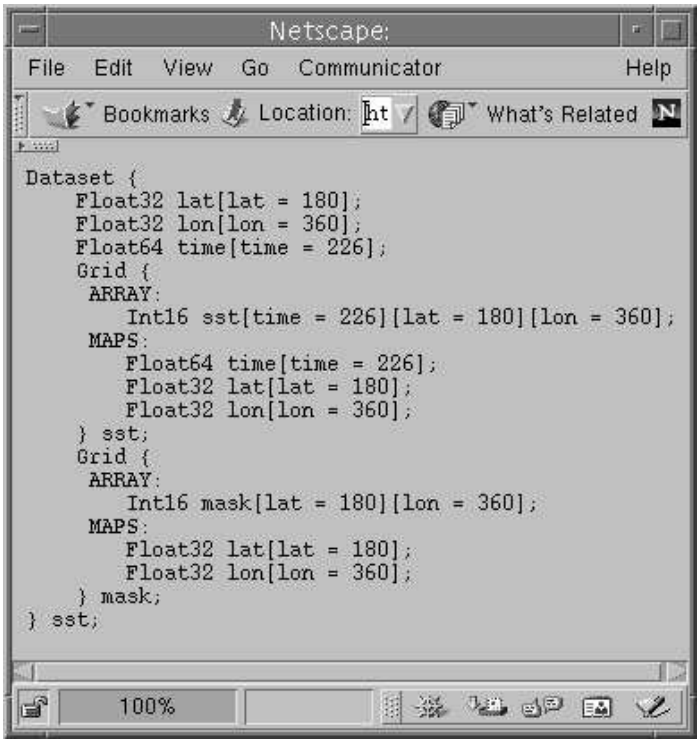

Figure 1.1: A DODS DDS (`sst.mnmean.nc.dds`)

There's a DAS from the same dataset in figure 1.2

Other metadata is helpful, too.

After getting the *metadata* (defined, for DODS purposes, as the contents of the DAS and DDS), the DODS client can request actual data. This is binary data, and is often too large to view easily in a web browser. (See *The DODS Quick Start Guide* for strategies to use to examine DODS data from a standard web browser.)

Depending on the data format in use, the DAS and DDS are either generated from the data served, or from ancillary information text files you have to supply (or both). The data in these structures may be cached by the client system.

The DODS Quick Start Guide shows how to look at all the DODS services.

In addition to the three basic message types (DAS, DDS, and data), a DODS server can also provide information about the server operation and about the data, can return data in ASCII comma-separated tables, and can provide a query form allowing users to craft subsampling requests to the server. Some of these services are provided by other service programs that must be installed with the dispatch script and its companions.

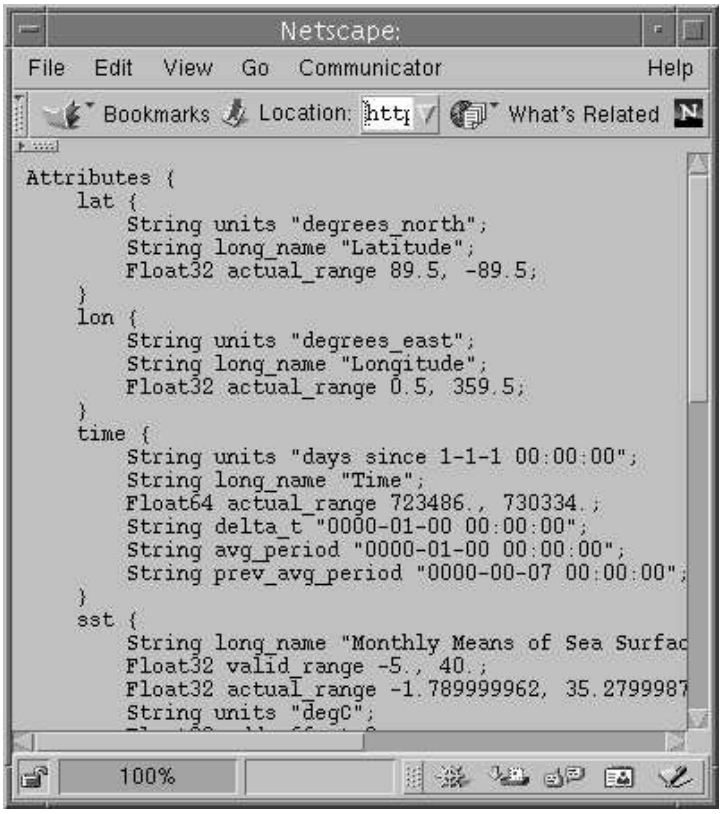

Figure 1.2: A DODS DAS (`sst.mnmean.nc.das`)

1.2 More Explanation of How It Works

To understand the operation of the DODS server, it is useful to follow the actions taken to reply to a data request. The diagrams in figure 1.3 and figure 1.4 lay out the relationship between the various entities. Consider a DODS URL such as the following:

A description of the different URL parts.

```
http://dods.org/cgi-bin/nph-dods/data/fnoc1.nc
```

The URL as written refers to the entire data file, but any particular request must be slightly more specific. The precision is supplied by appending a suffix to the data URL. Do you want binary data (`.dods`), ASCII data (`.asc`), the DDS (`.dds`), the DAS (`.das`), usage information (`.info`), or a query form (`.html`)? To get a DDS, for example, you would use this URL:

```
http://dods.org/cgi-bin/nph-dods/data/fnoc1.nc.dds
```

A DODS client may silently add the appropriate suffix to the URL, but if you're using a standard WWW client, such as Netscape, you have to add the suffix yourself.

Once the proper suffix has been appended to the URL, the URL is sent out into the world. Through the magic of IP addressing, it makes its way to the web server (`httpd`) running on the platform, `dods.org`. Figure 1.3 shows these first steps. The client makes an internet connection to the `dods.org` machine, and the `httpd` daemon executes the dispatch script (`cgi-bin/nph-dods`) and forwards it the remaining parts of the URL it had received.[1] (In this case, that would be `data/fnoc1.nc.dods`.) DODS requests are "GET" requests, not "POST" requests, so all the information forwarded is in the URL.[2]

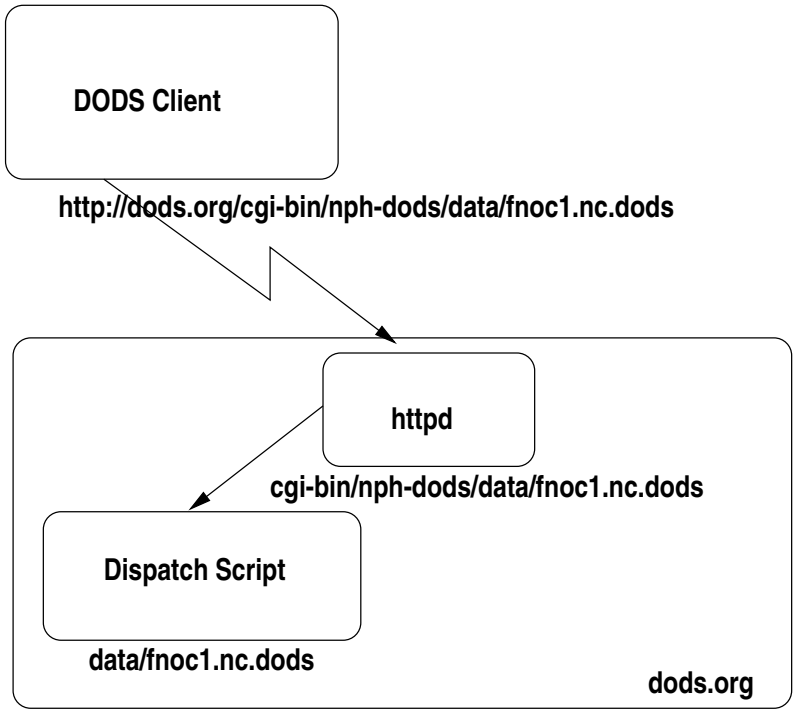

Figure 1.3: The Architecture of a DODS Data Server, part I.

Figure 1.4 illustrates what happens next. Sitting in the CGI directory (here called `cgi-bin`) with the dispatch script are several *service programs* , also called *services* or *helper programs* . The dispatch script (`nph-dods`) analyzes the suffix on the URL to figure out what kind of request this is, and executes the corresponding service program.

[1] The actual directory name, or whether the CGI programs are kept in a particular directory (or named with a particular convention) is another detail of the specific web server and configuration used. The web server might refer to the directory as the `ScriptAlias` directory, as it does with Apache.

[2] When you fill out some HTML form, you are usually sending data in a "POST" request. When you type a URL into your web browser, this is a "GET" request. HTTP servers can respond to both kinds of request.

NOTE: As of DODS release 3.2, the dispatch script is named `nph-dods`. Earlier releases of DODS used diffferent dispatch scripts, depending on the storage format of the data. For earlier releases, there would be a `nph-nc` to handle netCDF data, `nph-jg` to handle JGOFS data, and so on.

In the case illustrated, the `.dods` suffix indicates that this is a request for binary data. Therefore, the dispatch script executes the `nc_dods` service, and forwards to it the rest of the URL, which includes a data object name (which may be a file or not, depending on the API), and possibly a *constraint expression* .[3] It's up to the service program to find the data, read it, read and parse the constraint expression (if any), and output the data message. If the service requires any ancillary data, it may also read an ancillary data file or two, as necessary.

The standard output of the service program is redirected to the output of the `httpd`, so the client will receive the program output as the reply to its request.

For APIs that are designed to read data in files, such as netCDF, the CGI program will be executed with the working directory (also called the default directory) specified by the `httpd` configuration. However, the DODS software will look for its data relative to the document root tree. On the `dods.org` server, for example, all CGI programs are executed native to the directory `/usr/local/spool/http/cgi-bin`, but the document root directory is `/usr/local/spool/http/htdocs`. The last section of the URL, then, specifies the file `fnoc1.nc` in the directory:

Data files are specified relative to the document root directory.

```
/usr/local/spool/http/htdocs/data.
```

Some existing data APIs, such as JGOFS, are not designed with file access as their fundamental paradigm. The JGOFS system, for example, uses an arrangement of "dictionaries" that define the location and method of access for specified data "objects." A URL addressing a JGOFS object may appear to represent a file, like the netCDF URL above.

```
http://dods.org/cgi-bin/nph-dods/station43
```

However, the identifier (`station43`) after the CGI program name (`nph-dods`) represents, not a file, but an entry in the JGOFS data dictionary. The entry will, in turn, identify a file or a database index entry (possibly on yet another system) and a method to access the data indicated. These are JGOFS server-specific installation issues covered in the installation documentation for that server.

Note that the name and location of the CGI directory (`cgi-bin`), as well as the name and location of the working directory used by the CGI programs, are local configuration details of the particular web server in use. The location of the

[3] This is not shown in this illustration, but it would follow a question mark in the URL, like this: `http://dods.org/cgi-bin/nph-dods/temp.nc.asc?temp[0:180][0:45]`. For more information about constraint expressions, see *The DODS Quick Start Guide* or *The DODS User Guide*.

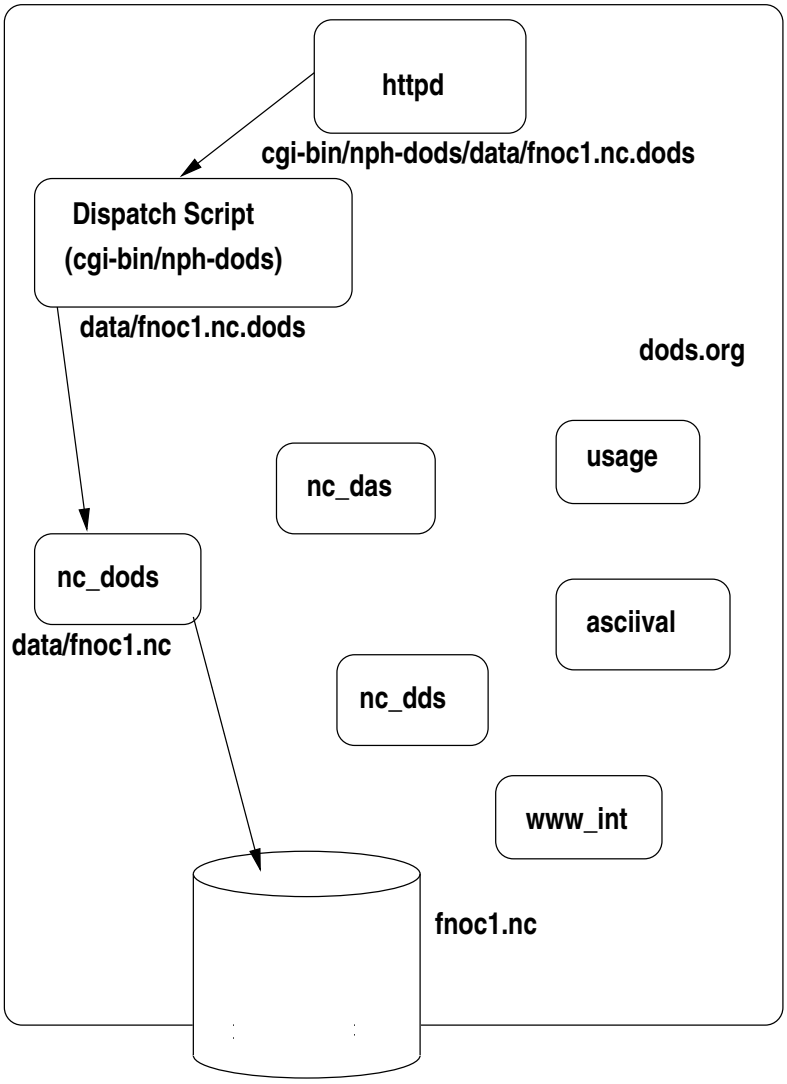

Figure 1.4: The Architecture of a DODS Data Server, part II.

JGOFS data dictionary is a configuration issue of the JGOFS installation. That is to say these details will probably be different on different machines.

1.3 Service Programs

The service programs do the real work of a DODS server.

When a server gets a DODS request, it executes the dispatch script, which then figures out which service program should be invoked. The output from that program is what gets returned to the client.

Table 1.1 contains a list of the service programs required for each of the services for the DODS netCDF server. The dispatch script is called `nph-dods`. (Though see note on page 10.) For another DODS server, the names of some of the helper programs would have a different root than `nc`. (For example, `ff` identifies the FreeForm server, `jg` for JGOFS, and so on.)

Table 1.1: DODS Services, with their suffixes and helper programs. The *pp* is replaced by the prefix corresponding to the appropriate data access format, e.g. `nc` for netCDF, `jg` for JGOFS, and so on.

Service	Suffix	Helper Program
Data Attribute	`.das`	*pp*`_das`
Data Descriptor	`.dds`	*pp*`_dds`
DODS Data	`.dods`	*pp*`_dods`
ASCII Data	`.asc` or `.ascii`	`asciival`
Information	`.info`	`usage`, see Chapter 7.
WWW Interface	`.html`	`www_int`
Version	`.ver`	None
Compression	None	`deflate`
Help	Anything else	None

The service program invoked depends on the URL suffix.

The service programs are started by the dispatch script depending on the extension given with the URL. If the URL ends with '.das' and the file name in the URL ends in '.nc', then the DAS service program (`nc_das`) is started. Similarly, the extension '.dds' will cause the DDS service to run and so on.

On the client side, when using a DODS client, the user may never see the '.das,' '.dds,' or '.dods' URL extensions. Nor will the user necessarily be aware that each data URL given to the DODS client may produce three different requests for

information. These manipulations happen within the DODS client software, and the user need never be aware of them.

NOTE: This is only true *when using a DODS client*. Programs that don't use the DODS client libraries can still be clients of a DODS server. You can use Netscape to contact a DODS server and get data, in which case you have unmediated access to the server and need to include the service program URL extensions.

1.4 Choosing a Server

There are a variety of DODS servers available, to handle different data storage formats. DODS servers exist to serve data stored in the netCDF, HDF and HDF-EOS, Matlab, and DSP storage formats. If you have data stored with one of these formats, the choice is quite simple: choose the one that works with your data.

There is also a JDBC DODS server, written in Java, for serving data stored in relational databases. See Chapter 4 for more information about installing that software. See the DODS home page Java section for more information about the DODS Java software.

If your data is not already stored in one of the supported formats, don't despair. Some standard API formats include tools for translating data into that format. For example, netCDF includes an application called `ncgen` you can use to translate array data into standard netCDF files by writing a data description in the netCDF CDL (Common Data Language). See the netCDF documentation for more information about this.[4]

There are two DODS servers that can accommodate various file formats.

If your data is not in a supported format and you don't want to translate it into one of those formats, there is still a way to serve your data. There are two other DODS servers available that can be used to serve data that are not already in one of these formats. These are the FreeForm and JGOFS servers. It may be that one of these servers can be easily adapted to your uses. The FreeForm server is somewhat easier to set up, and the JGOFS server is more flexible. A key difference is that the JGOFS server can handle data contained in several different data files. (The FreeForm server can, as well, but, being slightly less flexible, it may require the files to be rearranged or renamed.)

There's a brief comparison of the two in table 1.2

[4] A user has contributed examples for this. Go to http://seawater.tamu.edu/noppdodsgom and click on "Resources."

Server	Advantages	Disadvantages
FreeForm	Simple to set up. Serving data in a new format requires only creating a text file describing that format. Serves data in Arrays or Sequences.	Not quite as flexible as its name implies. If the format in question is too complex or too variable, the FreeForm API cannot handle it. Sequences can be served, but only flat ones. (That is, Sequences that contain other Sequences will not work.) Generally, data must line up in columns.
JGOFS	Extremely flexible. Uses specialized access methods to read data, and these methods can be extensively customized. Optimized for Sequence data (relational tables), including hierarchical Sequences (Sequences that contain other Sequences).	Writing a data access method can be complex, since it involves writing a program in C or C++. Does not support Array data types.

Table 1.2: Advantages and Disadvantages of the Two Flexible DODS Servers

It is possible that none of these options is the right one for you, in which case you can use the DODS DAP library to craft a server of your very own. The library is available in both C++ and Java. If you choose this route, contact the DODS team; we may be able to direct you to someone who has already done something like it. The *The DODS Toolkit Programmer's Guide* contains useful information about the DAP library, including instructions on how to construct servers and clients.

2
Installing a DODS Server

Most of the task of installing a DODS server consists of getting the required Web server installed and running. The variety of available Web servers make this task beyond the scope of this guide. Proceed with the following steps only after the Web server itself works. Look at Chapter 6 for hints on how to tell whether the server is working.

First, get the web server working. Then install DODS.

If you want to install the DODS Relational Database Server (DRDS), to serve data stored in a relational DBMS, like Oracle or SQL Server, see Chapter 4.

2.1 Step by Step

Here are the steps to installing the DODS CGI programs (these are the components of the server) and data to be served.

❶ **Install the web server.**

❷ **Download and unpack the DODS software.**

❸ **Install the DODS software.**

❹ **Install your data.**

In a little more detail, here are those same steps:

❶ Install the web server to be used. This is not a DODS program, and will have its own documentation. If the server is already installed, figure out how to run a CGI program with that server.

Testing: If your web server is running, you should be able to request a web page from it. From a standard web browser, like Netscape, Try sending a simple request containing only the machine name: `http://machine`, where *machine* should be replaced by the name of the computer you're doing the installation on. When the simple page works, try executing a CGI program. (The simplest CGI Perl program is on 46.) See Chapter 6 for more ways to check if the web server is working.

❷ Download and unpack the DODS software. It is usually easiest to use the pre-compiled binary distributions. Look at the DODS Home page to get the software. See Appendix A on page 57 for more information. *Don't forget the DODS Server Tools.* These are needed for all the DODS servers.

Testing: The software should all be unpacked under a single directory listing that we call the `$DODS_ROOT` directory.

❸ Use the install script, `$(DODS_ROOT/etc/installServer`, to copy the CGI programs and Perl modules in the `etc` directory in the DODS source tree (`$(DODS_ROOT)/etc`) to one of the directories where the Web server expects to find its CGI programs. (You can do this by hand too, if you like.) The service programs used by the CGI are generally kept in the same directory as the CGI itself, although this can be changed by modifying the DODS CGI dispatch script. (The dispatch script is the one called `nph-dods` or that looks like `nph-`*api*.) Make sure the CGI programs are executable by the web server.

Testing: When the DODS server is working, you should get a response to a version request, where you query the DODS software for its version

(release) number. To do this, enter a URL like the following into a web browser:

```
http://machine/cgi-bin/nph-dods/version
```

Remember to replace *machine* with the machine you're using. Also, the CGI directory you're using may not be called `cgi-bin`. If you're not using a CGI directory, see the next step.

- If your server uses name conventions to identify CGI programs, change the names of the dispatch script to conform with the local convention: e.g. `nph_dods` to `nph_dods.cgi`. The other service program names do not need to be changed.

  Testing: Try to get a version number from the DODS software. If your CGI programs are identified with a suffix like `.cgi`, try a URL like this:

  ```
  http://machine/nph-dods.cgi/version
  ```

- If you are using DODS release 3.2 or later (your dispatch script will be called `nph-dods` if this is true), you also need to make sure that the dispatch configuration file `dods.ini` is also copied into the CGI directory. See Section 2.2 on page 18 for instructions about how to configure this file.

  Testing: This cannot be tested without having some data installed.

❹ See Chapter 3 for instructions on installing and testing the data to be served.

See Chapter 6 for more tests to make sure you've done each step correctly.

> **NOTE:** In addition to some specialized Perl modules, the DODS server CGI programs use a few standard (and relatively common) Perl modules, including LWP, HTTP, HTML, and MIME. DODS is tested with the releases of the modules that ship with DODS, and may not work with subsequent releases of these packages. On the other hand, you may have already existing CGI programs that use later releases of these packages which may balk at using older versions. In this case, it is best to establish a second CGI directory, and segregate the DODS servers and their Perl modules there.

The CGI configuration of a web server is dependent on the particular web server you use, and its configuration data. Consult the documentation for that server for more information. Our observations are that for most servers, having a CGI directory is the default situation, and there are a couple of potential security holes avoided with that configuration.

How to find the CGI directory.

To find which directory is the cgi-bin directory, you can look in the server's configuration file for a line like:

```
ScriptAlias /cgi-bin/ /usr/local/etc/httpd/cgi-bin/
```

For both the NCSA and Apache servers, the option `ScriptAlias` defines where CGI programs may reside. In this case they are in the directory /usr/local/etc/httpd/cgi-bin. URLs with 'cgi-bin' in their path will automatically refer to programs in this directory.

At this point, you will be wondering if things are working yet or not. Again, check out Chapter 6 for a detailed set of tests to get you going.

2.2 Configure the Server

NOTE: If you are installing a version of DODS before release 3.2, you can ignore this section. You can tell which version you have by the name of the dispatch script. Before release 3.2, the script was called `nph-`*api*, where *api* is the abbreviation of the API used to access your data files. (e.g. `nc` for netCDF, `jg` for JGOFS, etc.)

The `nph-dods` dispatch script receives a request and dispatches it to the appropriate service program. The way it figures out what is the appropriate service program is to match the filename part of the URL it received with a set of patterns listed in a configuration file. Each line of the configuration file contains a pattern and the name of a "handler" to use for files that match that pattern. Note that the pattern must match the entire filename, including whatever directories are part of that name.

Here is the default configuration file:

```
.*\.(HDF|hdf)(.Z|.gz)*$ hdf
.*\.(NC|nc|cdf|CDF)$ nc
.*\.(mat|Mat|MAT)$ mat
.*\.(dat|bin)$ ff
.*\.(pvu)(.Z|.gz)*$ dsp
.*\/test$ jg
.*\/[^\.]+$ jg
```

Each line consists of a regular expression pattern followed by an abbreviation like "jg" or "mat". Consider a URL like this:

```
http://dods.org/cgi-bin/nph-dods/data/ctd.mat?temp
```

When the `nph-dods` script is executed, the "file name" part of the URL is `/data/ctd.mat`. Testing this against the default initialization file matches the third line, which indicates that the `mat` (Matlab) service programs are the ones to

use to process this request. The request is then dispatched to the `mat-dods` (or `mat-dds` or `mat-das`) programs for processing.

> **NOTE:** The default configuration file is set up so that files without extensions are handled by the JGOFS service programs. If this is not right for your installation, delete the last line shown above.

"Regular expressions", advanced pattern-matching languages, are a powerful feature of Perl and many other computer programs. Powerful enough, in fact, to warrant at least one book about them (Mastering Regular Expressions by Jeffrey Friedl, O'Reilly, 1997). You can find a brief tutorial to regular expressions in the DODS bookshelf. See the DODS documentation page at DODS Home page..

(For a complete reference online (which is not a particularly good place to learn about them for the first time), see http://www.perldoc.com/perl5.6/pod/perlre.html.)

Briefly, however, the above patterns test whether a filename is of the form *file*`.`*ext*`.`*comp*, where *comp* (if present) is `Z` or `gz`, and *ext* is one of several possible filename extensions that might indicate a specific storage API. If a file has no extension, it is assumed to be a JGOFS file.

If these default rules will not work for your installation, you can rewrite them. For example, if all your files are HDF files, you could replace the default configuration file with one that looks like this:

```
.* hdf
```

The `.*` pattern matches all possible patterns, and indicates that whatever the name of the file sought, the HDF service programs are the ones to use.

If you have a situation where all the files in a particular directory (whatever its extension) are to be handled by the DSP service programs, and all other files served are JGOFS files, try this:

```
\/dsp_data\/.* dsp
.* jg
```

The rules are applied in order, and the first rule with a successful match returns the handler that will be applied. The above set of rules implies that everything in the `dsp_data` directory will be handled with the DSP service programs, and everything else will be handled with the JGOFS programs.

2.3 Compression

Data compression can be a confusing subject for people installing a DODS server. Part of the confusion arises because there are two entirely separate issues here. The first is that data may be *stored* in a compressed format, and the second is that data may be *transmitted* in a compressed format. These issues are completely independent of one another; data stored in compressed form can be transmitted in uncompressed form, and *vice versa*.

2.3.1 Storing Your Data In Compressed Form

The DODS dispatch script is equipped to handle compressed data. If a request URL arrives for a data file ending with `.Z` or `.gz`, the dispatch script uses `uncompress` or `gzip` to produce a decompressed file in a temporary location.

> **NOTE:** The introduction of DODS release 3.2 produced substantial changes (and improvements) in how compressed data files are handled. The instructions for pre-3.2 releases are included below.

Create a temporary directory to hold uncompressed data.

If you have compressed data and need to take advantage of this feature, follow these steps:

❶ Check to make sure that `uncompress` or `gzip` is installed on your computer.

❷ Create a temporary directory somewhere. Make sure that this directory is owned (or at least writable) by the `httpd` process.[1]

❸ Edit the dispatch script to change the definition of `$dispatch-cache_dir` to point to the new temporary directory. By default, this is `/usr/tmp`.

That should be enough to make your server serve compressed files.

The dispatch script will keep your temporary directory from overflowing by occasionally deleting old files. The default maximum cache size is 50 megabytes. If you have more files in the directory than that, the script starts deleting the oldest ones first until the size is lower than the given limit. To change the maximum cache size, you can edit the dispatch script. Look for a line like this:

```
purge_cache($dispatch->cache_dir(), 50);
```

[1] The user is yet another configurable feature. For the Apache server, you can look for a line in the configuration file specifying the user, using the keyword `User`. On my machine, where `httpd` runs as `nobody`, it looks like this: `User nobody`.

The numeric argument to `purge_cache` indicates the cache size limit, in megabytes.

If you need to serve files compressed with another compression technique (besides `gzip` or `compress`), you can edit the `DODS_Cache.pm` perl module.

First change the regular expression that marks compressed files to something like this:

```
my $compressed_regex = "(\.gz|\.Z|\.bz2)";
```

Then edit the `decompress_and_cache` function to contain a test for bzip2 files. This might read like this:

```
sub decompress_and_cache {
    my $pathname = shift;
    my $cache_dir = shift;

    my $cache_entity = cache_name($pathname, $cache_dir);

    if ((! -e $cache_entity) && (-e $pathname)) {
      if ($pathname =~ m/.*\.bz2/) {
        my $uncomp = "bzip2 -c -d " . $pathname . " > " .  $cache_entity;
      } else {
        my $uncomp = "gzip -c -d " . $pathname . " > " . $cache_entity;
      }
        system($uncomp);
    }

    return $cache_entity;
}
```

2.3.2 Storing Your Data In Compressed Form (pre v.3.2)

The DODS dispatch script for HDF files is equipped to handle compressed data. If a request URL arrives with a data file ending with `.Z` or `.gz`, the dispatch script uses `uncompress` or `gzip` to produce a decompressed file in a temporary location.

Create a temporary directory to hold uncompressed data.

If you have compressed data and need to take advantage of this feature, follow these steps:

❶ Check to make sure that `uncompress` or `gzip` is installed on your computer.

❷ Create a temporary directory somewhere. Make sure that this directory is owned (or at least writable) by the `httpd` process.[2]

❸ Edit the dispatch script to change the definition of `$spool_dir` to point to the new temporary directory.

[2]The user is yet another configurable feature. For the Apache server, you can look for a line in the configuration file like this: `User nobody`.

That should be enough to make your server serve compressed files.

To keep things tidy, you should consider creating a `cron` job to clean out the temporary directory from time to time.

> **CAUTION:** Since the old decompression algorithm causes decompressed files to be written to a single directory under their original file name, there may be errors if you have more than one file with the same name, even if they are kept in different directories. For example, if you have two files, called `data/1993/january.nc.gz` and `data/1994/january.nc.gz`, they will both produce a file called `january.nc` in the temporary directory. Using compression in a case like this can produce incorrect results. To get around this problem, consider a new naming convention, stop using compression, or upgrade to DODS 3.2.

If you need to serve files compressed with another compression technique (besides `gzip` or `compress`), you can edit the dispatch script to add it. You'll see, when you open that file, a conditional testing the file type. Just add another condition to it. Here's an example:

```
if (! -e $UCfile_name) {
    if ($file_type eq "Z") {
        $uncomp = "uncompress -c " .  $file_name . " > " .  $UCfile_name;
    } elsif ($file_type eq "bz2") {
        $uncomp = "bzip2 -c -d " . $file_name . " > " .  $UCfile_name;
    } else {
        $uncomp = "gzip -c -d " . $file_name . " > " .  $UCfile_name;
    }
    system($uncomp); #uncompress and put output to spool
}
```

2.3.3 Enabling Transmission of Compressed Data

To save communication bandwidth, the DODS server can send compressed data to certain clients who are equipped to handle it. When making a request for data, a client can signal to the server that data may be sent compressed. If a server receives such a signal, it looks for a helper program called `deflate`, and it runs the outgoing data through that program on its way back to the client.

All the DODS servers can handle data compression for transmission if the `deflate` program is installed on the `$PATH` used by the web server. To keep things simple, we recommend that it be kept in the same CGI directory as the other DODS service programs. When you download the precompiled binary tar files, the `deflate` program comes in the `$DODS_ROOT/etc` directory with the other service programs.

2.4 Security

Many data providers ask whether a DODS server can be configured to limit access to a particular file or sets of files. The answer is that DODS is as flexible as the http server you use to implement it. DODS clients are enabled to prompt the user for usernames and passwords if the http daemon instructs them to. Most web servers will also allow usernames and passwords to be embedded in the requesting URL like this:

```
http://user:password@www.dods.org/nph-dods/...
```

Depending on the server, you can restrict access to the DODS CGI programs or to the individual data files or directories. Some sites find it simplest to set up two servers: one public and one secure.

3
Installing Your Data

After installing the CGI program and the services, the data to be provided must be put in some location where it may be served to clients. The DODS server navigates a directory tree rooted in the web server's *document root* directory. This is the directory (often called `htdocs`) in which the web server first looks for web pages to serve. If you send your web server this URL:

Now the server is installed, you have to install the data.

```
http://yourmachine/trash.html
```

It will look for the file `trash.html` in the document root directory. Again, the location of this directory depends entirely on the web server you use and its configuration data. A server may also be enabled to follow links from the root directory tree, which means that you can store your data somewhere else, and make symbolic links to it from the root directory tree. By default, this is disabled for most servers, since it can become a security problem, but it is relatively easy to enable it. (The option is called `FollowSymLinks` for Apache users.) Further, there may be other options provided by the specific server used in a particular installation. In other words, there is really no way to avoid consulting the configuration instructions of the web server you use.

As noted, the location of the data depends not only on the configuration of the Web server, but also on the API used to access the data requested. For example, the netCDF server simply stores data in files with a pathname relative to the document root directory, `htdocs`, while the JGOFS server uses its data dictionary to specify the location of its data. Refer to the specific installation notes for each API for more information about the location of the data.

Fortunately, this is not hard.

3.1 Special Instructions for the NetCDF, HDF, DSP and Matlab Servers

To install data for the NetCDF, HDF, DSP or Matlab formats, follow these steps:

❶ Make a data directory somewhere in the document root tree. For Apache, this is the `htdocs` directory and all its subdirectories. If your web browser is configured to follow symbolic links (Apache: `FollowSymLinks` is enabled), you can just put links to the data files or to their directories in the document root tree. Do be aware that the reason symbolic links are not enabled by default is that there are potential security holes in its use. Consult your web server's documentation for more information.

❷ Put the data files into the data directory you've just made.

❸ Test the URL with a web browser. See *The DODS Quick Start Guide* for information about how to construct a URL if you know the machine name.

That's all.

3.2 Special Instructions for the DODS/JGOFS Server

The JGOFS data system is a distributed, object-based data management system for multidisciplinary, multi-institutional programs. It provides the capability for all JGOFS scientists to work with the data without regard for the storage format or for the actual location where the data resides.

A full description of the JGOFS data system and U.S JGOFS program can be found at http://www1.whoi.edu/.

3.2.1 About JGOFS Servers

The JGOFS API reads "objects", not files.

In the JGOFS data system, scientific datasets are encapsulated in *data objects* . Data objects are simply names defined in the JGOFS *objects* dictionary which associate specific scientific datasets with a computer program, known as an access *method* , which can read them. This mechanism provides a single, uniform access methodology for all scientfic datasets served by this data server, regardless of whether they are simple single file or complex multi-file datasets.

Here is an example of a JGOFS object dictionary entry:

```
test0=def(/usr/local/apache/htdocs/data/t0)
```

In this example,

`test0` defines the data object name.

`def` specifies the access 'method' or program used to read the scientific dataset.

`/usr/local/apache/htdocs/data/t0` specifies the scientific dataset to read.

The DODS/JGOFS server requires only two components of a complete JGOFS installation to function, the 'objects' dictionary and the access 'methods' used to read local data files.

For those sites who wish to use the DODS/JGOFS server without installing a complete JGOFS data system we provide these required components. For sites which currently have a JGOFS data system installed the DODS/JGOFS servers can be configured to use the current installation.

Sample data is included for testing purposes.

Included with the DODS/JGOFS server is an example object dictionary, named `.objects`, and a binary version of the JGOFS default (`def`) method. The `def` method can read single or multi-file flat ASCII datasets. An example ASCII dataset is also provided to test the server installation.

> **NOTE:** The DODS/JGOFS server currently *requires* the object dictionary to be named `.objects`.

The server uses two mechanisms for determining the location of the data objects dictionary and access method directory:

❶ The CGI dispatch script `nph-dods` defines two shell variables, `JGOFS_OBJECT` and `JGOFS_METHOD`. These are used by the DODS server to determine the location of the objects dictionary and access method directory. Initially, the definition of these variables is commented out in the `nph-jg` dispatch script.

❷ If the shell variables `JGOFS_OBJECT` and `JGOFS_METHOD` are not defined, the DODS/JGOFS server checks for the existence of a `jgofs` user in the `/etc/passwd` file. If a `jgofs` user exists it will set the `JGOFS_METHOD` location to `jgofs/methods/`, and the `JGOFS_OBJECT` location to `jgofs/objects/`.

❸ If the server cannot locate the objects directory using either of the above two mechanisms it will search the current working directory.

If the DODS/JGOFS server cannot resolve the filesystem paths for the objects dictionary or methods directory using any of these mechanisms it will report an error to the client and exit.

> **TIP:** If you have an existing `jgofs` user but would like to use a different objects dictionary for the DODS server then specifying the `JGOFS_OBJECT` and `JGOFS_METHOD` variables in the `nph-jg` script will override your existing JGOFS setup.

Once you have installed the DODS service programs (the helper programs that belong in the CGI directory), you must insure that the DODS/JGOFS server can locate the objects dictionary and methods directory.

To use the `.objects` dictionary and `def` access method supplied with the DODS/JGOFS server you must edit the `nph-jg` script. In the 'nph-jg' script, update the lines specifying the `JGOFS_OBJECT` and `JGOFS_METHOD` variables. These variables must contain the directory name containing the `.objects` dictionary and `def` access method. Also, you must remove the comment specifiers at the beginning of these lines.

> **TIP:** Though not required, copying these two files to the cgi-bin location will keep all the DODS/JGOFS files in one convenient location.

The httpd must have permissions to access the dictionary.

To make data accessible via the server, the entries in the objects dictionary must define an object name, and associate that name with an access method and a dataset to serve. Scientific datasets served as JGOFS objects can be located anywhere within your computer's filesystem. However, the user and group used to run the WWW daemon must have permission to read these files.

> **ANOTHER TIP:** To find out which user/group httpd is running as, look in the `httpd.conf` file for lines like:
>
> ```
> User nobody
> Group nobody
> ```

To test your server installation using the provided dictionary, and test data, edit the `.objects` dictionary so that the `test0` object points to the files located in the `DODS/etc/data` directory. You are free to move these files to any location on your computer so long as the WWW daemon has permission to read these files, and the objects dictionary points to their location.

Using the supplied test datasets, try issuing the following URL:

```
http://yourmachine/cgi-bin/nph-jg/test0.dds
```

You will see something like this:

```
Dataset {
    Sequence {
        String leg;
        String year;
        String month;
        Sequence {
            String station;
            String lat;
            String lon;
            Sequence {
                String press;
                String temp;
                String sal;
                String o2;
                String sigth;
            } Level_2;
        } Level_1;
    } Level_0;
} test0;
```

You can go further and ask for some data:

```
http://yourmachine/cgi-bin/nph-jg/test0.asc
```

You should see a comma-separated list of data values.

3.2.2 Attribute Data for DODS/JGOFS

You may need to add ancillary attribute data for a JGOFS install.

The JGOFS data access API does not support a wide variety of attribute data. To meet the attribute data needs of DODS, the DODS server will look for a file named for an *ancillary data* file named *object*`.das`, in the same directory which contains the primary datafile for the JGOFS object referenced by the URL.

For example, suppose that you have installed the DODS/JGOFS server, as described above, and a partial listing of the data objects dictionary contained the following definitions:

```
test0=def(/home/httpd/htdocs/data/test0.data)
s87_xbt=def(/home/httpd/htdocs/data/xbt.catalog)
```

To find attribute data for the JGOFS object 'test0', the server will search for the file:

```
/home/httpd/htdocs/data/test0.das
```

The file itself should simply be a text version of the DAS you wish the server to supply to clients. For example:

```
Attributes {
    leg {
       String Description "The number of the voyage leg.";
    }
    year {
       String Range "Data between 1982 and 1992";
    }
}
```

3.2.3 Data Types for DODS/JGOFS Data

DODS allows JGOFS to support new data types.

The JGOFS data access system is designed only to manipulate and return string data. That is, all data is returned to the client as character strings. The DODS/JGOFS server supports a variety of other data types, allowing the data provider (i.e. you) to specify the variable type to use when returning data to the remote client. To accomplish this, the server will look for an ancillary DAS file (described in Section 3.2.2 on page 29) for the object, and search that file for attribute containers for each variable. If the variable's container exists, it will then look for the `DODS_Type`, and `missing_value` attributes in that container.

If `DODS_Type` exists it will set the output type of the variable. If the `missing_value` attribute is set, it well replace any `nd`, or missing data values in the output stream for that variable to the value specified in the `missing_value` attribute. If the `missing_value` attribute is not specified, the server will set the missing value field to the largest value possible for specified type. The server assumes that the data provider knows the proper data type to use to contain the full range of possible data values for the object. If the JGOFS variable values cannot be converted to the specified data type, then a `missing_value` value will be returned.

Suppose that you have installed the DODS/JGOFS server, as described above, and a partial listing of the data objects dictionary contained the following definitions:

```
test0=def(/home/httpd/htdocs/data/test0.data)
```

The ancillary DAS file for this dataset could contain these new attributes:

```
Attributes {
    leg {
        String DODS_Type "Int32";
        Int32 missing_value -99;
    }
    press {
        String DODS_Type "Float32";
        Float32 missing_value -1.99.;
    }
    temp {
        String DODS_Type "Float64";
    }
}
```

The server will use the values specified in the `DODS_Type` attribute for any variable container provided, to define the output DODS variable type. In this example, the variable named `leg` will be returned as a 32-bit integer (Int32), with missing values set to -99. Similarly, variable `press` will be returned as a 32-bit floating point number, with missing values set to -1.99. The variable `temp` will be returned as a 64-bit floating point number, but missing values will be returned as the largest possible 64-bit floating point number (1.797693e+308).

3.2.4 Limiting Access to a DODS/JGOFS Server

Generally speaking, it is straightforward to create DODS servers with limited access. The details depend on the web server you are using, however, and are not be covered in this document.

JGOFS offers new opportunities for limiting data access.

However, the data dictionary structure of the JGOFS data access API provides a second way to differentiate between public and private datasets. To do this, you would maintain different data object dictionaries which encapsulate different scientific datasets. One dictionary would define data objects for general public access, and other dictionaries could define data objects for specific user communities, where access is resticted.

To accomplish this the site would maintain different versions of the `nph-dods` dispatch script. For example, using `nph-dods` for public access, this file would set the `JGOFS_OBJECT` variable to the public-access dictionary, and another dispatch script, say `nph-dods-globec` would set the `JGOFS_OBJECT` variable to a restricted-access dictionary.

The WWW daemon is then configured to permit general access to `nph-dods` and restricted access to `nph-dods-globec`. Finally, since the dispatch script `nph-dods-globec` does not conform to the general DODS usage of `nph-<root>`, the restricted-access user community must be informed to use this dispatch script to access these restricted datasets.

3.3 Special Instructions for the FreeForm Server

The DODS FreeForm server can serve data stored in a wide variety of formats. To read data, it uses a *format file* that describes the structure of the data file to be read. Armed with a knowledge of the file structure, the server can read a data file, and send it along to the client.

The FreeForm server has its own documentation: see *The DODS Freeform ND Server Manual*. You will find a complete description of the server, as well as instructions on how to write a format file for your data.

4

Installing the DODS Relational Database Server

The DODS Relational Database Server (DRDS) allows data managers to serve data stored in relational DBMS systems, such as Oracle, SQL Server, or mySQL. The DRDS is written in Java, and uses the Java Database Connectivity API (JDBC) to connect with the DBMS.

As of version 1.1, the DRDS cannot issue table join commands. This means it cannot serve multiple tables connected by index numbers. If your DBMS supports views, you can use these to allow users to query multiple tables. If your DBMS does not support views, you need to put your data into a single table to use DRDS.

Here is a brief description of how to set up the DRDS. These instructions are for DRDS version 1.1 and later.

This Java program requires some software already installed. Like Java.

There are a certain number of prerequisites. The variety of installations and software make it impossible to describe here how to install these things and figure out the relevant parameters. Nonetheless, they must be installed. Though we can't offer instruction, we can at least offer a checklist. Here is a list of the software that must be installed, and the parameters you must determine:

❶ Java. You must have at least Java version 1.2 (Inexplicably, this is often called version 2.0).

❷ Data in a database that supports Java Database Connectivity (JDBC). If your data is in multiple tables, you will need to prepare a view of the data to be served.

❸ A web server that can execute servlets. DRDS has been extensively tested with the Tomcat servlet engine, which can be run either as a module to Apache, or by itself.

❹ The JDBC client library (also called the "driver") for your DBMS. This is specific to the kind of DBMS. That is, an Oracle JDBC driver won't necessarily work with a MySQL DBMS.

❺ You must know the name of the driver. This is a string of characters identifying the particular JDBC driver in use. For Oracle, it reads something like this: `oracle.jdbc.driver.OracleDriver`.

❻ Your DBMS must have a user with enough privilege to read data from the network. The DRDS is a read-only application, and cannot issue commands that will try to change the data.

❼ You must know the "Connection URL" that will be used to contact your DBMS. This is a long and awkward-looking string that identifies the machine, the driver, the port, and some other information used by the JDBC driver to contact your DBMS. For a recent Oracle installation, the Connection URL looked like this:

```
jdbc:oracle:thin:@dods.org:1521:orcl
```

Figuring out the Connection URL will probably be the most challenging part of installing DRDS. It's not a great deal of help to say so, but we're hoping to make up in sympathy what we cannot offer in concrete assistance.

After you've checked off the items in this list, download the DRDS distribution from the DODS Home pageor the DODS Java home page. For a basic installation, you want the file called `dods.war`. To install the server, put this entire file into the "webapps" directory of your servlet engine (e.g. Tomcat). When you restart the servlet engine, it will unpack the archive into the directory `webapps/dods`.

To configure the servlets, you'll need to edit the file `webapps/dods/WEB-INF/web.xml`. This is described in the next two sections. After you finish editing the configuration file, restart the servlet engine again, and the server should be runnning.

4.1 Test Server

When installing a server, it's useful to isolate the steps so you're not trying to tune several parameters at the same time. To assist here, the DRDS comes with a test server that invents data to return to a client requesting data. You can use it to test your servlet engine and the DODS code, without taxing the JDBC drivers or your DBMS.

Use the test server to test your installation

Edit the file `webapps/dods/WEB-INF/web.xml`. In the `ContextManager` element, add this text:

```
<Context path="/dods"
    docBase="webapps/dods"
    crossContext="false"
    debug="0"
    reloadable="true" >
</Context>
```

In the `web-app` section, you'll want a servlet declaration like the following:

```
<!-- DODS test server (DTS) setup -->
<servlet>
    <servlet-name>
        dts
    </servlet-name>
    <servlet-class>
        dods.servers.test.dts
    </servlet-class>
    <init-param>
        <param-name>iniFilePath</param-name>
        <param-value>/home/user/dodsServer/dts</param-value>
    </init-param>
    <init-param>
        <param-name>iniFileName</param-name>
        <param-value>dods.ini</param-value>
    </init-param>
</servlet>
 <servlet-mapping>
    <servlet-name>dts</servlet-name>
    <url-pattern>/dts</url-pattern>
</servlet-mapping>
 <servlet-mapping>
    <servlet-name>dts</servlet-name>
    <url-pattern>/dts/*</url-pattern>
</servlet-mapping>
<!-- End - DODS test server (DTS) setup -->
```

The servlet engine is now configured with information that will allow it to run the DRDS servlet. Now we need to configure the DRDS servlet. The `web.xml` file

indicates a file containing initialization information for the DRDS servlet. In the above, we've called it `dods.ini`. (Look at the `iniFilePath` and `iniFileName` parameters.) For the test server, we can use the following for the text of this file:

```
[Server]
info_dir      = /home/user/dodsServer/drds/info/
dds_cache_dir = /home/user/dodsServer/drds/dds/
das_cache_dir = /home/user/dodsServer/drds/das/
```

Describe the imaginary datasets with the DDS and DAS.

These declarations tell the test server where to find the DDS and DAS files necessary for serving data.

For each dataset you want to test, you will need to describe the dataset in terms of the DDS, DAS, and "info" responses you would get from the DODS dataset. These are text files containing data about the data to be served. The *The DODS User Guide* has more information about each of these data structures. This will require the creation of two files with the same name. One should contain the DDS information and one the DAS. (You can ignore the `info` data for now.)

Here's a sample DDS:

```
Dataset {
  Sequence {
    String dou;
    Float64 mcd_id;
    String month_id;
    String ship;
    String cruiseid;
    UInt32 year_id;
    String crdate_begin;
    String crdate_end;
  } CRUISES;
} GLOBEC;
```

Here's a DAS that can go with this DDS:

```
Attributes {
    mcd_id {
      String long_name "MCD_ID";
    }
    cruiseid {
      String long_name "CRUISEID";
    }
}
```

Put these files in the directories indicated by the `dds_cache_dir` and `das_cache_dir` lines in the `dods.ini` file. Give them the same name, say `RALPH`, and you're ready to test the server. Restart the servlet to read the configuration files, and try a URL like this in a web browser:

```
http://host:8080/dods/dts
```

The machine and port name are specific to the servlet engine you are using. The above is the default if you're using Tomcat as a standalone server. The "dods" in the URL is from the subdirectory of the `webapps` directory, and the "dts" is from the `servlet-mapping` section of the `web.xml` file. This URL should bring up a list of datasets you can select from, which should include your fake dataset "RALPH."

4.2 Serve Real Data

To install the real data in your servlet engine, you will have to complete the installation of JDBC, make sure your DBMS is correctly configured, and reconfigure the DRDS to use the JDBC library to find your data. Before starting this part, make sure that the JDBC drivers are properly installed.

Now that the test server works, attach the DRDS to real data.

To begin, edit the `web.xml` file to include the real DRDS information. Here's an example:

```
<!-- DODS SQL server (DRDS) setup -->
<servlet>
    <servlet-name>
        drds
    </servlet-name>
    <servlet-class>
        dods.servers.sql.drds
    </servlet-class>
    <init-param>
        <param-name>iniFilePath</param-name>
        <param-value>/home/user/dodsServer/drds</param-value>
    </init-param>
    <init-param>
        <param-name>iniFileName</param-name>
        <param-value>dods.ini</param-value>
    </init-param>
</servlet>
 <servlet-mapping>
    <servlet-name>drds</servlet-name>
    <url-pattern>/drds</url-pattern>
</servlet-mapping>
 <servlet-mapping>
    <servlet-name>drds</servlet-name>
    <url-pattern>/drds/*</url-pattern>
</servlet-mapping>
<!-- End - DODS SQL server (DRDS) setup -->
```

You can use the same initialization file as for the test server, if you add a section to it containing JDBC information, including the JDBC driver name, the

Connection URL, and the username and password for the DBMS.

```
[JDBC]
Driver            = oracle.jdbc.driver.OracleDriver
ConnectionURL     = **your-jdbc-connection-url**
username          = **username**
password          = **password**
MaxResponseLength = 300

[Server]
info_dir      = /home/user/dodsServer/drds/info/
dds_cache_dir = /home/user/dodsServer/drds/dds/
das_cache_dir = /home/user/dodsServer/drds/das/
```

4.2.1 Configure the Table Data Types

Each database table (including table "views") you wish to serve must be described with a DODS DDS, DAS. Optionally, you may also include an "info" message to provide information to users about the source and quality of the data provided, as well as restrictions on its use. These structures are contained in text files that must be tailored to each table to be served.

Here is an example using a table called "b31" that is defined like this (using Oracle):

Name	Null?	Type
ID	NOT NULL	NUMBER
CLASS	NOT NULL	CHAR(1)
TEXT	NOT NULL	VARCHAR(270)

Determine the DODS data types that most closely match yours.

First, determine the DODS data types that correspond with the JDBC data types (see the data type mapping in table 4.1). These mappings are used to create a DDS. In the cache directories defined in the DRDS `dods.ini` file, create the files `dds/b31` (see the example DDS below), `das/b31` (see the example DAS below), and `info/b31` (if desired).

> **NOTE:** The DODS data types described in table 4.1 are only the primitive data types (except for Array). DODS supports more sophisticated data types: Grids, Arrays, Structures, and Sequences, among others For information about all these, refer to the *The DODS User Guide*.

Make a DDS

In a file named for your table, in the directory nominated by the `dds_cache_dir` directive in the `dods.ini` file (see 38), put a DDS for your table. This must

JDBC Data Type	DODS Data Type
TINYINT	Byte
SMALLINT	Int16
INTEGER	Int32
BIGINT	No sensible mapping. Use Int32.
REAL	Float32
FLOAT	Float64
DOUBLE	Float64
DECIMAL	No sensible mapping. Use Float64
NUMERIC	No sensible mapping. Use Float64
BIT	Boolean
CHAR	String
VARCHAR	String
LONGVARCHAR	Array(of bytes)
BINARY	Array(of bytes)
VARBINARY	Array(of bytes)
LONGVARBINARY	Array(of bytes)
DATE	String
TIME	String
TIMESTAMP	String

Table 4.1: Mapping from JDBC Data Types to DODS Data Types

Describe the structure of your dataset with the DDS.

contains declarations for each of the data types you expect a user to ask for (or select over). The data types you don't list are effectively invisible to the user of the DRDS. Use table 4.1 to determine the data types that most closely correspond to the data in your tables.

The data served from a DRDS server is always enclosed in a DODS data type called a "Sequence." Each instance of a Sequence corresponds to one row of a relational table. In effect, the DDS declares the data types and arrangement of the table columns. For more information about what the DDS is, see *The DODS User Guide*. There is also a very cursory introduction to these structures in *The DODS Quick Start Guide*.

> **NOTE:** When writing your DDS, be sure the name of the top-level sequence matches the database table name. It is the sequence name, rather than the dataset name, that is used in the SQL statement that will be submitted to your DBMS.

For our example server, we would create a file called "b31," and put it in the `dds_cache_dir`. The file contents would look like this:

```
Dataset {
  Sequence {
    Int16 id;
    String class;
    String text;
  } b31;  # Sequence name
} b31;    # Dataset name
```

Provide details about each variable with the DAS.

Make a DAS

Making a DAS is precisely comparable to the DDS. Make a file called "b31," and put it in the `das_cache_dir`. Here's the DAS for our example.

```
Attributes {
#  b31 {
    id {
      String long_name "Id";
    }
    class {
      String long_name "class";
    }
    text {
      String long_name "text";
    }
#  }
}
```

NOTE: Currently, the sequence nesting cannot be declared in the DAS, which is why it is commented out in the example.

Make an Info File

Info files can make your data more useful to users.

This is entirely optional, but it can be helpful to users who want to know more about your data. Create a file with HTML in it (without the HTML or BODY tags), and put it in the `info_cache_dir` directory with the same name as the DDS and DAS files, it will be served to the user when they make an info request. See *The DODS Quick Start Guide* for information about the contents of the info response.

Done

You should now be done installing the DRDS. The URL for this server should be the same as the URL for the test server, using the "drds" directory instead of "dts," per the `servlet-mapping` direction in the `web.xml` file. (See Section 4.2 on page 37.) This should bring up a list of datasets you can select from.

5
Constructing the URL

After a dataset has been installed, and the server programs installed, you need to know what its address is. If you've tested the web server installation as in Chapter 2, you know the beginning of the URLs that will point to data at your site.

How do you address the data you just installed?

(If you're using the DRDS Java server, see Section 4.2.1 on page 41.)

Suppose your web server's document root is at:

```
/usr/local/spool/http/
```

And you are serving netCDF data that you've stored at:

```
/usr/local/spool/http/datasets/atlantic/cooldata.nc
```

Following the test URL in Chapter 2, this would be the URL to use in a DODS client.

```
http://yourmachine/cgi-bin/nph-dods/datasets/atlantic/cooldata.nc
```

Note that this URL will work for a DODS client, but not for a standard web browser. If you want to use a standard web browser to test your installation, read on, and also take a look at *The DODS Quick Start Guide*. For DODS releases before 3.2, the dispatch script would be called `nph-nc`. See note on page 10.

5.1 Constructing a DODS/JGOFS URL

The situation is slightly different for JGOFS data.

Suppose that you have installed the DODS/JGOFS server, as described above, on the machine `dods.org`. And a partial listing of the the data objects dictionary contained the following definitions:

```
test0=def(/home/httpd/htdocs/data/test0.data)
s87_xbt=def(/home/httpd/htdocs/data/xbt.catalog)
s87_ctd=def(/home/httpd/htdocs/data/ctd.catalog)
htf=autoedges(/fronts/hatteras-to-florida/edges)
htn=autoedges(/fronts/hatteras-to-nova-scotia/edges)
glk=autoedges(/fronts/great-lakes/edges)
```

A URL that references the data object 's87˙xbt' would look like:

```
http://dods.org/cgi-bin/nph-dods/s87_xbt
```

6
Testing the Installation

It is possible to test the DODS server to see whether an installation has been properly done. The easiest way to test the installation is with a simple Web client like Netscape or Mosaic. (A command-line web client called `geturl` is provided in the DODS core software which can retrieve text from Web servers, and print it on standard output. Look for it in the `$(DODS_ROOT)/bin` directory. If you try it out on a couple of URLs you are familiar with, you'll quickly see how it works.)

You can test the server installation first, and when you are sure the server is working well, test the DODS software.

Check out the Quick Start Guide for information about forming DODS URLs.

6.1 Testing the Web Server

To begin, confirm that the server works properly by simply sending a URL for a simple web page, or without a file at all:

```
http://yourmachine
```

If you don't get anything with this, examine the document root directory (`htdocs`), and make a request for a file in that directory:

```
http://yourmachine/somefile.html
```

Debugging these early steps is essential to getting a DODS server to work, but troubleshooting any problems you have with these steps is an issue specific to the web server you are using.

Once you have a web server that can return a web page, you should exercise the CGI configuration. Here is the simplest possible CGI program:

```
#!/usr/local/bin/perl
print "Content-type: text/html\n\n";
print "Hello World!\n\n";
```

Put this text in a file, and try to execute it from the command line. You may have to edit the first line in case your copy of Perl isn't stored in the `/usr/local/bin` directory. Don't forget to make the file executable. (Use the `chmod` command.)

When you can execute this from the command line, put it into the CGI directory (and make sure that the permissions will allow the `httpd` to execute it), and try to execute it with a URL like this:

```
http://yourmachine/cgi-bin/test-cgi
```

If you are configuring your site to use suffixes instead of a directory to identify CGI files, name it accordingly:

```
http://yourmachine/test.cgi
```

Entering one of these URLs in a web server should cause the "Hello, World!" text to appear in the web browser window. If not, check the permissions of the CGI program, then consult the documentation for the web server.

If everything works so far, it's time to move on to testing the DODS installation.

6.2 Testing the DODS Software

You can run the DODS CGI programs locally, that is, without going over the internet. Sometimes this is the simplest way to make sure that everything is running. Try changing to the CGI directory and executing the DDS program by hand:

```
./nc_dds datafile.nc
```

The program should print the dataset's DDS on standard output.

The simplest remote test is simply to ask for the version of the DODS server, sending a URL like this:

```
http://yourmachine/cgi-bin/nph-dods/version
```

The DODS server will respond to this URL with some text containing release numbers. If you enter this URL into a standard web browser you're doing fine if you see a message like this:

```
Core version: DODS/3.2.5
Server version: nc/3.2.2
```

The DODS dispatch script can respond to a number of requests without the assistance of its helper programs, or the existence of any data. Besides the version request, you can also ask for the help and info messages:

```
> geturl http://dods.gso.uri.edu/cgi-bin/nph-dods/test.nc.ver
> geturl http://dods.gso.uri.edu/cgi-bin/nph-dods/test.nc.info
> geturl http://dods.gso.uri.edu/cgi-bin/nph-dods/test.nc.help
```

(For DODS releases before 3.2, the dispatch script is named after the data API it uses. That is, you would use `nph-nc` instead of `nph-dods` in the above example. See note on page 10.)

Now we need to test the requests that use the service programs, such as `nc_dds` and `nc_dods`. These programs compose their output by looking at data files, so testing these requires data to be in place.

To return the data attribute structure of a dataset, use a URL such as the following:

```
> geturl http://dods.gso.uri.edu/cgi-bin/nph-dods/data/test.nc.das
```

The `geturl` program knows about the DODS protocols, so you can also omit the `.das` suffix, and use the `-a` option to the `geturl` command. This tells `geturl` to append `.das` for you:

```
> geturl -a http://dods.gso.uri.edu/cgi-bin/nph-dods/data/test.nc
```

Refer to *The DODS User Guide* for a description of a data attribute structure. You can compare the description against what is returned by the above URL to test the operation of the DODS server.

Check the list of services and helper programs in Section 1.3 on page 12. From a web browser, you can access all the DODS services, except the data service (`*_dods`), which returns binary, not ASCII, data. That one can only be tested from a DODS client. However, if all the service programs work, and the data service is configured the same way, the odds are on your side.

Using the `.html` suffix produces the WWW Interface , providing a forms-based interface with which a user can query the dataset using a simple web browser. There's more about the WWW Interface in *The DODS User Guide*.

6.3 Error Logs

When troubleshooting a DODS server, the logs of the web server are quite useful. The Apache server keeps two logs, an Access log and an Error log. (You can find these in the `logs` subdirectory where you installed Apache.) The DODS server software writes any messages it issues to the error log.

You can make the DODS error messages slightly more verbose by editing the `DODS_Cache.pm` and `DODS_Dispatch.pm`. Find the `$debug` variable, and set it to 1 or 2 instead of zero.

6.4 User Support

As a last resort, you can use the DODS technical support service. The DODS project provides technical support by email: support@unidata.ucar.edu.

7
Documenting Your Data

Documenting your data will make it useful to more people.

DODS contains provisions for supplying documentation to users about a server, and also about the data that server provides. When a server receives an information request (through the `info` service that invokes the `usage` program), it returns to the client an HTML document created from the DAS and DDS of the referenced data. It may also return information about the server, and more detail about the dataset.

Users access this information by appending '.html' to a DODS URL. For example to get the HTML page for an HDF file, you might type something like this:

```
http://dods.org/cgi-bin/nph-dods/S00444.HDF.html
```

The Usage service will return important information about your dataset even if you do not write custom HTML files for it. If you do write those files they will be concatenated with the default information returned by the usage server.

If you would like to provide more information about a dataset than is contained in the DAS and DDS, simply create an HTML document (without the `<html>` and `<body>` tags, which are supplied by the `info` service), and store it in the same directory as the dataset, with a name corresponding to the dataset filename. For example, the dataset `fnoc1.nc` might be documented with a file called `fnoc1.html`.

You can also provide documentation for a class of files that may have a common root in their names. For example, a file that would be used for data files called `S00443.nc`, `S00444.nc` and `S00445.nc` could be called `S.html` (or `S00.html` in this case). This file should be located in the directory where the data is located.

If you'd like to append information to the `info` message for all the files on a server, you can write a file called *servername*`.html`, where *servername* is one of `nc`, `ff`, `hdf`, `jg`, and so on.

You may prefer to override this method of creating documentation and simply provide a single, complete HTML document that contains general information for

a dataset. If you call your documentation file `fnoc1.ovr`, the client will see only the contents of that file, and will not get the text generated by the dataset DAS and DDS, or the *servername*`.html` file.

More information about providing user information, including sample HTML files, and a complete description of the search procedure for finding the dataset documentation, may be found in *The DODS Toolkit Programmer's Guide*.

7.1 Special Instructions for the DODS/JGOFS Server

The DODS/JGOFS Server provides usage data in almost the same way as the other servers, but the `usage` helper program is called `usage-jg`, and it functions slightly differently. For the general DODS/JGOFS server and for each data object it serves, you can write a HTML document which the usage server will return when requested by users. You can write a HTML document that describes any special features of your particular DODS/JGOFS server and save that document in a file named `jg.html` in the cgi-bin directory that holds the server programs.

Yes, this is different for JGOFS, too.

> **TIP:** The `jg.html` file could be used to provide descriptive information, including the names, for all the data objects served at your site.

The only special thing about this file is that you should include only those HTML tags that would fall between the `<body>` and `</body>` tags. Thus it should not contain the `<html>`, `<head>`, `<body>` or their matching counterparts.

To provide HTML for each JGOFS data object you serve, create a file whose name is based on the names of the data object you want to describe. For example, a file that would be used for all the `s87_xbt` data object used in the previous section's example would be `s87_xbt.html`. This file must be located in the directory where the top-level datafile is located, as defined in the data objects dictionary.

Users access this information by appending `.info` to a DODS URL. For example to get the HTML page for the URL used in the previous section, you'd type:

```
http://dods.gso.uri.edu/cgi-bin/nph-dods/s87_xbt.info
```

8
The Catalog Server

Some DODS data servers are responsible for serving data from thousands of data files. Trying to assert some kind of order to a dataset like this can be a challenge. DODS allows you to create a catalog of these data files, which will allow a user to select among the data files according to variables or variable ranges.

A catalog server is just a DODS server that serves URLs.

A DODS *catalog server* , or *file server* , is just a collection of DODS URL's, compiled into a big list, along with some data a user might use to select among these data files. Typically the *selection data* is data not included in the files themselves, but is data *about* the file. For example, the URI AVHRR archive records the time each file was recorded next to the file's name.

The good thing about catalog servers is that they are just vanilla DODS servers, but used to serve information about data, rather than just data. The only requirement is that they serve Sequence data, and that one of the fields be called `DODS_URL`. This means that the procedures you've used to get your data served are exactly the same procedures you will use to get your data catalog served. The steps that are already done will not need to be done again.

Here are the steps in the process.

8.1 Step One: Install the DODS FreeForm ND Server

The catalog server typically uses the DODS FreeForm server, though it could use the JGOFS server just as easily, or any server that can serve DODS Sequence data. This guide shows how to use the FreeForm server to make a catalog server, since that's what most sites choose to do.

Any DODS server that serves Sequences can be a catalog server.

If you haven't done so already, download and install the FreeForm server (or whatever server you intend to use). If you're using the FreeForm server, move on to the next section. If not, look at the file server chapter in *The DODS Quick Start Guide* and make your catalog output look like that.

8.2 Step Two: Make Your Catalog and Format File

Make a catalog text file.

Once you've got the FreeForm server installed and working (check it out with one of the sample data and format files that come with it), compile a list of all the URLs of all the files you need to serve. Put this in a text file, along with the selection data you intend to use, one file per line.

Here's a part of the AVHRR archive catalog file used at URI:

```
1979/04/11:18:53:59  http://dods.gso.uri.edu/cgi-bin/nph-dsp/avhrr/1979/4/t79101185359.pvu.Z
1979/04/15:19:48:52  http://dods.gso.uri.edu/cgi-bin/nph-dsp/avhrr/1979/4/t79105194852.pvu.Z
1979/04/16:19:40:24  http://dods.gso.uri.edu/cgi-bin/nph-dsp/avhrr/1979/4/t79106194024.pvu.Z
1979/04/17:08:00:11  http://dods.gso.uri.edu/cgi-bin/nph-dsp/avhrr/1979/4/t79107080011.pvu.Z
1979/04/17:09:49:59  http://dods.gso.uri.edu/cgi-bin/nph-dsp/avhrr/1979/4/t79107094959.pvu.Z
1979/04/17:19:28:33  http://dods.gso.uri.edu/cgi-bin/nph-dsp/avhrr/1979/4/t79107192833.pvu.Z
1979/04/18:07:50:34  http://dods.gso.uri.edu/cgi-bin/nph-dsp/avhrr/1979/4/t79108075034.pvu.Z
1979/04/18:20:57:43  http://dods.gso.uri.edu/cgi-bin/nph-dsp/avhrr/1979/4/t79108205743.pvu.Z
```

Now you need to create a *format file* that describes the layout of the data file to the FreeForm server. The first row of the format section should read `ASCII_input_data`, and be followed by the name of the Sequence. (You choose it, it doesn't really matter what it is.)

Make a format file to match.

After the first line, each row of the format file describes another data value, and the columns that contain it. The indexes start at one, and are inclusive. Here is the format file for the AVHRR archive shown above:

```
ASCII_input_data "URI_Avhrr"
year     1 4    int32 0
month    6 7    int32 0
day      9 10   int32 0
hours    12 13  int32 0
minutes  15 16  int32 0
seconds  18 19  int32 0
DODS_URL 21 92  text  0
```

The `year` variable is in columns one through four, the `month` variable in columns six and seven and so on. The data type is also indicated in the format table. The last column in the format table is for scaling. All numeric data are scaled by the given power of ten.

NOTE: The FreeForm format is quite flexible, and can handle many more cases than the ones shown. *The DODS Freeform ND Server Manual* contains examples and complete descriptions of all the FreeForm capabilities, as well as tools you can use to check your format files. Refer to that book for more information about the server or writing format files.

Install them. You're done.

Once you've created a file containing your data files, and created a format file describing them, you need only find a place to park them. Give the catalog file some name ending in `.dat`, and the format file the *same name*, but ending in `.fmt`. Put them both in the same directory, somewhere in your document root directory tree, and test them by requesting a list of URLs.

The URI archive shown above is sampled in the *The DODS Quick Start Guide*. That book explains how to construct a *constraint expression* , and gives examples of them for use with a catalog server.

9

Building DODS Data Servers

Though servers are included in the DODS core software, some users may wish to write their own DODS data servers. The architecture of the `httpd` server and the DODS core software make this a relatively simple task.

There are a few occasions when it makes sense to build your own server.

A user may wish to write his or her own DODS server for any or all of the following reasons:

- The data to be served may be stored in a format not compatible with one of the existing DODS servers.
- The data may be arranged in a fashion that allows a user to optimize the access of those data by rewriting the service programs.
- The user may wish to provide ancillary data to DODS clients not anticipated by the writers of the servers available.

The design of the DODS library makes the task a relatively simple one for a programmer already familiar with the data access API to be used. The library provides a complete set of classes with most of the hard parts done already: evaluation of constraint expressions, network connections, and so on. You can create working sub-classes simply by adding the read methods which read data from the local disk.

Also, though the servers provided with the DODS core software are written in C++, they may be written in any language from which the DODS libraries may be called. There is also a Java version of the DODS class libraries.

Once it is invoked, a CGI program scoops up whatever input is going to the standard input stream of the Web server (`httpd`) that invoked it. Further, the standard output of the CGI is sent directly back to the requesting client. This means that the CGI program itself need only read its input from standard input and write its output to standard output.

Most of the task of writing a server, then, consists of reading the data with the data access API and loading it into the DODS classes. Methods defined for each class make it simple to output the data so that it may be sent back to the requesting client.

Refer to *The DODS Toolkit Programmer's Guide* for specific information about the classes and the facilities of the DODS core software, and instructions about how to write a new server.

A

Getting the DODS Software

Get the software from DODS Home page. If you can, it is usually advisable to download the pre-compiled binaries. If you do so, install this way:

Use the pre-compiled binaries if you can.

❶ Unpack tar files.

❷ Inspect the DODS tree that is created, look in the `/etc` directory to see what's there, and then move to the installation instructions in Section 2 on page 15.

Depending on your version of `tar`, you can use one of the following commands to unpack your software:

```
tar xzf DODS-...tar.gz
gzip -dc DODS-...tar.gz | tar xf -
```

If you can't use the pre-compiled binaries, download the source code from DODS Home page. You will need a few different files, depending on the server(s) you want to install. The "Download Software" page has instructions specifying which files to use.

NOTE: In case you missed the directions to do so on the web page, don't forget to download an unpack the file containing the DODS Server Tools. These are a collection of utilities and Perl scripts used by all the different DODS servers, and is essential to making your server go. Unpack this file from the same directory you unpack the others, and you'll wind up with a full-equipped DODS home tree.

When you unpack the files, they will all go into the same DODS tree. Go to the root of that tree, and type:

```
./configure
make World
```

This should start the compilation of all the necessary libraries, and the DODS source code.

When the compilation completes, inspect the DODS tree that is created, look in the `/etc` directory to see what's there, and then move to the installation instructions in Section 2 on page 15.

Index

Symbols

A

B

C

D

E

F

G

H

I

J

W