OPeNDAP Server Installation Guide
2006-08-31

Tom Sgouros James Gallagher Nathan Potter

Revision: 14459

Contents

List of Figures

List of Tables

1 The OPeNDAP Server

The OPeNDAP software 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. The OPeNDAP software implements the Data Access Protocol[1] (DAP), version 2. A *DAP server* is a program that sends data in a standard transmission format to a client that has requested it. A *DAP client* is a program, running on a networked computer somewhere in the world, that requests and receives data. A client can be a specialized DAP client, or a standard web browser.

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

Nothing limits the use of the DAP to science data; its framework will support many different data types. But the DAP has facilities for accommodating large arrays, relational tables, irregular grids, and many other otherwise anomalous data types. DAP 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.

The OPeNDAP DAP server is just an ordinary WWW server (`httpd`) equipped with a CGI program that enable it to respond to requests for data from DAP 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 each different one of them.) Once you have one installed, this guide will explain how to use it to serve data using the OPeNDAP server for the DAP.

The OPeNDAP DAP 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 OPeNDAP server's architecture, you need only understand the following:

[1] http://spg.gsfc.nasa.gov/rfc/004

- 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 DAP 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 way the server is configured depends on the storage format of the data you intend to serve. The OPeNDAP server supports a variety of storage formats, including NetCDF, HDF4, and DSP. The server 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

The OPeNDAP 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 OPeNDAP 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.

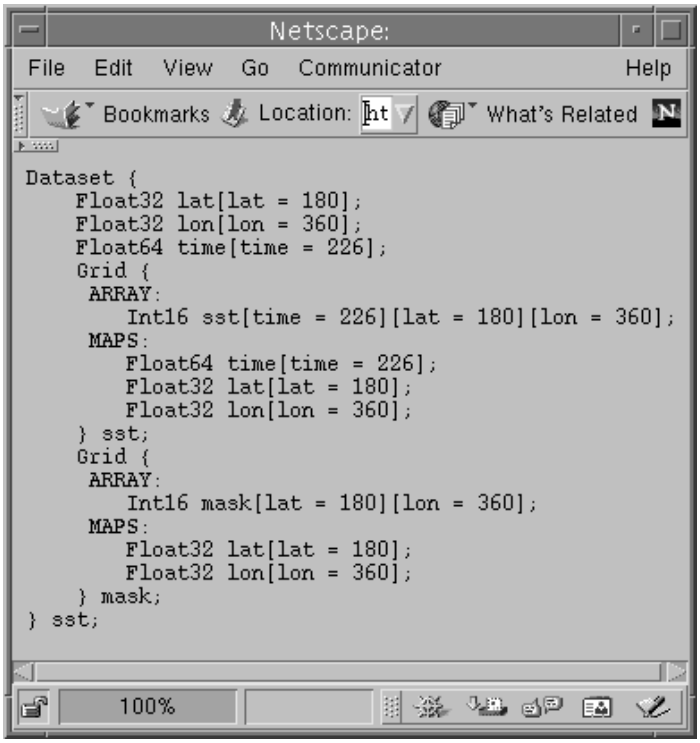

Figure 1.1: A DDS (One of the basic response types defined by the DAP, version 2)(`sst.mnmean.nc.dds`)

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

Other metadata is helpful, too.

After getting the *metadata* (defined, for DAP purposes, as the contents of the DAS and DDS), the DAP 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 DAP 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/OPeNDAP Quick Start Guide shows how to look at all the services provided by the OPeNDAP Server.

In addition to the three basic message types (DAS, DDS, and data), the OPeNDAPserver 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.

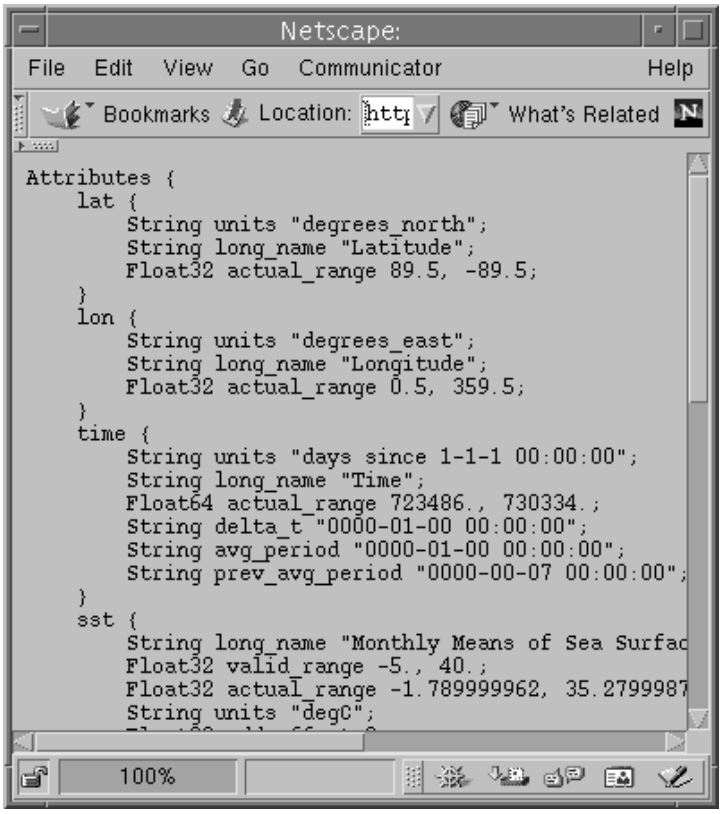

Figure 1.2: A DAS (`sst.mnmean.nc.das`)

1.2 More Explanation of How It Works

To understand the operation of the OPeNDAP 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 DAP request URL such as the following:

A description of the different URL parts.

```
http://test.opendap.org/opendap/nph-dods/data/nc/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://test.opendap.org/opendap/nph-dods/data/nc/fnoc1.nc.dds
```

A DAP 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, `test.opendap.org`. Figure 1.3 shows these first steps. The client makes an internet connection to the `test.opendap.org` machine, and the `httpd` daemon executes the dispatch script (`opendap/nph-dods`) and forwards it the remaining parts of the URL it had received.[2] (In this case, that would be `data/nc/fnoc1.nc.dods`.) DAP requests are "GET" requests, not "POST" requests, so all the information forwarded is in the URL.[3]

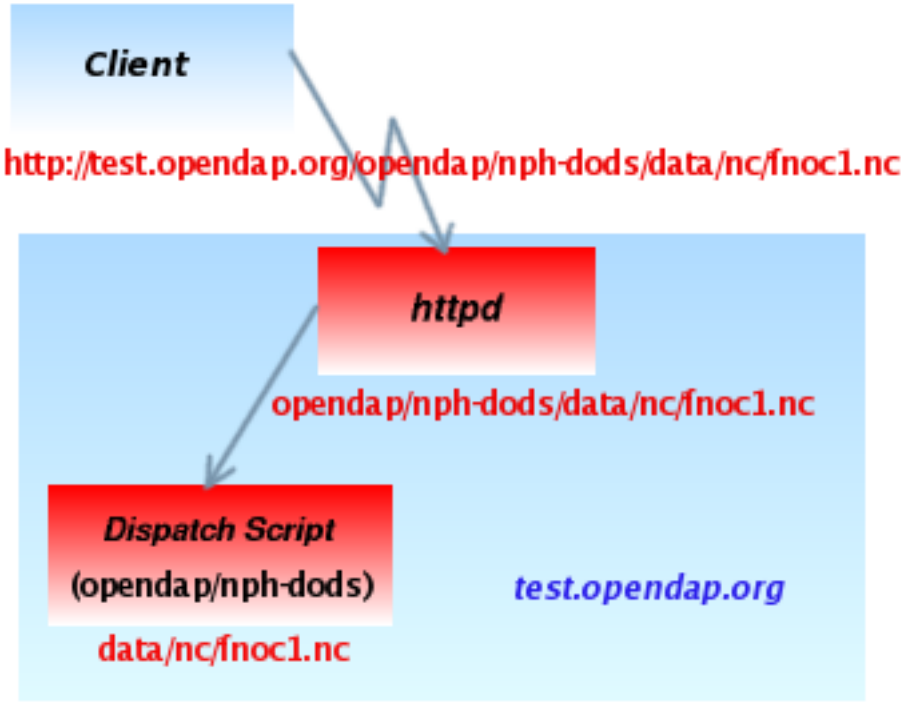

Figure 1.3: The Architecture of the OPeNDAP Server, part I.

Figure 1.4 illustrates what happens next. Sitting in the CGI directory (here called `opendap`) 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.

[2]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.

[3]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` and OPeNDAP will use this name for all of its version 3.x releases. Earlier releases of DODS used different 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 `dap_nc_handler` 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* .[4] 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 OPeNDAP software will look for its data relative to the document root tree. On the `test.opendap.org` server, for example, the nph-dods CGI program is executed native to the directory `/usr/local/share/dap-server/`, but the document root directory is `/var/www/html/`. 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.

```
/var/www/html/data/nc/
```

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://test.opendap.org/opendap/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 (`opendap`), as well as the name and location of the working directory used by the CGI programs, are local

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

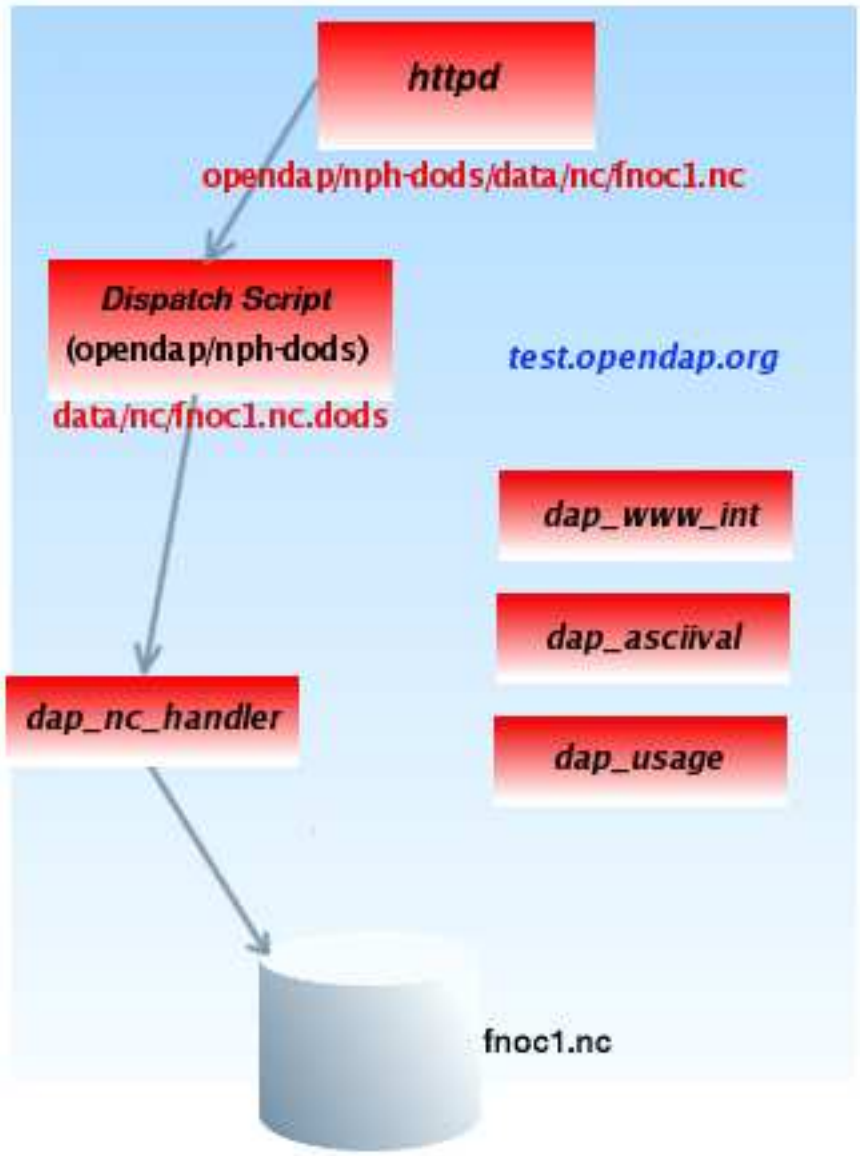

Figure 1.4: The Architecture of the OPeNDAP Server, part II.

configuration details of the particular web server in use. The location of the 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 the server.

When the server gets a DAP 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 of the server. The dispatch script is called `nph-dods`. (Though see note on page 12.) 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: Services, with their suffixes and helper programs. In the OPeNDAP server, different handlers are used for each supported data access format, e.g. `nc` for netCDF, `jg` for JGOFS, and so on.

Service	Suffix	Helper Program
Data Attribute	`.das`	`dap_nc_handler`
Data Descriptor	`.dds`	`dap_nc_handler`
DODS Data	`.dods`	`dap_nc_handler`
ASCII Data	`.asc` or `.ascii`	`dap_asciival`
Information	`.info`	`dap_usage`, see Chapter 7.
WWW Interface	`.html`	`dap_www_int`
Version	`.ver`	None
Compression	None	`deflate`
Help	Anything else	None

The service program invoked, and its arguments, 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 (`dap_nc_handler`) is started using the `-O das` argument. Similarly, the extension '.dds' will cause the `dap_nc_handler` service to be run with the `-O dds` argument, and so on.

On the client side, when using a DAP 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 DAP client may produce three different requests for information. These manipulations happen within the DAP client software, and the user need never be aware of them.

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

1.4 Choosing a Data Handler

There are a variety of data format handlers available for the OPeNDAP data server, each designed to handle a different data storage format. Data handlers from OPeNDAP exist to serve data stored in the netCDF, HDF and HDF-EOS, and DSP storage formats (other groups have built data handlers for other 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 server, written in Java, for serving data stored in relational databases. See Chapter 4 for more information about installing that software. See the DRDS Download page 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.[5]

There are two data handlers 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 data handlers 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 handler is somewhat easier to set up, and the JGOFS handler is more flexible. A key difference is that the JGOFS handler can process data contained in several different data files. (The FreeForm handler can, as well, but, being slightly less flexible, it may require the files to be rearranged or renamed.)

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

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

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 OPeNDAP 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 OPeNDAP; 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 the OPeNDAP Server

Most of the task of installing the OPeNDAP 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 the OPeNDAP Server.

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 server's base software and data handlers (these are the components of the server) and data to be served.

❶ **Install the web server.**

❷ **Download and install the OPeNDAP server base software.**

❸ **Choose one or more data handlers, download and install.**

❹ **Configure your data.**

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

❶ Install the web server to be used. This is not an OPeNDAP 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 web browser, 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 58.) See Chapter 6 for more ways to check if the web server is working.

❷ Download and install the OPeNDAP server base software. It is usually easiest to use the pre-compiled binary distributions. Look at the OPeNDAP Home page to get the software. See Appendix A on page 69 for more information.

Testing: The software should all be installed in three directories: 1) The Perl software in `/usr/local/share/dap-server`; 2) The CGI program in `/usr/local/share/dap-server-cgi`; and 3) The binary programs `dap_usage`, `dap_asciival` and `dap_www_int` in `/usr/local/bin`. If you build the software from sources you can install the server software in a root that is different from `/usr/local`.

❸ Once the server software has been installed, you'll need to install one or more *data handlers* . OPeNDAP provides data handlers for NetCDF, HDF4 (and HDF-EOS), DSP, JGOFS, and FreeForm. You need to download these separately and follow their installation instructions.

❹ Next you will need to configure the server's CGI so that it can find the handlers. During this step a handful of configuration parameters must be set. Locate the file named `dap-server.rc` in

`/usr/local/share/dap-server-cgi`. See Section 2.2 on page 20 for instructions about how to configure this file.

Testing: When the server is working, you should get a response to a *version* request, where you query the 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 server. 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 3.5 or later, you also need to make sure that the dispatch configuration file `dap-server.rc` is also copied into the directory with the CGI and correctly protected. See Section 2.2 on page 20 for instructions about how to configure this file. Note that earlier servers (from version 3.2 through 3.4 used a configuration file named `dods.in` or `dods.rc`).

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

❺ 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 OPeNDAP server uses the HTML::Parser Perl module. You should have installed this prior to installing the dap-server package.

The CGI configuration of a web server is dependent on the particular web server you use. 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/ /var/www/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 `/var/www/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

Starting with version 3.5, the OPeNDAP server makes much more extensive use of its configuration file. The `dap-server.rc` configuration file is used to tailor the server to your site. The file contains a handful of parameters plus the mappings between different data sources (typically files, although that doesn't have to be the case) and hander-programs. The format of the configuration file is:

```
<parameter> <value> ... <value>
```

The configuration file is line-oriented, with each parameter appearing on its own line. Blank lines are ignored and the # character is used to begin comment lines. Comments have to appear on lines by themselves.

The parameters recognized are:

- data_root

  ```
  data_root <path>
  ```

  Define this if you do not want the server to assume data are located under the web server's DocumentRoot. The value `<path>` should be the fully qualified path to the directory which you want to use as the root of your data tree. For example, we have a collection of netcdf, hdf, *et c.*, files that we store in directories named `/usr/local/test_data/data/nc`, `/usr/local/test_data/data/hdf`, *et c.*, and we set `data_root` to `/usr/local/test_data`. The value of `<path>` should not end in a slash.

  NOTE: The OPeNDAP server's directory browsing functions do not work when using the `data_root` option. They do still work when locating data under DocumentRoot. Also, it's not an absolute requirement that this path be a real directory or that your data are in 'files.' Data can reside in a relational database, for example, and in that case the base software will use `<path>` as a prefix to the path part of the URL it receives from the client.

- timeout

  ```
  timeout <seconds>
  ```

  This sets the OPeNDAP server timeout value, in seconds. This is different from the httpd timeout. OPeNDAP servers run independently of httpd once the initial work of httpd is complete. Setting `timeout` ensures that your OPeNDAP server does not continue indefinitely if something goes wrong (*i.e.*, a user makes a huge request to a database).

  Default is 0 which means no time out.

- cache_dir

  ```
  cache_dir <directory>
  ```

  When data files are stored in a compressed format such as gzip or UNIX compress, the OPeNDAP server first decompresses them and then serves the decompressed file. The files are cached as they are decompressed. This parameter tells the server where to put that cache.

  Default: `/usr/tmp`

- cache_size

  ```
  cache_size <size in MB>
  ```

  How much space can the cached files occupy? This value is given in MegaBytes. When the total size of all the decompressed files exceeds this value, the oldest remaining file will be removed until the size drops below the parameter value. If you are serving large files, make *sure* this value is at least as large as the largest file.

- maintainer

  ```
  maintainer <email address>
  ```

  The email address of the person responsible for this server. This email address will be included in many error messages returned by the server.

  Default: `support@unidata.ucar.edu`

- curl

  ```
  curl <path>
  ```

  This parameter is used to set the path to the curl executable. The curl command line tool is used to dereference URLs when the server needs to do so. In some cases the curl executable might not be found by the CGI. This can be a source of considerable confusion because a CGI program run from a web daemon uses a very restricted PATH environment variable, much more restricted than a typical user's PATH. Thus, even if you, as the server installer have curl on your PATH, nph-dods may not be able to find the program unless you tell it exactly where to look.

- exclude

  ```
  exclude <handler> ... <handler>
  ```

  This is a list of handlers whose regular expressions should not be used when building the HTML form interface for this server. In general, this list should be empty. However, if you have a handler that is bound to a regular expression that is very general (such as `.*` which will match all files), then you should list that handler here, enclosing the name in double quotes. See the next item about the 'handler' parameter. Default: No handlers are excluded.

- handler

  ```
  handler <regular expression> <handler name>
  ```

  The handler parameter is used to match data sources with particular handler programs used by the server. In a typical OPeNDAP server setup, the data sources are files and the regular expressions choose handlers based on the data file's extension. However, this need not be that case. The OPeNDAP server actually matches the entire pathname of the data source when searching for the correct handler to use.

  Here are the values assigned to the `handler` parameter in the default `dap-server.rc` file:

  ```
  # Look for common file extensions.
  handler .*\.(HDF|hdf|EOS|eos)(.Z|.gz)*$ /usr/local/bin/dap_hdf_handler%$
  handler .*\.(NC|nc|cdf|CDF)$ /usr/local/bin/dap_nc_handler %$
  handler .*\.(dat|bin)$ /usr/local/bin/dap_ff_handler %$
  handler .*\.(pvu)(.Z|.gz)*$ /usr/local/bin/dap_dsp_handler %$

  # For JGOFS datasets, match either the dataset name or the absence of an
  # extension. The later case is sort of risky, but if you have lots of JGOFS
  # datasets it might be appealing.
  handler .*/test$ /usr/local/bin/dap_jg_handler%$
  # handler .*/[^/]+$ /usr/local/bin/dap_jg_handler%$
  ```

  Consider a URL like this:

  ```
  http://test.opendap.org/opendap/nph-dods/data/nc/fnoc1.nc
  ```

  When the `nph-dods` script is executed, the "file name" part of the URL is `/data/nc/fnoc1.nc`. Testing this against the default initialization file matches the second line, which indicates that the `dap_nc_handler` (NetCDF) data handler should be used to process this request. The request is then dispatched to the handler for processing.

NOTE: The default configuration file used to be set up so that files without extensions are handled by the JGOFS data handler. This caused some problems for sites where data files did not always use common extensions for data files. If you need to serve many JGOFS data files, then uncomment this line of write special regular expressions for the JGOFS data sources.

"Regular expressions", advanced pattern-matching languages, are a powerful feature of Perl and many other computer languages. Powerful enough, in fact, to warrant at least one book about them (Mastering Regular Expressions by Jeffrey Friedl, O'Reilly, 1997).

(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 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:

```
handler .* /usr/local/bin/dap_hdf_handler
```

The `.*` pattern matches all possible patterns (the `.` matches a single character and `*` matches zero or more occurrences of the previous character or *metacharacter*), 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:

```
handler \/dsp_data\/.*$ /usr/local/bin/dap_dsp_handler %$
handler .*/[^/]+$ /usr/local/bin/dap_jg_handler %$
```

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 processed with the DSP handler, and everything else will be sent to the JGOFS handler.

2.3 Compression

Data compression can be a confusing subject for people installing the OPeNDAP 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 OPeNDAP server can process compressed data, even when the native API which should be used to read the data cannot process it. The nph-dods CGI program will decompress the file(s) and cache them in a temporary directory. 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.

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 `dap-server.rc` configuration file to change the value of `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. Edit `dap-server.rc` and change the value of `cache_size` to the maximum size that you want the contents of this directory to be.

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

[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`.

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 Enabling Transmission of Compressed Data

To save communication bandwidth, the OPeNDAP server can send compressed data to certain clients who are equipped to handle it. When making a request for data, a client can signal the server that data may be sent in a compressed form. 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 OPeNDAP servers can handle data compression for transmission if the `deflate` program is installed on the `$PATH` used by the web server. In version 3.5 of libdap, `deflate` is installed in `$prefix/bin` (libdap is required to run the server).

2.4 Security

Many data providers ask whether the OPeNDAP server can be configured to limit access to a particular file or sets of files. The answer is that it is as flexible as the http server you use to implement it. Clients built using the OPeNDAP DAP2 library are capable of prompting 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@test.opendap.org/nph-dods/...
```

Depending on the specific web server, you can restrict access to the CGI programs or to the individual data files or directories.

By using the web server's authorization software we are ensuring that the security checks used by the server have been tested by many many sites. In addition, WWW servers already support a very full set of security features and many system administrators are comfortable with, and confidant in, them. The tradeoff with using the web server's security system for our servers is that two security settings must be made for each group of data to be configured and more than one copy of the nph-dods CGI program and dap-server.rc configuration file may be needed even if you're serving only one type of data.

Because the security features rely almost entirely on the host machine's WWW server, the steps required to install a secure the server will vary depending on the WWW server used. Thus, before installing a secure server, check over your WWW server's documentation to make sure it provides the following security features: Access limits to files in the document root on a per user and/or per machine basis, and; The ability to place CGI scripts in protected directories.

There are two levels of security which the OPeNDAP server supports: Domain restrictions and user restrictions. In conjunction with a WWW server, access to the OPeNDAP server can be limited to a specific group of users (authenticated by password), specific machine(s) or a group of machines within a given domain or domains.

NOTE: Because security features are used to protect sensitive or otherwise important information, once set-up they should be tested until you are comfortable that they work. You should try accessing from at least one machine that is not allowed to access your data. If you would like, we will help you evaluate your set-up.

It is important to distinguish securing a server from securing data. If data are served, then those data may also be accessible through a web browser. If so the data themselves need to be stored in directories that have limited access. If all data access will take place through the server this limitation can exclude all access

except the local machine. This is the case because some the server's directory function requires being able to read the data through the local host's web server.

It bears repeating: If you're serving sensitive information with the OPeNDAP server and those data are located under the WWW daemon's DocumentRoot, that information is accessible two ways: via the OPeNDAP server and through the WWW server. You need to make sure *both* are protected.

NOTE: With version 3.5 of the OPeNDAP server, you can use the `data_root` parameter in the `dap-server.rc` configuration file to serve data that are not accessible using your WWW server. This simplifies securing the data but has the drawback that the directory response is not supported by the OPeNDAP server.

2.4.1 About serving both limited- and open-access data from the same server

In the past it was possible to install two or more OPeNDAP servers on a computer and assign different protections to each one. However, in practice this has proven to be very hard to configure correctly. In many cases where this feature was used, a secure server was setup up for one group of data while an open server was set up for another. It was often the case that all the data were accessible using the open server! Thus, if you need to secure some data, it is best to host all the sensitive information on one machine and put other data on a second machine with an open-access server. If you must run two or more servers from the same physical host, we suggest that you configure your web server to see two (or more) virtual hosts. This will provide the needed separation between the groups of data.

2.4.2 Using a secure opendap server

Using a secure sever is transparent if the server is configured to allow access based on hosts or domains. Give the URL to a client; the server will respond by answering the request if allowed or with an error message if access is not allowed.

Accessing a server which requires password authentication is a little different and varies depending on the type of client being used. All OPeNDAP clients support passing the authentication information along with the URL. To do this add `<username>:<password>@` before the machine name in a URL. For example, suppose I have a secure server set up on 'test.opendap.org' and the user 'guest' is allowed access with the password 'demo'. A URL for that server would start out:

```
http://guest:demo@test.opendap.org/...
```

For example,

```
http://guest:demo@test.opendap.org/secure/nph-dods/sdata/nc/fnoc1.nc.info
```

will return the info on the data set fnoc1.nc from a secure server. You cannot access the data without including the username and password `guest` and `demo`.

Some clients will pop up a dialog box and prompt for the username and password. Netscape, and some other web browsers, for example, will do this. Similarly, some DODS clients may also popup a dialog.

2.4.3 Configuring a server

In the following I'll use the Apache 1.3.12 server as an example[2] and describe how to install a server which limits access to a set of users. While this example uses the Apache server, it should be simple to perform the equivalent steps for any other WWW server that supports the set of required security features.

Create a directory for the server

To limit access to a dataset to particular machine, begin by creating a special directory for the server. This maybe either an additional CGI bin directory or a directory within the web server's document root. In this example, I chose the latter.

```
cd /var/www/html/
mkdir secure
```

Establish access limitations for that directory

Establish the access limitations for this directory. For the Apache server, this is done either by adding lines to the server's `httpd.conf` file or by using a per-directory access control file. Note: The use of per-directory access control files is a configurable feature of the Apache server; look in the server's `httpd.conf file` for the value of the AccessFileName resource.

I modified Apache's `httpd.conf` file so that it contains the following:

[2]. . . also tested on Apache 2.0.40, 07/25/03 jhrg

```
# Only valid users can use the server in 'secure'. 7/6/2000 jhrg
<Directory /var/www/html/secure>
    Options ExecCGI Indexes FollowSymLinks

    Order deny,allow
    Deny from all
    # ALLOW SERVER (IP OF SERVER) MACHINE TO REQUEST DATA ITSELF
    Allow from __YOUR_SERVER_HERE__
    Require valid-user
    # ALL VISITORS NEED USERNAME AND PASS BUT NOT SERVER
    Satisfy any

    AuthType Basic
    AuthUserFile /etc/httpd/conf/htpasswd.users
    AuthGroupFile /etc/httpd/conf/htpasswd.groups
    AuthName "Secure server"
</Directory>

# Protect the directory used to hold the secure data.
<Directory /var/www/html/sdata>
    Options Indexes

    Order deny,allow
    Deny from all
    # ALLOW SERVER (IP OF SERVER) MACHINE TO REQUEST DATA ITSELF
    Allow from __YOUR_SERVER_HERE__
    Require valid-user
    # ALL VISITORS NEED USERNAME AND PASS BUT NOT SERVER
    Satisfy any

    AuthType Basic
    AuthUserFile /etc/httpd/conf/htpasswd.users
    AuthGroupFile /etc/httpd/conf/htpasswd.groups
    AuthName "Secure data"
</Directory>
```

and

```
ScriptAlias /secure/ "/var/www/html/secure/"
```

The first group of lines establishes the options allowed for the `secure` directory, including that it can contain CGI programs. The lines following that establish that only users in the Apache password file can access the contents of the directory, with the exception that this server is allowed to access the directory without authentication. This last bit is important because OPeNDAP servers sometimes make requests to themselves (e.g., when generating the directory response) but don't pass on the authentication information.[3]

The ScriptAlias line tells Apache that executable files in the directory are CGIs. You can also do this by renaming the `nph-dods` script to `nph-dods.cgi` and

[3]Brock Murch <bmurch@marine.usf.edu> worked out some thorny configuration details for securing the Apache/DODS/OPeNDAP combination.

making sure `httpd.conf` contains the line:

```
AddHandler cgi-script .cgi
```

The AuthType directive selects the type of authentication used. Apache 2.0 supports 'Basic' and 'Digest' while other servers may also support GSS-Negotiate and NTLM. Version 3.4 onward of the DAP software supports all these authentication schemes, although only Basic and Digest have been thoroughly tested. Configuration of Apache 2.0 for Digest authentication is slightly different then for Basic authentication, but is explained well in Apache's on line documentation.

Copy the server into the new directory

Copy the CGI program (`nph-dods`) and the server configuration file to the newly created directory. Note that if you're using the extension '.cgi' to tell Apache that `nph-dods` is a CGI you must rename it to `nph-dods.cgi`. If you forget to do that then you will get a Not Found (404) error from the server and debugging information generated by the server won't appear in Apache's `error_log` even if it has been turned on.

You are done.

2.4.4 Tips

Here are some tips on setting up secure servers:

- Using the per-directory limit files makes changing limits easier since the server reads those every time it accesses the directory, while changes made to the `httpd.conf` file are not read until the server is restarted or sent the HUP signal. However, using `httpd.conf` for your security configuration seems more straightforward since all the information is in one place.
- If the protections are set up so that it is impossible for the server host to access the data and/or the OPeNDAP server itself, then an infinite loop can result. This can be frustrating to debug, but if you see that accesses generate an endless series of entries in the access˙log file, it is likely that is the problem. Make sure that you have `allow from <server host name>` set for both the directory that holds the OPeNDAP server and that holds the data. Also make sure that the server's name is set to the full name of the host.
- Configuring a secure server can be frustrating if you're testing the server using a web browser that remembers passwords. You can turn this feature off in some browsers. Also, the getdap tool supplied with OPeNDAP's libdap (which is required to build the server) can be useful to test the server since it will not remember passwords between runs.

3
Installing Your Data

After installing the server, the data to be provided must be put in some location where it may be served to clients. The server navigates a directory tree rooted in the web server's *document root* directory. This is the directory (often called `htdocs` or the `DocumentRoot`) 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/page.html
```

It will look for the file `page.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.

> **NOTE:** You can also use the server's `data_root` configuration parameter to tell the server where to look for data. The value of `data_root` is used as a prefix for the location of data. Using this you can configure a server for data that cannot also accessed using the web. The only drawback to serving data this way is that the directory browsing functions of the OPeNDAP server will not work.

Fortunately, this is not hard.

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.

3.1 Special Instructions for the NetCDF, HDF, and DSP handlers

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

> **NOTE:** The `data_root` parameter can be substituted for the web server's `DocumentRoot` in the discussion that follows.

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 OPeNDAP/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

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 scientific datasets served by this data server, regardless of whether they are simple single file or complex multi-file datasets.

The JGOFS API reads "objects", not files.

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 OPeNDAP/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 OPeNDAP/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 OPeNDAP/JGOFS servers can be configured to use the current installation.

Sample data is included for testing purposes.

Included with the OPeNDAP/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 OPeNDAP/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 server to determine the location of the objects dictionary and access method directory. Initially, the definition of these variables is set to the OPeNDAP server's directory (i.e., the directory that contains `nph-dods`).

❷ If the shell variables `JGOFS_OBJECT` and `JGOFS_METHOD` are not defined, the OPeNDAP/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 OPeNDAP/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 OPeNDAP server, then specifying the `JGOFS_OBJECT` and `JGOFS_METHOD` variables in the `nph-dods` script will override your existing JGOFS setup.

To use the `.objects` dictionary and `def` access method supplied with the OPeNDAP/JGOFS server, edit the `nph-dods` script. In the `nph-dods` 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. Don't forget to 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 OPeNDAP/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. 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 `$prefix/share/dap-server/jgofs_data` directory.[1] You are free to move

[1] '$prefix' is '/usr/local/' unless you built the server from source code and specified a different root directory for its installation.

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-dods/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-dods/test0.asc
```

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

3.2.2 Attribute Data for OPeNDAP/JGOFS

The JGOFS data access API does not support a wide variety of attribute data. The OPeNDAP server will look for a file that contains *ancillary data* file named *object*`.das`, in the same directory which contains the JGOFS methods (i.e., the directory referenced by $JGOFS˙METHODS).

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

NOTE: In the past the server looked in the same directory as the data.

For example, suppose that you have installed the OPeNDAP/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:

```
$JGOFS_METHODS/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 OPeNDAP/JGOFS Data

The OPeNDAP server allows JGOFS to support new data types.

The JGOFS data access system returns string data when using the default (`def`) method. That is, all data is returned to the client as character strings. The OPeNDAP/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 35) 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 OPeNDAP/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 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 OPeNDAP/JGOFS Server

JGOFS offers new opportunities for limiting data access.

Generally speaking, it is straightforward to configure OPeNDAP servers with limited access. However, the data dictionary structure of the JGOFS data access API provides an additional 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 restricted.

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.

3.3 Special Instructions for the FreeForm Server

The OPeNDAP FreeForm data handler 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 OPeNDAP Relational Database Server

The OPeNDAP 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 1.4.[1]

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.

[1] We have reports that our Java DAP libraries work with versions as old as 1.2. jhrg 10/13/05.

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 OPeNDAP Home pageor the OPeNDAP 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`. If you have installed version 1.1.7 or newer, then the DODS Test Servlet (DTS) should be running after you get your servlet engine restarted. You should be able to access it at: `http://yourserver:port/dods/dts/`

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 running.

4.1 General Servlet Configuration

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 servlet installation, without taxing the JDBC drivers or your DBMS. We'll get to that in a moment, first let's review some aspects of the Servlet configuration that are common to all DODS servlets.

Use the test server to test your installation

The OPeNDAP servlets get their configuration from the servlet's `web.xml` file. The default location of the `web.xml` file is (at least in Tomcat 4.1) in `$TOMCAT_HOME/webapps/dods/WEB-INF` (Obviously if the dods directory gets renamed then things change accordingly.)

Each instance of a servlet running in the opendap servlet area needs an entry in the `web.xml` file. Multiple instances of a servlet and/or several different servlets can be configured in the one `web.xml` file. For instance you can have a DTS and 2 instances of the DRDS configured through one `web.xml` file.

Each instance of a servlet needs a unique name which is specified inside a `<servlet>` element in the `web.xml` file using the `<servlet-name>` tag. This is a name of convenience, for example if you where serving data from an ARGOS satellite you might call that servlet argos.

Additionally each instance of a `<servlet>` must specify which Java class contains the actual servlet to run. This is done in the `<servlet-class>` element. For example the DRDS's class name is `opendap.servers.sql.drds`

Here is a syntax example combining the two previous example values:

```
<servlet>
    <servlet-name>argos</servlet-name>
    <servlet-class>opendap.servers.sql.drds</servlet-name>
    .
    .
    .
</servlet>
```

This servlet could then be accessed as:

```
http://hostname/opendap/servlet/argos
```

You may also add to the end of the web.xml file a set of `<servlet-mapping>` elements. These allow you to abbreviate the URL or the servlet. By placing the servlet mappings:

```
 <servlet-mapping>
    <servlet-name>argos</servlet-name>
    <url-pattern>/argos</url-pattern>
</servlet-mapping>
<servlet-mapping>
    <servlet-name>argos</servlet-name>
    <url-pattern>/argos/*</url-pattern>
</servlet-mapping>
```

At the end of the web.xml file our previous example changes it's URL to:

```
http://hostname/opendap/argos
```

Eliminating the need for the word servlet in the URL. For more on the `<servlet-mapping>` element see the Jakarta-Tomcat documentation.

4.1.1 <init-param> Elements

The OPeNDAP servlets use `<init-param>` elements inside of each `<servlet>` element to get specific configuration information.

`<init-param>`'s common to all OPeNDAP servlets are:

- `DebugOn` - This controls output to the terminal from which the servlet engine was launched. The value is a list of flags that turn on debugging instrumentation in different parts of the code. Common values are: `showRequest`, `showResponse`, `showException`, and `probeRequest`. Other debugging values that are specific to each servlet should be documented in each servlets javadoc documentation.

  Example:

  ```
  <init-param>
      <param-name>DebugOn</param-name>
       <param-value>showRequest showResponse</param-value>
  </init-param>
  ```

  Default: If this parameter is not set, or the value field is empty then debugging instrumentation is not turned on.

- `DDScache` - This is should be set to the directory containing the DDS files for the data sets used by the servlet. Some servlets have been developed that do not use DDS's that are cached on the disk, however the default behavior is for the servlet to load DDS images from disk.

  Example:

  ```
  <init-param>
      <param-name>DDScache</param-name>
      <param-value>/usr/OPeNDAP/sdds-testsuite/dds/</param-value>
  </init-param>
  ```

Default: If this parameter is not set (does not appear in as an `<init-param>`) then it is set to `$TOMCATHOME/webapps/opendap/datasets/servlet-name/dds` (where servlet-name is the same as the name specified in the `<servlet-name>` element of the servlet configuration)

- `DAScache` - This is should be set to the directory containing the DAS files for the data sets used by the servlet. Some servlets have been developed that do not use DAS's that are cached on the disk, however the default behavior is for the servlet to load DAS images from disk. Example:

```
<init-param>
    <param-name>DAScache</param-name>
    <param-value>/usr/OPeNDAP/sdds-testsuite/das/</param-value>
</init-param>
```

Default: If this parameter is not set (does not appear in as an `<init-param>`) then it is set to `$TOMCATHOME/webapps/opendap/datasets/servlet-name/das` (where servlet-name is the same as the name specified in the `<servlet-name>` element of the servlet configuration) .

- `INFOcache` - This is should be set to the directory containing the files used by the ".info" service for the servlet. This directory should contain any data set specific "over-ride" files (see below), any data set specific additional information files (see below), and any servlet specific information files(see below). Example:

```
<init-param>
    <param-name>INFOcache</param-name>
    <param-value>/usr/OPeNDAP/sdds-testsuite/info/</param-value>
</init-param>
```

Default: If this parameter is not set (does not appear in as an `<init-param>`) then it is set to `$TOMCATHOME/webapps/opendap/datasets/servlet-name/info` (where servlet-name is the same as the name specified in the `<servlet-name>` element of the servlet configuration)

4.2 DODS Test Server (DTS)

classname: `dods.servers.test.dts`

This servlet is primarily used to test the core code (in particular the DAP). This servlet will take any DDS in it's `DDScache` and populate it with invented data per client request. This allows the testing of unusual DDS structures. By default the DTS will look in these locations:

```
\$TOMCAT_HOME/webapps/dods/datasets/dts/dds
\$TOMCAT_HOME/webapps/dods/datasets/dts/das
\$TOMCAT_HOME/webapps/dods/datasets/dts/info
```

For files containing DDS, DAS, and ancillary information. These directories come full of test dataset descriptions and metadata in the distribution (1.1.7 or higher). If the `web.xml` file entry for the DTS contains entries for any of the `DDScache`, `DAScache`, or the `INFOcache` directories then the DTS will look there instead of in the defalut location(s).

4.2.1 DTS Configuration

The DTS comes configured to startup the first time the servlet's Web Archive File (WAR file) is unpacked by Tomcat. You may wish to change the length of the returned Sequences via the information below, but it is not neccessary.

`<init-param>`'s unique to the DTS:

- `SequenceLength` - This `<init-param>` sets the number of rows each Sequence returned by the DTS will have. Common values are typically small (5-100) for simple testing. If you are to testing client code against the DTS you may wish to use a large value (>50000) here to check the client's ability to handle large Sequences.

  Example:

  ```
  <init-param>
  <param-name>SequenceLength</param-name>
  <param-value>100</param-value>
  </init-param>
  ```

  Default: Is set to 5 if this parameter is not set (does not appear in as an `<init-param>`).

- `DebugOn` - There are no DTS specific values for this `<init-param>`.

Example of web.xml content:

```
<servlet>
<servlet-name>
dts
  </servlet-name>
<servlet-class>
dods.servers.test.dts
</servlet-class>
<init-param>
<param-name>DebugOn</param-name>
<param-value>showRequest showResponse </param-value>
</init-param>
<param-name>SequenceLength</param-name>
<param-value>100</param-value>
</init-param>
</servlet>
```

In this example SequenceLength gets set to 100.

4.3 DODS Relational Database Server (DRDS)

classname: `dods.servers.sql.drds`

This DODS server serves data from a relational database to DODS clients.

The DRDS uses the JDBC interface to connect to the database. This means that to use this server you will need to locate and install JDBC drivers for your database system. (This requirement is for the server only, the DODS clients that use the server need know nothing about JDBC or SQL). Here are some links that should lead you to the JDBC support pages for some of the more common RDBM's:

❶ MySQL JConnector Site:
http://www.mysql.com/products/connector/j/index.html

❷ PostgreSQL JDBC site: http://jdbc.postgresql.org/

Additional PostgreSQL JDBC docs:
http://developer.postgresql.org/docs/postgres/jdbc.html

❸ Oracle's SQLJ, JDBC, & JPublisher site:
http://www.oracle.com/technology/tech/java/sqlj˙jdbc/index.html

❹ Informix JDBC Drivers site:
http://www-306.ibm.com/software/data/informix/tools/jdbc/

Most modern database vendors usually provide JDBC drivers for their product. The glaring exception has been Microsoft, which isn't surprising as they have made no bones about wanting to kill Java. As of the release of MSSQL-Server 2000, Microsoft appears to be offering a JDBC driver for Win2000/XP systems. I have developed using MSSQL-Server for some time (still do actually.) and I have been using a purchased set of drivers from DataDirect Technologies (formerly known as Merant DataDirect). I am using their "SequeLink" product and it's been working great. Find it, and the Microsoft stuff, at the following links:

❶ Microsoft SQL-Server 2000 JDBC Drivers: http://www.microsoft.com/downloads/details.aspx?FamilyID=86212d54-8488-481d-b46b-af29bb18e1e5&DisplayLang=en

❷ Data Direct Technologies: http://www.datadirect.com/index.ssp

In this release the DRDS does not support SQL JOIN operations. Each database table must appear as a DODS Sequence data type in its own DDS file. If your data crosses multiple tables then you will need to make a "view" or a "virtual table" in the database in order to serve the data with the DRDS. This situation will improve in the next major revision. (I have an as yet un-implemented plan to allow the DRDS to support SQL JOIN operations.)

4.3.1 DRDS Datatype Translation

Since the DRDS is reading data from a relational database through a JDBC connection it is important to note that their are several layers of type translation happening in this:

Database -> JDBC -> Java -> DAP2

The Database types are the native types for the particular database that is being read from. The translation from Database->JDBC is handled before we get to the data (most likely by the JDBC Drivers). Our mapping of JDBC type to DAP2 types (the intermediate Java types happen in the process) can be seen in table 4.1. The mappings are handled in the read() method for each of the corresponding DAP2 data types.

4.3.2 DRDS Configuration

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.

Next modify your web.xml to include the ¡init-param¿ elements listed below. Look carefully at the example web.xml section provided below.

`<init-param>`'s unique to the DRDS:

- `JDBCdriver` - This must be set to the fully qualified Java class name of the JDBC drivers that the servlet is to use to make the JDBC connection to the DBMS. In my servlet that is using the Merant DataDirect JDBC drivers the class name of the JDBC driver is com.merant.sequelink.jdbc.SequeLinkDriver

  Example:

  ```
  <init-param>
      <param-name>JDBCDriver</param-name>
      <param-value>com.merant.sequelink.jdbc.SequeLinkDriver</param-value>
  </init-param>
  ```

  Default: This is a required `<init-param>`, there is no default value.

- `JDBCconnectionURL` - This is the connection URL (aka the connection string) that the DRDS is to use to connect to to the DBMS. This is usually defined by the developers of the JDBC driver. Read the documentation for the JDBC driver that you are using. It is not always easy to ascertain for a particular installation. For example, in my server that is using the Data Direct drivers the value is set to:

  `jdbc:sequelink://sugar.oce.orst.edu:19996`

  If you are stumped get in touch with support at `support@unidata.ucar.edu` or Nathan Potter `ndp@OPeNDAP.org`.

  Example:

  ```
  <init-param>
      <param-name>JDBCconnectionURL</param-name>
      <param-value>jdbc:sequelink://foogoo.oce.orst.edu:18796</param-value>
  </init-param>
  ```

  Default: This is a required `<init-param>`, there is no default value.

- `JDBCusername` - This is the user name for the DBMS that the JDBC connection will be made under. This is often set to ”guest”.

  Example:

  ```
  <init-param>
      <param-name>JDBCusername</param-name>
      <param-value>guest</param-value>
  </init-param>
  ```

  Default: This is a required `<init-param>`, there is no default value.

- `JDBCpassword` - The password associated with the above username. This is stored as simple text, so make sure that the JDBC user doesn’t have any significant privileges! If there is no password required, you must still set the this `<init-param>`, just leave the value element empty.

  Example:

```
<init-param>
    <param-name>JDBCpassword</param-name>
    <param-value>abracadabra</param-value>
</init-param>
```

Default: This is a required `<init-param>`, there is no default value.

- `JDBCMaxResponseLength` - This limits the number of lines that the DRDS will for a given client request. For debugging I use 300. For production I use 300000. Your milage may vary depending on the amount of memory resources available. You may wish to perform some testing on your server to establish an appropriate value. *Warning: Small values may lead to incomplete returns from queries.*

  Example:

```
<init-param>
    <param-name>JDBCMaxResponseLength</param-name>
    <param-value>50000</param-value>
</init-param>
```

  Default: : If this parameter is not set (does not appear in as an `<init-param>`) then it is set to 100.

- `UseDatasetName` - This is a (probably temporary) hack. Some databases (in particular MS-SQL Server 7.0) require that the database name and the owner of the database be specified in every variable and table name. This is awkward for the current implementation of the DRDS. The work around is to name the data set (in the DDS file) with the database name and owner name of the table being served. For example in one data set that I server the database name is ”EOSDB” and the owner of the database is ”DBO”. I set the `<init-param>` UseDatasetName then I define the DDS as follows:

```
Dataset {
    Sequence {
        Float64 battery;
        Float64 checksum;
        Float64 data_age;
    } Drifters;
} EOSDB.DBO;
```

  Thus the hack is invoked. It doesn’t matter if the value of this init-param is empty (although if it’s not you should set it to ”true”), it simply needs to appear in the `web.xml` file.If you don’t want to use this hack then DO NOT even included the init-param ”UseDatasetName” in the `web.xml` entry for the DRDS.

  Example:

```
<init-param>
    <param-name>JDBCDriver</param-name>
    <param-value>com.merant.sequelink.jdbc.SequeLinkDriver</param-value>
</init-param>
```

Default: If this `<init-param>` does not appear in the configuration then the hack is not invoked..

- `DebugOn` - Values specific to the DRDS are: `JDBC`

Example of web.xml content:

```
<servlet>
    <servlet-name>
        drds
    </servlet-name>
    <servlet-class>
        dods.servers.sql.drds
    </servlet-class>
    <init-param>
        <param-name>JDBCdriver</param-name>
        <param-value> com.merant.sequelink.jdbc.SequeLinkDriver</param-value>
    </init-param>
    <init-param>
        <param-name>JDBCconnectionURL</param-name>
        <param-value>jdbc:sequelink://sugar:19996</param-value>
    </init-param>
    <init-param>
        <param-name>JDBCusername</param-name>
        <param-value>guest</param-value>
    </init-param>
    <init-param>
        <param-name>JDBCpassword</param-name>
        <param-value></param-value>
    </init-param>
    <init-param>
        <param-name>JDBCMaxResponseLength</param-name>
        <param-value>300</param-value>
    </init-param>
    <init-param>
        <param-name>UseDatasetName</param-name>
        <param-value></param-value>
    </init-param>
    <init-param>
        <param-name>INFOcache</param-name>
        <param-value>/usr/Java-DODS/sdds-testsuite/info/</param-value>
    </init-param>
    <init-param>
        <param-name>DDScache</param-name>
        <param-value>/usr/Java-DODS/sdds-testsuite/dds/</param-value>
    </init-param>
    <init-param>
        <param-name>DAScache</param-name>
        <param-value>/usr/Java-DODS/sdds-testsuite/das/</param-value>
    </init-param>
    <init-param>
        <param-name>DebugOn</param-name>
        <param-value>showRequest showResponse JDBC</param-value>
    </init-param>
</servlet>
```

4.3.3 Configure the Table Data Types

Each database table (including table "views") you wish to serve must be described with a DDS and 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 DAP data types that most closely match yours.

First, determine the DAP 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 servlet's `web.xml` 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 data types described in table 4.1 are only the primitive data types (except for Array). DAP2 supports more sophisticated data types: Grids, Arrays, Structures, and Sequences, among others For information about all these, refer to the *The OPeNDAP User Guide*.

Make a DDS

In a file named for your table, in the directory nominated by the `dds_cache_dir` directive in the `web.xml` file (see 41), put a DDS for your table. This must contains declarations for each of the data types you expect a user to ask for (or select over). The fields 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.

Describe the structure of your dataset with the DDS.

The data served from a DRDS server is always enclosed in a DAP2 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 OPeNDAP User Guide*. There is also a very cursory introduction to these structures in *The DODS Quick Start Guide*.

JDBC Data Type	DAP2 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 DAP2 Data Types

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
```

Make a DAS

Provide details about each variable with the 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";
    }
  }
}
```

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 4.3.2 on page 46.) This should bring up a list of datasets you can select from.

5
Constructing the URL

After a dataset has been installed, and the handler 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 4.3.3 on page 54.)

Suppose your web server's document root is at:

```
/var/www/html/
```

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

```
/var/www/html/datasets/atlantic/cooldata.nc
```

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

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

Note that this URL will work for a DAP 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*. See note on page 12.

5.1 Constructing a OPeNDAP/JGOFS URL

The situation is slightly different for JGOFS data.

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

```
test0=def(/var/www/html/data/test0.data)
s87_xbt=def(/var/www/html/data/xbt.catalog)
s87_ctd=def(/var/www/html/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://test.opendap.org/cgi-bin/nph-dods/s87_xbt
```

6
Testing the Installation

It is possible to test the 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. (A command-line web client called `getdap` is provided with `libdap` which can retrieve text from Web servers, and print it on standard output. Look for it in the `/usr/local/bin` directory. If you try it out on a couple of URLs you are familiar with, you'll quickly see how it works.)

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

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

6.1 Testing the Web Server

To begin, confirm that the web 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 an OPeNDAP 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/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/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 OPeNDAP server installation.

6.2 Testing the OPeNDAP Server

You can run the handler programs locally, that is, without going over the Internet. Sometimes this is the simplest way to make sure that everything is running.

```
./dap_nc_handler -o dds  datafile.nc
```

The program should print the dataset's DDS on standard output. Use the `-h` option to see the list of options the handlers accept.

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

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

The 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 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:

```
> getdap http://test.opendap.org/opendap/nph-dods/data/nc/test.nc.ver
> getdap http://test.opendap.org/opendap/nph-dods/data/nc/test.nc.info
> getdap http://test.opendap.org/opendap//nph-dods/data/nc/test.nc.help
```

Now we need to test the requests that use the handler programs, such as `dap_nc_handler`. 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:

```
> getdap http://test.opendap.org/opendap/nph-dods/data/nc/test.nc.das
```

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

```
> getdap -a http://test.opendap.org/opendap/nph-dods/data/nc/test.nc
```

Refer to *The OPeNDAP 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 server.

Check the list of services and helper programs in Section 1.3 on page 14. From a web browser, you can access all the DAP services, except the data service, which returns binary, not ASCII, data. That one can only be easily tested from a DAP-enabled 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 OPeNDAP User Guide*.

6.3 Error Logs

When troubleshooting an OPeNDAP data 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 OPeNDAP server software writes any messages it issues to the error log.

You can make the error messages slightly more verbose by editing the `DODS_Dispatch.pm` file (in `/usr/share/dap-server`). Find the `$debug` variable, and set it to 1 or 2 instead of zero. Note that these messages are, as of version 3.5, now written to the `dbg_log` file which is usually in `/usr/share/dap-server`. Check the `DODS_Dispatch.pm` file to determine the actual path.

6.4 User Support

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

7
Documenting Your Data

Documenting your data will make it useful to more people.

OPeNDAP's version 3.5 server 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 `dap_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 `.info` to a DAP URL. For example to get the HTML info page for an netCDF file, you might type something like this:

```
http://test.opendap.org/opendap/nph-dods/data/nc/fnoc1.nc.info
```

The Info 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 `fnoc2.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 OPeNDAP/JGOFS Handler

Yes, this is different for JGOFS, too.

The JGOFS handler provides usage data in almost the same way as the other servers, but the `dap_usage` helper program is called `usage-jg`, and it functions slightly differently. For the general JGOFS handler 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 JGOFS handler and save that document in a file named `jg.html` in the cgi-bin directory that holds the server programs.

> **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 DAP 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 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. You can 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 server that serves URLs.

A *catalog server* , or *file server* , is just a collection of DAP 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 data servers, but used to serve information about data, rather than the data themselves. 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 FreeForm Data Handler

Any handler that serves Sequences can be a catalog server.

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

If you haven't done so already, download and install the FreeForm handler (or whatever server you intend to use). If you're using the FreeForm handler, 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 handler 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-dods/avhrr/1979/4/t79101185359.pvu.Z
1979/04/15:19:48:52  http://dods.gso.uri.edu/cgi-bin/nph-dods/avhrr/1979/4/t79105194852.pvu.Z
1979/04/16:19:40:24  http://dods.gso.uri.edu/cgi-bin/nph-dods/avhrr/1979/4/t79106194024.pvu.Z
1979/04/17:08:00:11  http://dods.gso.uri.edu/cgi-bin/nph-dods/avhrr/1979/4/t79107080011.pvu.Z
1979/04/17:09:49:59  http://dods.gso.uri.edu/cgi-bin/nph-dods/avhrr/1979/4/t79107094959.pvu.Z
1979/04/17:19:28:33  http://dods.gso.uri.edu/cgi-bin/nph-dods/avhrr/1979/4/t79107192833.pvu.Z
1979/04/18:07:50:34  http://dods.gso.uri.edu/cgi-bin/nph-dods/avhrr/1979/4/t79108075034.pvu.Z
1979/04/18:20:57:43  http://dods.gso.uri.edu/cgi-bin/nph-dods/avhrr/1979/4/t79108205743.pvu.Z
```

Make a format file to match.

Now you need to create a *format file* that describes the layout of the data file to the FreeForm handler. 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.)

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 OPeNDAP Data Handler

NOTE: This text covers building a data handler for OPeNDAP's Server3 data server. This will be updated soon (winter 2007) with information about building data handler modules for Server4. You may also find the Writing an OPeNDAP Server guide on our web page helpful. It provides an example and sample source code for a real data handler.

Though several different handlers are included in the OPeNDAP software, some users may wish to write their own data handlers. The architecture of the `httpd` server and the OPeNDAP 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 data handler 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 handlers.
- The data may be arranged in a fashion that allows a user to optimize the access of those data by rewriting one of the handlers.
- The user may wish to provide ancillary data to clients not anticipated by the writers of the servers available.

The design of the OPeNDAP libdap 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 handlers provided with the server are written in C++, they may be written in any language from which the OPeNDAP libraries may be called. There is also a Java version of the class library.

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 libdap 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 libdap software, and instructions about how to write a new server.

A

Getting the OPeNDAP Software

Get the software from OPeNDAP Home page. If you can, it is usually advisable to download the pre-compiled binaries. If you do so, be sure to follow the directions that acompany the distributions, even if you're upgrading a server, since the places where the server's components are installed can change as the software evolves. Installation instructions for the binary distributions can be found on the web pages where you downloadded the software.

If you can't use the pre-compiled binaries, download the source code from the OPeNDAP Home page. You will need a few different files, depending on the handler(s) you want to install. The "Download Software" page has instructions specifying which files to use. If you are building the software yourself, be sure to read the README, INSTALL and NEWS files included in the distributions.

You will need to download the dap-server 'base' software and one or more handlers. The base software and each handler each is contained in a single `tar.gz` or source `rpm` file. To build the source files, expand them, run the enclosed `configure` script, run `make` and then `make install`. Once the software is installed, see the configuration instructions in Section 2 on page 17.

B

GNU Free Documentation License

Version 1.2, November 2002

51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA

Preamble

The purpose of this License is to make a manual, textbook, or other functional and useful document ”free” in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of ”copyleft”, which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The **”Document”**, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as **”you”**. You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A **”Modified Version”** of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A **”Secondary Section”** is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document’s overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The **”Invariant Sections”** are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The **”Cover Texts”** are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A **”Transparent”** copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not ”Transparent” is called **”Opaque”**.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats

include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The **”Title Page”** means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, ”Title Page” means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

A section **”Entitled XYZ”** means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as **”Acknowledgements”**, **”Dedications”**, **”Endorsements”**, or **”History”**.) To **”Preserve the Title”** of such a section when you modify the Document means that it remains a section ”Entitled XYZ” according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document’s license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and

legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do

this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties–for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in

the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an ”aggregate” if the copyright resulting from the compilation is not used to limit the legal rights of the compilation’s users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document’s Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled ”Acknowledgements”, ”Dedications”, or ”History”, the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License ”or any later version” applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright ©YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled ”GNU Free Documentation License”.

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the ”with...Texts.” line with this:

> with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Index

Symbols

A

B

C

D

E

F

G

H

I

J

L

M

N