DODS FreeForm ND Server User's Guide
Using DODS to Serve Data in Unusual Formats
Version 1.2

NGDC, URI

August 30, 1999

THANKS

To the developers of FreeForm and FreeForm ND: Tom Carey, S.D Davis, Liping Di, Rich Fozzard, Kevin Frender, Ted Habermann, Mark A. Ohrenshall, John Rex, Mark Van Gorp, and others. Special thanks to Mark A. Ohrenshall for his meticulous documentation of the design and development of FreeForm ND and for writing this document.

This document has been updated to include information on FreeForm ND and the DODS FreeForm ND Server by Emma Gibson and Tom Sgouros.

TRADEMARKS

Preface

This document describes the DODS FreeForm ND Server. It is not a complete description of the FreeForm ND software. For that, please refer to the FreeForm ND manual.

This document contains much material originally written at the National Oceanic and Atmospheric Administration's National Environmental Satellite, Data, and Information Service, which is part of the National Geophysical Data Center in Boulder, Colorado.

This document has been updated to include information on FreeForm ND, the last release of FreeForm. FreeForm is now supported only for use with the Distributed Oceanographic Data System; see the DODS Home page for more information.

We are interested in your comments about the DODS software, and the FreeForm ND software and this document. Send them to: support@unidata.ucar.edu.

Using FreeForm ND with DODS, a researcher can easily make his or her data available to the wider community of DODS users without having to convert that data into another data file format. This document presents the FreeForm NDsoftware, and shows how to use it with DODS.

0.1 Tasks Illustrated in this Guide

For a quick start to getting, installing, and using the FreeForm ND software, see the list below of tasks described in this document.

- Quick start. (page **??**)
- Getting and installing the FreeForm ND software. (page **??**)
- Installing the rest of the DODS software. (page **??**)
- Troubleshooting. (page **??**)

0.2 Who is this Guide for?

This guide is for people who wish to use FreeForm ND to serve scientific datasets using the DODS software. Scientists who wish to share their data with colleagues may also find this a useful system, since it is a relatively simple matter to set up a server that can allow remote access to your data.

This documentation assumes that the readers are familiar with computers and the internet, but are not necessarily programmers. More than a passing familiarity with different data file formats will be useful, as will an understanding of elementary internet concepts, such as URLs and http.

This manual also assumes some familiarity with the DODS software. If you are starting from scratch, knowing nothing at all about DODS, we strongly encourage you to browse the *The DODS User Guide* before reading too far here.

0.3 Organization of this Document

This book contains both introductory and reference material. There is also a description of the installation procedure.

Chapter 1 contains an overview of the DODS FreeForm ND Server software, including how to get it and install it.

Chapter 2 provides a brief introduction to writing format descriptions and using the DODS FreeForm ND Server.

Chapter 3 provides detailed information about writing format descriptions to facilitate access to data in tabular formats.

Chapter 4 provides detailed information about writing format descriptions to facilitate access to data in non-tabular (array) formats.

Chapter 5 tells you how to work with header formats.

Chapter 6 describes the operation of the DODS FreeForm ND Server, with tips for writing format files.

Chapter 8 presents FreeForm ND file name conventions, the search rules for locating format files, and standard command line arguments for FreeForm ND programs.

Chapter 9 shows you how to use the FreeForm ND program `newform` to convert data from one format to another and also how to read the data in a binary file.

Chapter 10 discusses the FreeForm ND program `checkvar`, which you can use to check data distribution and quality.

Appendix A on page 93 provides explanations for a small selection of tools that will be useful for programmers working with the HDF file format.

Appendix B on page 105 presents a list of common FreeForm ND error messages. These are the error messages that may be issued by the FreeForm ND utilities, such as `newform`, not the DODS FreeForm ND Server.

Table 1: Typographic Conventions

`Literal text`	Typed by the computer, or in a code listing.
User input	Type this precisely as written.
Variables	Used as a place holder for a user-specified or variable value. Choose an appropriate value and use that in place.
Button Text	Used to indicate text on a GUI button.
Menu Name	This is the name of a GUI menu.

0.4 Conventions

The typographic conventions shown in Table 1 are followed in this guide and all the other DODS documentation.

When referring to a button in a menu, we will often use the notation: **Menu,Button**. For example, **Options,Colors,Foreground** would indicate the **Foreground** button in the Colors menu, selected under the Options menu.

A position box is often used in this book to indicate column position of field values in data files. It is shown at the beginning of a data list in the documentation, but does not appear in the data file itself. It looks something like this:

```
1         2         3         4         5         6
0123456789012345678901234567890123456789012345678901234567890
```

Contents

List of Figures

List of Tables

1 Introduction

The DODS FreeForm ND Server is a DODS server that uses FreeForm ND software to serve data from files in almost any format. The FreeForm ND Data Access System is a flexible system for specifying data formats to facilitate data access, management, and use. Since DODS allows data to be translated over the internet and read by a client regardless of the storage format of the data, the combination allows several format restrictions to be overcome.

The large variety of data formats is a primary obstacle in the way of creating flexible data management and analysis software. FreeForm ND was conceived, developed, and implemented at the National Geophysical Data Center (NGDC) to alleviate the problems that occur when you need to use data sets with varying native formats or to write format-independent applications.

DODS was originally conceived as a way to move large amounts of scientific data over the internet. As a consequence of establishing a flexible data transmission format, DODS also allows substantial independence from the storage format of the original data. Up to now, however, DODS servers have been limited to data in a few widely used formats. Using the DODS FreeForm ND Server, many more datasets can be made available through DODS.

1.1 The FreeForm ND Solution

FreeForm ND uses a *format descriptor* file to describe the format of one or more data files. This descriptor file is a simple text file that can be created with a text editor, describing the structure of your data files.

A traditional DODS server, illustrated in figure 1.1, receives a request for data from a DODS client who may be at some remote computer[1]. The data served by this server must be stored in one of the data formats supported by DODS (such as netCDF, HDF, or JGOFS), and the server uses specialized software to read this data from disk.

When it receives a request, the server reads the requested data from its archive, reformats the data into the DODS transmission format (also called the DODS Data Access Protocol, or DAP), and sends the data back to the client.

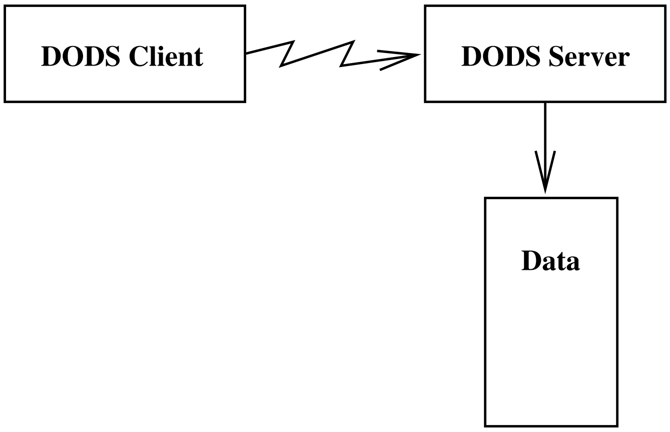

Figure 1.1: A Traditional DODS Server

The DODS FreeForm ND Server works in a similar fashion to a traditional DODS server, but before the server reads the data from the archive, it first reads the data format descriptor to determine how it should read the data. Only after it has absorbed the details of the data storage format does it attempt to read the data, pack it into the transmission format and send it on its way back to the client.

[1]The request comes via http. The DODS server is, in reality, an ordinary http server, equipped with a set of CGI programs to process requests from DODS clients. See Section **??** on page **??** and *The DODS User Guide* for more information.

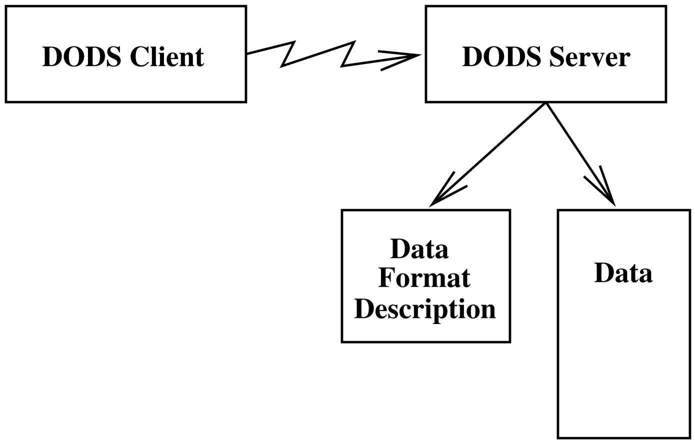

Figure 1.2: The DODS FreeForm ND Server

1.2 The FreeForm ND System

The DODS FreeForm ND Server comprises a format description mechanism, a set of programs for manipulating data, and the server itself. The software was built using the FreeForm ND library and data objects. These are documented in The FreeForm ND User's Guide.

The DODS FreeForm ND Server includes the following programs:

`nph-ff` The DODS FreeForm ND Server *dispatch script* . This Perl script (requires Perl version 5 or greater) receives the data request from the client, and dispatches it to one of several filter programs. For definitions of dispatch script and filter programs, and other details of the implementation of a DODS server, see *The DODS User Guide*.

`ff-dods` This filter program reads a FreeForm ND format description, and then reads data from the specified source, packs it into the transmission format, and sends it back to the client.

`ff-dds` In order to process the data stream correctly, the client first asks the server for a description of the data it will send. This description, called the Data Description Structure (DDS) is created and sent by this filter program.

`ff-das` Other metadata that may be useful to the user of this data may be sent to the client in the Data Attribute Structure (DAS), created and sent by this program. Section **??** on page **??** contains information about how to include metadata in this structure.

`asciival` This filter program allows data to be sent back to the user in ASCII format. This allows a wide variety of clients to access this data, including simple web browsers, such as Netscape Navigator.

`usage` This filter provides a way to get information about the data and about the server itself to a curious user. See Section **??** on page **??** for information about what can be sent and how to set it up.

The DODS FreeForm ND Server distribution also includes the following FreeForm ND utilities. These are quite useful to write and debug format description files.

`newform` This program reformats data according to the input *and output* specifications in a format description file.

`chkform` After writing a format description file, you can use this program to cross-check the description against a data file.

`readfile` This program is useful to decode the format used by a binary file. It allows you to try different formats on pieces of a binary file, and see what works.

1.3 Installing the DODS FreeForm ND Server

If you don't have the DODS FreeForm ND Server, and want it, follow these directions. If you have a copy of the DODS FreeForm ND Server, and want to know how to use it, see Chapter 2 for quick instructions and examples of its use, and Chapter 6 for further information.

You can get the DODS FreeForm ND Server from the DODS Home page. Follow the links to "Download Software" and then to "Current Release." If your computer is one of the platforms for which we provide a binary release, get that, otherwise get the source code.

To get a binary release, go to that page, click on the computer you use, and click on the "FreeForm" button in the "Servers" box. Click the **Download** button, and follow the directions. The server will make a custom binary file for you, which you then download.

To install the DODS FreeForm ND Server, first make sure you have an http server running on the machine where you plan to serve the data. If you are unsure whether one is running, you can use a web client, like Netscape, to send an http request to your own machine. *The DODS User Guide* contains some hints about installing a web server.

When you are sure a web server is properly installed and running, unpack the archive file you downloaded from the DODS Home page. You will find a directory called `DODS`, containing `etc` and `bin`.

❶ Copy all the files in the `etc` directory into your web server's CGI directory (see *The DODS User Guide* for tips on how to find this directory)

❷ Make sure the permissions of those files will allow the web server to execute them. Typically you might type use a command like:

```
> chmod 555 cgi-bin/*
```

❸ While you are here, copy the files in the `bin` directory to somewhere in your command `PATH`, or modify your command `PATH` to include the DODS `bin` directory. This is not strictly necessary to use the server, but will allow you to use the FreeForm ND tools that come with the DODS FreeForm ND Server distribution[2].

[2] There is a utility called `newform` that comes as part of the Solaris OS. This conflicts with the FreeForm ND utility of the same name. When installing this package on Solaris, make sure to put the `bin` directory before the `/usr/bin` directory in your `PATH`, or change the name of the FreeForm ND `newform` program.

The DODS FreeForm ND Server is now installed. You can confirm the installation with a web client like Netscape. Enter a URL containing your machine address, the CGI directory name, and then `nph-ff`, like this:

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

This is not a complete DODS URL, so you will get some kind of error message[3]. If the message you see mentions DODS, however, you have successfully installed the server. If it doesn't, and says something like "The requested URL was not found on this server," than you need to re-check your installation. (Check the permissions first, then make sure all the files from the `etc` directory were copied, then check the installation of the web server. Remember that the `nph-ff` program requires Perl v5.0 or greater; this is a common problem. There is a FAQ list at DODS Home page that may have useful suggestions.)

Now that the DODS FreeForm ND Serveris installed, Chapter 6 will explain how to use it to serve your data. It may be a good idea to familiarize yourself with the information in the intervening chapters, which will explain how to specify your data's format.

Compiling the DODS FreeForm ND Server

If the computer and operating system combination you use is not one of the ones we own, you will have to compile the DODS FreeForm ND Server from its source. Go to the "Download Source" page, and click on the "Core" button, the "FreeForm Server" button, and the "Packages" button.

When you have downloaded the source archive (it may be quite large), unpack it by typing:

```
> tar xf DODS-3993.tar
```

This archive file is made up of several other compressed archive files, so after this step you will need to unpack each of those files individually:

```
> gzip -dc DODS-dap-2.22.tar.gz | tar xf -
> gzip -dc DODS-ff-dods-2.22.tar.gz | tar xf -
```

and so on.

When all the archive files are unpacked, move to the `DODS/packages/src` directory, and type:

```
> ./configure
> make World
> make install
```

[3]The example is from a working DODS server. Try this example and compare it with the result from your own server.

Each of these packages has a README and INSTALL file containing directions in case of any trouble.

After the packages have been installed, go to the top DODS directory (wherever you have installed it), and type:

```
> ./configure
> make World
```

Then see Section 1.3 on page 5 for instructions on the final installation.

2

Quick Tour of the DODS FreeForm ND Server

This chapter provides you a quick introduction to the DODS FreeForm ND Server, including writing format descriptions and serving test datasets.

2.1 Getting Started Serving Data

To get going with the DODS FreeForm ND Server, follow these steps:

❶ Obtain and install the core DODS distribution and the DODS FreeForm ND Server distribution. (You can get these files from the DODS Home page.)

❷ Install the DODS FreeForm ND Server. (See the instructions in the INSTALL file accompanying the server.)

❸ Examine the structure of the data file(s) you intend to serve, and construct a FreeForm ND format definition file that describes the layout of data in the files. (Refer to the Chapter 3 for instructions about sequence data and Chapter 4 for array data. Consult *The DODS User Guide* if you do not know the difference between the two data types.)

❹ If you wish, you may include an output definition format within this file, to allow you to test that your input description is accurate. You can use the FreeForm ND utilities, such as `newform`, to validate the conversion. The Chapter 9 contains a detailed description of `newform`. This step is optional, since the DODS FreeForm ND Server ignores the output definition section of the format definition file.

❺ Although the DODS FreeForm ND Server can generate default DDS and DAS files, you may want to write these files yourself, to override the default data descriptions, or to add attribute data. The default descriptions are based on the format of the data the the DODS FreeForm ND Server receives from the FreeForm ND engine.

❻ Place the data files, and a corresponding format file for each data file, in a place where the DODS FreeForm ND Server can find them. This is generally in the server's document root, or in a subdirectory there. See *The DODS User Guide* for detailed instructions on installing a server if the document root is not a familiar concept.

Your data is now available to anyone who knows about your server. The next section contains examples of writing format descriptions.

2.2 Examples

You can easily create FreeForm ND format description files that describe the formats of input and output data and headers. The DODS FreeForm ND Server and other FreeForm ND-based programs then use these files to correctly access and manipulate data in various formats. An example format description file is shown and described below.

For complete information about writing format descriptions, see Chapter 3 and Chapter 4.

2.2.1 Sequence Data

Here is a data file, containing a sequence of four data types. (This data file and several of the other examples in this chapter are available under "Documentation" on the DODS web page (http://www.unidata.ucar.edu/packages/dods).)

Here is the data file, called `ffsimple.dat`:

```
Latitude and Longitude:   -63.223548 54.118314  -176.161101 149.408117
-47.303545 -176.161101 11.7125 34.4634
-25.928001   -0.777265 20.7288 35.8953
-28.286662   35.591879 23.6377 35.3314
 12.588231  149.408117 28.6583 34.5260
-63.223548   55.319598  0.4503 33.8830
 54.118314 -136.940570 10.4085 32.0661
-38.818812   91.411330 13.9978 35.0173
-34.577065   30.172129 20.9096 35.4705
 27.331551 -155.233735 23.0917 35.2694
 11.624981 -113.660611 27.5036 33.7004
```

The file consists of a single header line, followed by a sequence of records, each of which contains a latitude, longitude, temperature, and salinity.

Here is a format file you can use to read `ffsimple.dat`. It is called `ffsimple.fmt`:

```
ASCII_file_header "Latitude/Longitude Limits"
minmax_title 1 24 char 0
latitude_min 25 36 double 6
latitude_max 37 46 double 6
longitude_min 47 59 double 6
longitude_max 60 70 double 6

ASCII_data "lat/lon"
latitude 1 10 double 6
longitude 12 22 double 6
temp 24 30 double 4
salt 32 38 double 4

ASCII_output_data "output"
latitude 1 10 double 3
longitude_deg 11 15 short 0
longitude_min 16 19 short 0
longitude_sec 20 23 short 0
salt 31 40 double 2
temp 41 50 double 2
```

The format file consists of three sections. The first shows FreeForm ND how to parse the file header. The second section describes the contents of the data file. The third part describes how to write the data to another file. This part is not important for the DODS FreeForm ND Server, but is useful for debugging the input descriptions.

Download the `ffsimple` files described above, and type:

```
> newform ffsimple.dat
```

You should see results like this:

```
Welcome to Newform release 4.2.3 -- an NGDC FreeForm ND application

(ffsimple.fmt) ASCII_input_file_header  "Latitude/Longitude Limits"
File ffsimple.dat contains 1 header record (71 bytes)
Each record contains 6 fields and is 71 characters long.

(ffsimple.fmt) ASCII_input_data "lat/lon"
File ffsimple.dat contains 10 data records (390 bytes)
Each record contains 5 fields and is 39 characters long.

(ffsimple.fmt) ASCII_output_data        "output"
Program memory contains 10 data records (510 bytes)
Each record contains 7 fields and is 51 characters long.

   -47.304 -176   9  40           34.46     11.71
   -25.928    0 -46  38           35.90     20.73
   -28.287   35  35  31           35.33     23.64
    12.588  149  24  29           34.53     28.66
   -63.224   55  19  11           33.88      0.45
    54.118 -136  56  26           32.07     10.41
   -38.819   91  24  41           35.02     14.00
   -34.577   30  10  20           35.47     20.91
    27.332 -155  14   1           35.27     23.09
    11.625 -113  39  38           33.70     27.50
100% processed
```

Now take both the `ffsimple` files and put them into a directory in your web server's document root directory. (Refer to the *The DODS User Guide* for some tips on figuring out where that is.)

Here's an example, on a computer on which the web server document root is `/export/home/http/htdocs`:

```
> mkdir /export/home/http/htdocs/data
> cp ffsimple.* /export/home/http/htdocs/data
```

Now, using a common web browser such as Netscape Navigator, enter the following URL (substitute your machine name and CGI directory for the ones in the example):

```
http://dods.gso.uri.edu/cgi-bin/nph-ff/data/ffsimple.dat.asc
```

You should get something like the following in your web browser's window:

```
latitude, longitude, temp, salt
-47.3035, -176.161, 11.7125, 34.4634
-25.928, -0.777265, 20.7288, 35.8953
-28.2867, 35.5919, 23.6377, 35.3314
12.5882, 149.408, 28.6583, 34.526
-63.2235, 55.3196, 0.4503, 33.883
54.1183, -136.941, 10.4085, 32.0661
-38.8188, 91.4113, 13.9978, 35.0173
-34.5771, 30.1721, 20.9096, 35.4705
27.3316, -155.234, 23.0917, 35.2694
11.625, -113.661, 27.5036, 33.7004
```

Try this URL:

```
http://dods.gso.uri.edu/cgi-bin/nph-ff/data/ffsimple.dat.dds
```

This will show a description of the dataset structure (See *The DODS User Guide* for a detailed description of the DODS "Dataset Description Structure," or DDS.):

```
Dataset {
    Sequence {
        Float64 latitude;
        Float64 longitude;
        Float64 temp;
        Float64 salt;
    } lat/lon;
} ffsimple;
```

2.2.2 Array Data

If your data more naturally comes in arrays, you can still use the DODS FreeForm ND Server to serve your data. The FreeForm ND format for sequence data is somewhat simpler than the format for array data, so you may find it easier to begin with the example in the previous section.

One-dimensional Arrays

Here is a data file, called `ffarr1.dat`, containing four ten-element vectors:

```
123456789012345678901234567
 1.00  50.00 0.1000  1.1000
 2.00  61.00 0.3162  0.0953
 3.00  72.00 0.5623 -2.3506
 4.00  83.00 0.7499  0.8547
 5.00  94.00 0.8660 -0.1570
 6.00 105.00 0.9306 -1.8513
 7.00 116.00 0.9647  0.6159
 8.00 127.00 0.9822 -0.4847
 9.00 138.00 0.9910 -0.7243
10.00 149.00 0.9955 -0.3226
```

Here is a format file to read this data (`ffarr1.fmt`):

```
ASCII_input_data "simple array format"
index 1 5 ARRAY["line" 1 to 10 sb 23] OF float 1
data1 6 12 ARRAY["line" 1 to 10 sb 21] OF float 1
data2 13 19 ARRAY["line" 1 to 10 sb 21] OF float 1
data3 20 27 ARRAY["line" 1 to 10 sb 20] OF float 1

ASCII_output_data "simple array output"
index 1 7 ARRAY["line" 1 to 10] OF float 0
/data1 6 12 ARRAY["line" 1 to 10 sb 21] OF float 1
/data2 13 19 ARRAY["line" 1 to 10 sb 21] OF float 4
/data3 20 27 ARRAY["line" 1 to 10 sb 20] OF float 4
```

The output section is not essential for the DODS FreeForm ND Server, but is included so you can check out the data with the `newform` command.

Download the files from the DODS web site, and try typing:

```
> newform ffarr1.dat
```

You should see the `index` array printed out. Uncomment different lines in the output section of the example file to see different data vectors.

Now look a little closer at the input section of the file:

```
index 1 5 ARRAY["line" 1 to 10 sb 23] OF float 1
```

This line says that the array in question—called “index”—starts in column one of the first line, and each element takes up five bytes. The first element starts in column one and goes into column five. The array has one dimension, “line”, and is composed of floating point data. The remaining elements of this array are found by skipping the next 23 bytes (the newline counts as a character), reading the following five bytes, skipping the next 23 bytes, and so on.

Of course, the 23 bytes skipped in between the `index` array elements also contain data from other arrays. The second array, `data1`, starts in column 6 of line one, and has 21 bytes between values. The third array starts in column 13 of the first line, and the fourth starts in column 20.

Move the `ffarr1.*` files into your data directory:

```
> cp ffarr1.* /export/home/http/htdocs/data
```

Now you can look at this data the same way you looked at the sequence data. Request the DDS for the dataset with a URL like this one:

```
http://dods.gso.uri.edu/cgi-bin/nph-ff/data/ffarr1.dat.dds
```

You can see that the dataset is a collection of one-dimensional vectors. You can see the individual vectors with a URL like this:

```
http://dods.gso.uri.edu/cgi-bin/nph-ff/data/ffarr1.dat.asc?index
```

Multi-dimensional Arrays

Here's another example, with a two-dimensional array. (`ffarr2.dat`):

```
         1         2         3         4
1234567890123456789012345678901234567890
  1.00  2.00  3.00  4.00  5.00  6.00
  7.00  8.00  9.00 10.00 11.00 12.00
 13.00 14.00 15.00 16.00 17.00 18.00
 19.00 20.00 21.00 22.00 23.00 24.00
 25.00 26.00 27.00 28.00 29.00 30.00
```

There are no spaces between the data columns within an array row, but in order to skip reading the newline character, we have to skip one character at the end of each row. Here is a format file to read this data (`ffarr2.fmt`):

```
ASCII_input_data "one"
data 1 6 ARRAY["y" 1 to 5 sb 1]["x" 1 to 6] OF float 1

ASCII_output_data "two"
data 1 4 ARRAY["x" 1 to 6 sb 2]["y" 1 to 5] OF float 1
```

Again, the output section is only for using with the `newform` tool. Put these data files into your `htdocs` directory, and look at the DDS as you did with the previous example.

A Little More Complicated

You can use the DODS FreeForm ND Server to serve data with multi-dimensional arrays and one-dimensional vectors interspersed among one another. Here's a file containing this kind of data (`ffarr3.dat`):

```
         1         2         3         4
1234567890123456789012345678901234567890123
XXXX  1.00  2.00  3.00  4.00  5.00  6.00YY
XXXX  7.00  8.00  9.00 10.00 11.00 12.00YY
XXXX 13.00 14.00 15.00 16.00 17.00 18.00YY
XXXX 19.00 20.00 21.00 22.00 23.00 24.00YY
XXXX 25.00 26.00 27.00 28.00 29.00 30.00YY
```

In order to read this file successfully, we define three vectors to read the "XXXX", the "YY", and the newline. Here is a format file that does this (`ffarr3.fmt`):

```
dBASE_input_data "one"
headers 1 4 ARRAY["line" 1 to 5 sb 39] OF text 0
data 5 10 ARRAY["y" 1 to 5 sb 7]["x" 1 to 6] OF float 1
trailers 41 42 ARRAY["line" 1 to 5 sb 41] OF text 0
newline 43 43 ARRAY["line" 1 to 5 sb 42] OF text 0

ASCII_output_data "two"
data 1 4 ARRAY["x" 1 to 6 sb 2]["y" 1 to 5] OF float 0
/headers 1 6 ARRAY["line" 1 to 5] OF text 0
/trailers 1 4 ARRAY["line" 1 to 5] OF text 0
/newline 1 4 ARRAY["line" 1 to 5] OF text 0
```

The following chapters offer more detailed information about how exactly to create a format description file.

3
Format Descriptions for Tabular Data

Format descriptions define the formats of input and output data and headers. FreeForm ND provides an easy-to-use mechanism for describing data. FreeForm ND programs and FreeForm ND-based applications that you develop use these format descriptions to correctly access data. Any data file used by FreeForm ND programs must be described in a format description file.

This chapter explains how to write format descriptions for data arranged in tabular format—rows and columns—only. For data in non-tabular formats, see the next chapter.

Table 3.1: FreeForm ND Data Types

Name	Minimum Value	Maximum Value	Size in Bytes	Precision*
`char`			**	
`uchar`	0	255	1	
`short`	-32,767	32,767	2	
`ushort`	0	65,535	2	
`long`	-2,147,483,647	2,147,483,647	4	
`ulong`	0	4,294,967,295	4	
`float`	10^{-37}	10^{38}	4	6***
`double`	10^{-307}	10^{308}	8	15***

* Expressed as the number of significant digits
** User-specified
*** Can vary depending on environment

3.1 FreeForm ND Variable Types

The data sets you produce and use may contain a variety of variable types. The characteristics of the types that FreeForm ND supports are summarized in table 3.1, which is followed by a description of each type.

> **NOTE:** The sizes in table 3.1 are machine-dependent. Those given are for most Unix workstations.

`char` The `char` variable type is used for character strings. Variables of this type, including numerals, are interpreted as characters, not as numbers.

`uchar` The `uchar` (unsigned character) variable type can be used for integers between 0 and 255 (28- 1). Variables that can be represented by the `uchar` type (for example: month, day, hour, minute) occur in many data sets. An advantage of using the `uchar` type in binary formats is that only one byte is used for each variable. Variables of this type are interpreted as numbers, not characters.

`short` A `short` variable can hold integers between -32,767 and 32,767 ($2^{15} - 1$). This type can be used for signed integers with less than 5 digits, or for real numbers with a total of 4 or fewer digits on both sides of the decimal point (-99 to 99 with a precision of 2, -999 to 999 with a precision of 1, and so on).

`ushort` A `ushort` (unsigned short) variable can hold integers between 0 and 65,535 ($2^{16} - 1$).

`long` A `long` variable can hold integers between -2,147,483,647 and +2,147,483,647 ($2^{31} - 1$). This variable type is commonly used to represent floating point data as integers, which may be more portable. It can be used for numbers with 9 or fewer digits with up to 9 digits of precision, for example, latitude or longitude (-180.000000 to 180.000000).

`ulong` The `ulong` (unsigned long) variable type can be used for integers between 0 and 4,294,967,295 ($2^{32} - 1$).

`float, double` Numbers that include explicit decimal points are either `float` or `double` depending on the desired number of digits. A `float` has a maximum of 6 significant digits, a `double` has 15 maximum. The extra digits of a `double` are useful, for example, for precisely specifying time of day within a month as decimal days. One second of time is approximately 0.00001 day. The number specifying day (maximum = 31) can occupy up to 2 digits. A `float` can therefore only specify decimal days to a whole second (31.00001 occupies seven digits). A `double` can, however, be used to track decimal parts of a second (for example, 31.000001).

`header` Older versions of FreeForm ND included header variables. You can now specify header formats in format description files.

For details, see Section 3.4 on page 23 and also Chapter 5.

3.2 FreeForm ND File Types

FreeForm ND supports binary, ASCII, and dBASE file types. Binary data are stored in a fixed amount of space with a fixed range of values. This is a very efficient way to store data, but the files are machine-readable rather than human-readable. Binary numbers can be integers or floating point numbers.

Numbers and character strings are stored as text strings in ASCII. The amount of space used to store a string is variable, with each character occupying one byte.

The dBASE file type, used by the dBASE product, is ASCII text without end-of-line markers.

3.3 Format Description Files

Format description files accompany data files. A format description file can contain descriptions for one or more formats. You include descriptions for header, input, and output formats as appropriate. Format descriptions for more than one file may be included in a single format description file.

An example format description file is shown next. The sections that follow describe each element of a format description file.

```
1 / This format description file is for
2 / data files latlon.bin and latlon.dat.
3
4 binary_data "Default binary format"
5 latitude 1 4 long 6
6 longitude 5 8 long 6
7
8 ASCII_data "Default ASCII format"
9 latitude 1 10 double 6
10 longitude 12 22 double 6
```

Lines 1 and 2 are comment lines. Lines 4 and 8 give the format type and title, as described in Section 3.4.1 on page 23. Lines 5, 6, 9, and 10 contain variable descriptions, described in Section 3.4.2 on page 24. Blank lines signify the end of a format description

You can include blank lines between format descriptions and comments in a format description file as necessary. Comment lines begin with a slash (/). FreeForm ND ignores comments.

3.4 Format Descriptions

A format description file comprises one or more format descriptions. A format description consists of a line specifying the format type and title followed by one or more variable descriptions, as in the following example:

```
binary_data "Default binary format"
latitude 1 4 long 6
longitude 5 8 long 6
```

3.4.1 Format Type and Title

A line specifying the format type and title begins a format description. A *format descriptor* , for example, `binary_data`, is used to indicate format type to FreeForm ND. The *format title* , for example, “Default binary format”, briefly describes the format. It must be surrounded by quotes and follow the format descriptor on the same line. The maximum number of characters for the format title is 80 including the quotes.

Format Descriptors

Format descriptors indicate (in the order given) file type, read/write type, and file section. Possible values for each descriptor component are shown in the following table.

Table 3.2: Format Descriptor Components

File Type	Read/Write Type (optional)	File Section

`ASCII`
`binary`
`dBASE`
	`input`
	`output`
		`data`
		`file_header`
		`record_header`
		`file_header_separate`*
		`record_header_separate`*

* The qualifier `separate` indicates there is a header file separate from the data file.

The components of a format descriptor are separated by underscores (_). For example, `ASCII_output_data` indicates that the format description is for ASCII data in an output file. The order of descriptors in a format description should reflect the order of format types in the file. For instance, the descriptor `ASCII_file_header` would be listed in the format description file before

ASCII_data. The format descriptors you can use in FreeForm ND are listed in the next table, where XXX stands for ASCII, binary, or dBASE. (Example: XXX_data = ASCII_data, binary_data, or dBASE_data.)

Table 3.3: Format Descriptors

Data	Header	Special
XXX_data XXX_input_data XXX_output_data	XXX_file_header XXX_file_header_separate XXX_record_header XXX_record_header_separate XXX_input_file_header XXX_input_file_header_separate XXX_input_record_header XXX_input_record_header_separate XXX_output_file_header XXX_output_file_header_separate XXX_output_record_header XXX_output_record_header_separate	RETURN* EOL**

* The RETURN descriptor lets FreeForm ND skip over end-of-line characters in the data.
** The EOL descriptor is a constant indicating an end-of-line character should be inserted in a multi-line record.

For more information about header formats, see Chapter 5 on page 89.

3.4.2 Variable Descriptions

A variable description defines the name, start and end column position, type, and precision for each variable. The fields in a variable description are separated by white space. Two variable descriptions are shown below with the fields indicated. Each field is then described.

Here are two example variable descriptions. Each one consists of a name, a start position, and end position, a type, and a precision.

```
latitude    1  10  double  6
longitude  12  22  double  6
```

Name The variable name is case-sensitive, up to 63 characters long with no blanks. The variable names in the example are latitude and longitude. If the same variable is included in more than one format description within a format description file, its name must be the same in each format description.

Start Position The column position where the first character (ASCII) or byte (binary) of a variable value is placed. The first position is 1, not 0. In the example, the variable latitude is defined to start at position 1 and longitude at 12.

End Position The column position where the last character (ASCII) or byte (binary) of a variable value is placed. In the example, the variable latitude is defined to end at position 10 and longitude at 22.

Type The variable type can be a standard type such as char, float, double, or a special FreeForm ND type. The type for both variables in the example is double. See Section 3.1 on page 20 for descriptions of supported types.

Precision Precision defines the number of digits to the right of the decimal point. For float or double variables, precision only controls the number of digits printed or displayed to the right of the decimal point in an ASCII representation. The precision for both variables in the example is 6.

4

Format Descriptions for Array Data

If the tabular format discussed in the previous chapter doesn't describe your data well, FreeForm ND's array notation may prove useful. Describing a data file's organization as a set of as n-dimensional arrays allows for much more flexibility in writing format definitions. It also enables subsetting, pixel-manipulation, and reorienting arrays of arbitrary dimension and character.

4.1 Array Descriptor Syntax

FreeForm ND allows you to describe the same fundamental FreeForm ND data types in array notation. The arrays can have any number of dimensions, any number of elements in each dimension, and either an increasing or a decreasing sequencing in each dimension. Furthermore, elements in any dimension may be separated from each other (demultiplexed) and may even be placed in separate files. However, every element of an array must be of the same type.

Array descriptors are a string of n dimension descriptions for arrays of n dimensions. FreeForm ND accepts descriptions with the most significant dimension first (i.e. row-major order for 2 dimensions, plane-major order in 3 dimensions).

Individual dimension descriptions are enclosed in brackets. Each dimension description can contain various keywords and values which specify how the dimension is set up. Some of the specifications are optional; if you do not specify them, they default to a specific value.

> **NOTE:** You must not mix array and tabular formats within the input and output sections of the format definition file. Only one type of notation can be used within each section of the format description file, although the sections may use different forms. For example, a file's input format could use array definitions, but the output format might be entirely tabular.

The dimension description variables include:

dimension name (REQUIRED) A name for the dimension. This can be any ASCII string enclosed in double-quotes ("). The name for each dimension must be unique throughout the array descriptor.

This example specifies that a dimension named "latitude" exists:

```
["latitude" 0 to 180]
```

starting and ending indices (REQUIRED) A starting and ending index specifying a range for the dimension. The starting and ending indices are specified as integers separated by the word "to" following the dimension name. As long as both numbers are integral, there are no other restrictions on their values.

This example specifies that the dimension "temperature" has indices ranging from -50 to +50:

```
["temperature" -50 to 50]
```

granularity (optional) A specification for the density of elements in the indices. The number provided after the "by" keyword specifies how many index positions are to be skipped to find the next element.

This example specifies that index values 0, 50, 100, 150 and 200 are the only valid index values for the dimension "height":

```
["height" 0 to 200 by 50]
```

grouping (optional) A specification for splitting an array across "partitions" (files or buffers in memory). The number provided after the "gb" or "groupedby" keyword specifies how many elements of the dimension are in each partition. If no value is specified, the default is 0 (no partitioning). Each partition must have the same number of elements. Every more-significant dimension description (those to the left) must also have a grouping specified- "dangling" grouping specifications are not allowed. If a dimension is not partitioned, but is required to have a grouping specification because a less-significant dimension is partitioned, a grouping of M can be specified, where:

$$M = \mid end\_index - start\_index \mid +1$$

This example specifies that the dimension "latitude" is partitioned into 9 chunks of 10 "bands" of latitude each:

```
["latitude" 1 to 90 gb 10]
```

separation (optional) A specification for "unused space" in the array. The number provided after the "sb" or "separation" keyword specifies how many bytes of data following each element in the dimension should not be considered part of the array. An "element in the dimension" is considered to be everything which occurs in one index of that dimension. separation takes on a slightly different meaning if the dimension also has a specified grouping. In dimensions with a specified grouping, the separation occurs at the end of each partition, not after every element.

This example specifies a 2-dimensional array with 4 bytes between the elements in the "columns" and an additional 2 bytes at the end of every row:

```
["lat" -90 to 90 sb 2]["lon" -180 to 179 sb 4]
```

4.2 Handling Newlines

The convention of expecting a newline to follow each record of ASCII data becomes troublesome when dealing with array data, especially when expressed using format description notation that is intended for tabular data. It is the FreeForm ND convention that there is an implicit newline after the last variable of an ASCII format.

For example, these two format descriptions are equivalent:

```
ASCII_data "broken time --- BIP"
year 1 2 uint8 0
month 3 4 uint8 0
day 5 6 uint8 0
hour 7 8 uint8 0
minute 9 10 uint8 0
second 11 14 uint16 2

dBASE_data "broken time --- BIP"
year 1 2 uint8 0
month 3 4 uint8 0
day 5 6 uint8 0
hour 7 8 uint8 0
minute 9 10 uint8 0
second 11 14 uint16 2
EOL 15 16 constant 0
```

However, the EOL variable shown here assumes that newlines are two bytes long, which is true only on a PC. FreeForm ND adjusts to this by assuming that ASCII data always has native newlines, and it updates the starting and ending position of EOL variables and subsequent variables accordingly.

The EOL variable is typically used to define a record layout that spans multiple lines. However, the EOL variable in combination with the dBASE format type can completely replace the ASCII format type.We recommend using the dBASE format type when describing ASCII arrays, to ensure that separation, if specified, takes into account the length of any newlines.

In this output format a newline separates each band of data, but it would be just as easy to omit the newlines entirely.

```
dBASE_input_data "broken time --- BIP"
year 1 2 array["x" 1 to 10 sb 14] of uint8 0
month 3 4 array["x" 1 to 10 sb 14] of uint8 0
day 5 6 array["x" 1 to 10 sb 14] of uint8 0
hour 7 8 array["x" 1 to 10 sb 14] of uint8 0
minute 9 10 array["x" 1 to 10 sb 14] of uint8 0
second 11 14 array["x" 1 to 10 sb 12] of uint16 2
EOL 15 16 array["x" 1 to 10 sb 14] of constant 0

dBASE_output_data "broken time - BSQ"
year 1 2 array["x" 1 to 10] of uint8 0
EOL 21 22 constant 0
month 23 24 array["x" 1 to 10] of uint8 0
EOL 43 44 constant 0
day 45 46 array["x" 1 to 10] of uint8 0
EOL 65 66 constant 0
hour 67 68 array["x" 1 to 10] of uint8 0
EOL 87 89 constant 0
minute 90 91 array["x" 1 to 10] of uint8 0
EOL 110 111 constant 0
second 112 115 array["x" 1 to 10] of uint16 2
EOL 132 133 constant 0
```

NOTE: The separation size now takes into account the two-character PC newline. To use this format description with a native ASCII file on UNIX platforms, it would be necessary to change the separation sizes of 12 and 14 to 11 and 13, respectively.

4.3 Examples

The following examples should be helpful in understanding the array notation.

4.3.1 Tabular versus Array Descriptions

Array notation can simply replace the tabular format description, as in these examples.

A single element can be described in tabular format:

```
year 1 2 uint8 0
```

or as an array:

```
year 1 2 array["x" 1 to 10] of uint8 0
```

An image file can be described in tabular format:

```
binary_input_data "grid data"
data 1 1 uint8 0
```

or as an array:

```
binary_input_data "grid data"
data 1 1 array["rows" 1 to 180] ["cols" 1 to 360] of uint8 0
```

Multiplexed data can be described in tabular format:

```
ASCII_data "broken time --- tabular"
year 1 2 uint8 0
month 3 4 uint8 0
day 5 6 uint8 0
hour 7 8 uint8 0
minute 9 10 uint8 0
second 11 14 uint16 2
```

or as an array:

```
ASCII_data "broken time -- BIP"
year 1 2 array["x" 1 to 10 sb 12] of uint8 0
month 3 4 array["x" 1 to 10 sb 12] of uint8 0
day 5 6 array["x" 1 to 10 sb 12] of uint8 0
hour 7 8 array["x" 1 to 10 sb 12] of uint8 0
minute 9 10 array["x" 1 to 10 sb 12] of uint8 0
second 11 14 array["x" 1 to 10 sb 10] of uint16 2
```

These two format descriptions communicate much the same information, but the array example also indicates that the data file is blocked into ten data values for each variable.

In this example, the data is not multiplexed:

```
ASCII_data "broken time -- BSQ"
year 1 2 array["x" 1 to 10] of uint8 0
month 21 22 array["x" 1 to 10] of uint8 0
day 41 42 array["x" 1 to 10] of uint8 0
hour 61 62 array["x" 1 to 10] of uint8 0
minute 81 82 array["x" 1 to 10] of uint8 0
second 101 104 array["x" 1 to 10] of uint16 2
```

The starting position indicates the file offset of the first element of each array, the same as with the alternative definition given for starting position in tabular data format descriptions.

4.3.2 Array Manipulation

Consider a 6x6 array of data with an “XXXX” header and a “YY” trailer on each line. Each data element is a space, a row (“y”) index, a comma, and a column (“x”) index, as shown below:

```
XXXX 0,0 0,1 0,2 0,3 0,4 0,5YY
XXXX 1,0 1,1 1,2 1,3 1,4 1,5YY
XXXX 2,0 2,1 2,2 2,3 2,4 2,5YY
XXXX 3,0 3,1 3,2 3,3 3,4 3,5YY
XXXX 4,0 4,1 4,2 4,3 4,4 4,5YY
XXXX 5,0 5,1 5,2 5,3 5,4 5,5YY
```

The goal is to produce a data file that looks like the data below. To do that, we need to strip the headers and trailers, and transpose rows and columns:

```
0,0 1,0 2,0 3,0 4,0 5,0
0,1 1,1 2,1 3,1 4,1 5,1
0,2 1,2 2,2 3,2 4,2 5,2
0,3 1,3 2,3 3,3 4,3 5,3
0,4 1,4 2,4 3,4 4,4 5,4
0,5 1,5 2,5 3,5 4,5 5,5
```

The key to writing the input format description is understanding that the input data file is composed of four interleaved arrays:

❶ The “XXXX” headers

❷ The data

❸ The “YY” trailers

❹ The newlines

The array of headers is a one-dimensional array composed of six elements (one for each line) with each element being four characters wide and separated from the next element by 28 bytes (24 + 2 + 2 — 24 bytes for a row of data plus 2 bytes for the trailer plus two bytes for the newline).

The array of data is a two-dimensional array of six elements in each dimension with each element being four characters wide, each row is separated from the next by eight bytes (columns are adjacent and so have zero separation), and the first element begins in the fifth byte of the file (counting from one).

The array of trailers is a one-dimensional array composed of six elements with each element being two characters wide, each element is separated from the next by 30 bytes, and the first element begins in the 29th byte of the file.

The array of newlines is a one-dimensional array composed of six elements with each element being two characters wide (on a PC), each element is separated from the next by 30 bytes, and the first element begins in the 31st byte of the file.

The FreeForm ND input format description needed is:

```
dBASE_input_data "one"
headers 1 4 ARRAY["line" 1 to 6 separation 28] OF text 0
data 5 8 ARRAY["y" 1 to 6 separation 8]["x" 1 to 6] OF text 0
trailers 29 30 ARRAY["line" 1 to 6 separation 30] OF text 0
PCnewline 31 32 ARRAY["line" 1 to 6 separation 30] OF text 0
```

The output data is composed of two interleaved arrays:

❶ The data

❷ The newlines

The array of data now has a separation of two bytes between each row, the first element begins in the first byte of the file, and the order of the dimensions has been switched.

The array of newlines now has a separation of 24 bytes and the first element begins in the 25th byte of the file. Each array can be operated on independently. In the case of the data array we simply transposed rows and columns, but we could do other reorientations as well, such as resequencing elements within either or both dimensions.

The FreeForm ND output format description needed is:

```
dBASE_output_data "two"
data 1 4 ARRAY["x" 1 to 6 separation 2]["y" 1 to 6] OF text 0
PCnewline 25 26 ARRAY["line" 1 to 6 separation 24] OF text 0
```

4.3.3 Sampling and Data Manipulation

With a wider range of descriptive possibilities, FreeForm can more easily be used for sampling and subsetting data, as in these examples.

The following array descriptor pair subsets a two-dimensional array, retrieving one quarter (the north-west quarter of the earth).

```
INPUT: ["latitude" -90 to 90] ["longitude" -179 to 180]
OUTPUT: ["latitude" 0 to 90] ["longitude" -179 to 0]
```

The following array descriptor pair flips a two-dimensional array row-wise (vertically).

```
INPUT: ["row" 0 to 100] ["column" 13 to 42]
OUTPUT: ["row" 100 to 0] ["column" 13 to 42]
```

The following array descriptor pair rotates a two-dimensional array 90 degrees (exchanging rows and columns).

```
INPUT: ["row" 0 to 10] ["column" 0 to 42]
OUTPUT: ["column" 0 to 42] ["row" 0 to 10]
```

The following array descriptor pair outputs every other plane from a three-dimensional array (essentially cutting the depth resolution in half).

```
INPUT: ["plane" 1 to 18] ["row" 0 to 10] ["column" 0 to 42]
OUTPUT: ["plane" 1 to 18 by 2] ["row" 0 to 10] ["column" 0 to 42]
```

The following array descriptor pair replicates every plane from a three-dimensional array three times (essentially tripling the depth).

```
INPUT: ["plane" 1 to 54 by 3] ["row" 0 to 10] ["column" 0 to 42]
OUTPUT: ["plane" 1 to 54] ["row" 0 to 10] ["column" 0 to 42]
```

This array descriptor pair outputs the middle 1/27 of a three dimensional array with depth and width exchanged and height halved and flipped:

```
INPUT: ["plane" 1 to 27] ["row" 1 to 27] ["column" 1 to 27]
OUTPUT: ["column" 10 to 18] ["row" 18 to 10 by 2] ["plane" 10 to 18]
```

5
Header Formats

Headers are one of the most commonly encountered forms of metadata-data about data. Applications need the information contained in headers for reading the data that the headers describe. To access these data, applications must be able to read the headers. Just as there are many data formats, there are numerous header formats. You can include header format descriptions, which have exactly the same form as data format descriptions, in format description files.

5.1 Header Treatment in FreeForm ND

FreeForm ND is not 100 percent backwards compatible with FreeForm in the area of header treatment.

Headers have traditionally been handled differently from data in FreeForm ND. If a header format was not specified as either input or output, it was taken as both input and output. `newform` did little in processing headers, and FreeForm ND relied on extraneous utilities to work with headers.

5.1.1 New Behavior

In FreeForm ND, header formats are treated the same as data formats. This means that header formats must be identified as either input or output, explicitly or implicitly. If done explicitly, then either the input or the output descriptor will form the format type (e.g., `ASCII_input_header`). If done implicitly, then the same ambiguity resolution rules that apply to data formats will be applied to header formats. This means that ASCII header formats will be taken as input for data files with a `.dat` extension, dBASE header formats will be taken as input for data files with a `.dab` extension, and binary header formats will be taken as input for all other data files.

If an embedded header and the data have different file types, then either the header format or data format (preferably both) must be explicitly identified as input or output (for example, an ASCII header embedded in a binary data file). Obviously, ambiguous formats with different file types cannot both be resolved as input formats.

The same header format is no longer used as both an input and an output header format.

In FreeForm ND, `newform` honors output header formats that are separate (e.g., `ASCII_output_header_separate`). The header is written to a separate file which, unless otherwise specified, is named after the output data file with a `.hdr` extension. This requires that you name the output file using the `-o` option flag; redirected output cannot be used with separate output headers. The output header file name and path can be specified using the same keywords that tell FreeForm ND how to find an input separate header file (i.e., `header_file_ext`, `header_file_name`, and `header_file_path`).

When defining keywords to specify how an output header file is to be named, you must use a new type of equivalence section, `input_eqv`, which must appear in the format file along with `output_eqv`.

5.2 Header Types

FreeForm ND recognizes two types of headers. File headers describe all the data in a file whereas record headers describe the data in a single record or data block. FreeForm ND can read headers included in the data file or stored in a separate file. Header formats, like data formats, are described in format description files. For a list of the header descriptors you can use in format descriptions, see Chapter 3.

5.2.1 File Headers

A file header included in a data file is at the beginning of the file. Only one file header can be associated with a data file. Alternatively, a file header can be stored in a file separate from the data file.

In the following example, a file header is used to store the minimum and maximum for each variable and the data are converted from ASCII to binary. There are two variables, latitude and longitude. The file header format and data formats are described in the format description file `llmaxmin.fmt`.

Here is `llmaxmin.fmt`:

```
ASCII_file_header "Latitude/Longitude Limits"
minmax_title 1 24 char 0
latitude_min 25 36 double 6
latitude_max 37 46 double 6
longitude_min 47 59 double 6
longitude_max 60 70 double 6

ASCII_data "lat/lon"
latitude 1 10 double 6
longitude 12 22 double 6

binary_data "lat/lon"
latitude 1 4 long 6
longitude 5 8 long 6
```

The example ASCII data file `llmaxmin.dat` contains a file header and data as described in `llmaxmin.fmt`.

`llmaxmin.dat`:

```
         1         2         3         4         5         6         7
1234567890123456789012345678901234567890123456789012345678901234567890

Latitude and Longitude:   -83.223548 54.118314  -176.161101 149.408117
-47.303545 -176.161101
-25.928001    0.777265
-28.286662   35.591879
 12.588231  149.408117
-83.223548   55.319598
 54.118314 -136.940570
 38.818812   91.411330
-34.577065   30.172129
 27.331551 -155.233735
 11.624981 -113.660611
```

This use of a file header would be appropriate if you were interested in creating maps from large data files. By including maximums and minimums in a header, the scale of the axes can be determined without reading the entire file.

FreeForm ND naming conventions have been followed in this example, so to convert the ASCII data in the example to binary format, use the following simple command:

```
newform llmaxmin.dat -o llmaxmin.bin
```

The file header in the example will be written into the binary file as ASCII text because the header descriptor in `llmaxmin.fmt` (`ASCII_file_header`) does not specify read/write type, so the format is used for both the input and output header.

5.2.2 Record Headers

Record headers occur once for every block of data in a file. They are interspersed with the data, a configuration sometimes called a format sandwich. Record headers can also be stored together in a separate file.

The following format description file specifies a record header and ASCII and binary data formats for aeromagnetic trackline data.

Here is `aeromag.fmt`:

```
ASCII_record_header "Aeromagnetic Record Header Format"
flight_line_number 1 5 long 0
count 6 13 long 0
fiducial_number_corresponding_to_first_logical_record 14 22 long 0
date_MMDDYY_or_julian_day 23 30 long 0
flight_number 31 38 long 0
utm_easting_of_first_record 39 48 float 0
utm_northing_of_first_record 49 58 float 0
utm_easting_of_last_record 59 68 float 0
utm_northing_of_last_record 69 78 float 0
blank_padding 79 104 char 0

ASCII_data "Aeromagnetic ASCII Data Format"
flight_line_number 1 5 long 0
fiducial_number 6 15 long 0
utm_easting_meters 16 25 float 0
utm_northing_meters 26 35 float 0
mag_total_field_intensity_nT 36 45 long 0
mag_residual_field_nT 46 55 long 0
alt_radar_meters 56 65 long 0
alt_barometric_meters 66 75 long 0
blank 76 80 char 0
latitude 81 92 float 6
longitude 93 104 float 6

binary_data "Aeromagnetic Binary Data Format"
flight_line_number 1 4 long 0
fiducial_number 5 8 long 0
utm_easting_meters 9 12 long 0
utm_northing_meters 13 16 long 0
mag_total_field_intensity_nT 17 20 long 0
mag_residual_field_nT 21 24 long 0
alt_radar_meters 25 28 long 0
alt_barometric_meters 29 32 long 0
blank 33 37 char 0
latitude 38 41 long 6
longitude 42 45 long 6
```

The example ASCII file `aeromag.dat` contains two record headers followed by a number of data records. The header and data formats are described in `aeromag.fmt`. The variable count (second variable defined in the header format description) is used to indicate how many data records occur after each header.

`aeromag.dat`:

```
         1         2         3         4         5         6         7         8         9        10
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345
  420       5     5272     178       2   413669.  6669740.   333345.  6751355.
  420       5272   413669.  6669740.   2715963   2715449      1088      1348        60.157307 -154.555191
  420       5273   413635.  6669773.   2715977   2715464      1088      1350        60.157593 -154.555817
  420       5274   413601.  6669807.   2716024   2715511      1088      1353        60.157894 -154.556442
  420       5275   413567.  6669841.   2716116   2715603      1079      1355        60.158188 -154.557068
  420       5276   413533.  6669875.   2716263   2715750      1079      1358        60.158489 -154.557693
  411       10     8366     178       2   332640.  6749449.   412501.  6668591.
  411       8366   332640.  6749449.   2736555   2736538       963      1827        60.846806 -156.080185
  411       8367   332674.  6749415.   2736539   2736522       932      1827        60.846516 -156.079529
  411       8368   332708.  6749381.   2736527   2736510       917      1829        60.846222 -156.078873
  411       8369   332742.  6749347.   2736516   2736499       922      1832        60.845936 -156.078217
  411       8370   332776.  6749313.   2736508   2736491       946      1839        60.845642 -156.077560
  411       8371   332810.  6749279.   2736505   2736488       961      1846        60.845348 -156.076904
  411       8372   332844.  6749245.   2736493   2736476       982      1846        60.845062 -156.076248
  411       8373   332878.  6749211.   2736481   2736463      1015      1846        60.844769 -156.075607
  411       8374   332912.  6749177.   2736470   2736452      1029      1846        60.844479 -156.074951
  411       8375   332946.  6749143.   2736457   2736439      1041      1846        60.844189 -156.074295
```

This file contains two record headers. The first occurs on the first line of the file and has a count of 5, so it is followed by 5 data records. The second record header follows the first 5 data records. It has a count of 10 and is followed by 10 data records.

The FreeForm ND default naming conventions have been used here so you could use the following abbreviated command to reformat **aeromag.dat** to a binary file named **aeromag.bin**:

```
newform aeromag.dat -o aeromag.bin
```

The ASCII record headers are written into the binary file as ASCII text.

5.2.3 Separate Header Files

You may need to describe a data set with external headers. An external or separate header file can contain only headers-one file header or multiple record headers.

Separate File Header

Suppose you want the file header used to store the minimum and maximum values for latitude and longitude (from the llmaxmin example) in a separate file so that the data file is homogenous, thus easier for applications to read. Instead of one ASCII file (**llmaxmin.dat**), you will have an ASCII header file, say it is named **llmxmn.hdr**, and an ASCII data file-call it **llmxmn.dat**.

Here is **llmxmn.hdr**:

```
Latitude and Longitude:   -83.223548 54.118314  -176.161101 149.408117
```

And here is **llmxmn.dat**:

```
-47.303545 -176.161101
-25.928001    0.777265
-28.286662   35.591879
 12.588231  149.408117
-83.223548   55.319598
 54.118314 -136.940570
 38.818812   91.411330
-34.577065   30.172129
 27.331551 -155.233735
 11.624981 -113.660611
```

You will need to make one change to `llmaxmin.fmt`, adding the qualifier separate to the header descriptor, so that FreeForm ND will look for the header in a separate file. The first line of `llmaxmin.fmt` becomes:

```
ASCII_file_header_separate "Latitude/Longitude Limits"
```

Save `llmaxmin.fmt` as `llmxmn.fmt` after you make the change.

To convert the data in `llmxmn.dat` to binary format in `llmxmn.bin`, use the following command:

```
newform llmxmn.dat -o llmxmn.bin
```

> **NOTE:** When you run `newform`, it will write the separate header to `llmxmn.bin` along with the data in `llmxmn.dat`.

Separate Record Headers

Record headers in separate files can act as indexes into data files if the headers specify the positions of the data in the data file. For example, if you have a file containing data from 25 observation stations, you could effectively index the file by including a station ID and the starting position of the data for that station in each record header. Then you could use the index to quickly locate the data for a particular station.

Returning to the `aeromag` example, suppose you want to place the two record headers in a separate file. Again, the only change you need to make to the format description file (`aeromag.fmt`) is to add the qualifier separate to the header descriptor. The first line would then be:

```
ASCII_record_header_separate "Aeromagnetic Record Header Format"
```

The separate header file would contain the following two lines:

```
420       5     5272     178       2   413669.  6669740.   333345.  6751355.
411      10     8366     178       2   332640.  6749449.   412501.  6668591.
```

The data file would look like the current `aeromag.dat` with the first and seventh lines removed.

Assuming the data file is named `aeromag.dat`, the default name and location of the header file would be `aeromag.hdr` in the same directory as the data file. Otherwise, the separate header file name and location need to be defined in an equivalence table. (For information about equivalence tables, see the GeoVu Tools Reference Guide.)

5.2.4 The dBASEfile Format

Headers and data records in dBASE format are represented in ASCII but are not separated by end-of-line characters. They can be difficult to read or to use in applications that expect newlines to separate records. By using `newform`, dBASE data can be reformatted to include end-of-line characters.

In this example, you will reformat the dBASE data file `oceantmp.dab` (see below) into the ASCII data file `oceantmp.dat`. The input file `oceantmp.dab` contains a record header at the beginning of each line. The header is followed by data on the same line. When you convert the file to ASCII, the header will be on one line followed by the data on the number of lines specified by the variable count. The format description file `oceantmp.fmt` is used for this reformatting.

Here is `oceantmp.fmt`:

```
dbase_record_header "NODC-01 record header format"
WMO_quad 1 1 char 0
latitude_deg_abs 2 3 uchar 0
latitude_min 4 5 uchar 0
longitude_deg_abs 6 8 uchar 0
longitude_min 9 10 uchar 0
date_yymmdd 11 16 long 0
hours 17 19 uchar 1
country_code 20 21 char 0
vessel 22 23 char 0
count 24 26 short 0
data_type_code 27 27 char 0
cruise 28 32 long 0
station 33 36 short 0

dbase_data "IBT input format"
depth_m 1 4 short 0
temperature 5 8 short 2

RETURN "NEW LINE INDICATOR"

ASCII_data "ASCII output format"
depth_m 1 5 short 0
temperature 27 31 float 2
```

This format description file contains a header format description, a description for dBASE input data, the special RETURN descriptor, and a description for ASCII output data. The variable *count* (fourth from the bottom in the header format description) indicates the number of data records that follow each header. The descriptor RETURN lets `newform` skip over the end-of-line marker at the end of each data block in the input file `oceantmp.dab` as it is meaningless to `newform` here. Because the end-of-line marker appears at the end of the data records in each input data block, RETURN is placed after the input data format description in the format description file.

`oceantmp.dab`:

```
         1         2         3         4         5         6         7
1234567890123456789012345678901234567890123456789012345678901234567890
11000171108603131109998  46860210000000002767001027670020276700302767
110011751986072005690AM  46860910000000002928001028780020287200302872
11111176458102121909998  46810110000000002728009126890241110005000728
112281795780051918090PI  2681011000000000268900402711
```

Each dBASE header in `oceantmp.dab` is located from position 1 to 36. It is followed by four data records of 8 bytes each. Each record comprises a depth and temperature reading. The variable count in the header (positions 24-26) indicates that there are 4 data records each in the first 3 lines and 2 on the last line. This will all be more obvious after conversion.

To reformat `oceantmp.dab` to ASCII, use the following command:

```
newform oceantmp.dab -o oceantmp.dat
```

The resulting file `oceantmp.dat` is much easier to read. It is readily apparent that there are 4 data records after the first three headers and 2 after the last.

Here is `oceantmp.dat`:

```
         1         2         3         4
1234567890123456789012345678901234567890
11000171108603131109998  46860210000
    0                     27.67
   10                     27.67
   20                     27.67
   30                     27.67
11001175198607200569OAM  46860910000
    0                     29.28
   10                     28.78
   20                     28.72
   30                     28.72
11111176458102121909998  46810110000
    0                     27.28
   91                     26.89
  241                     11.00
  500                     07.28
11228179578005191809OPI  26810110000
    0                     26.89
   40                     27.11
```

6 The DODS FreeForm ND Server

The DODS FreeForm ND Server is a DODS server that uses FreeForm ND to convert and serve data in formats that are not directly supported by DODS servers. Bringing FreeForm ND's data conversion capacity into the DODS world widens data access for DODS clients, since any format that can be described in FreeForm ND can now be served by DODS.

Like all DODS servers, the DODS FreeForm ND Server responds to client requests for data by returning either data values or information about the data. It differs from other DODS servers because it invokes FreeForm ND to read the data from disk before serving it to the client.

The following sequence of steps illustrates how the DODS FreeForm ND Server works:

❶ A DODS client sends a request for data to a DODS FreeForm ND Server. The request must include the name of the file that contains the data, and may include a constraint expression to sample the data.

❷ The DODS FreeForm ND Server looks in its path for two files: a data file with the name sent by the client, and a format definition file to use with the data file. The format definition file contains a description of the data format, constructed according to the FreeForm ND syntax.

❸ The server uses both files in invoking the FreeForm ND engine. The FreeForm ND engine reads the data file and the format file, using the instructions in the latter to convert the former into data which is then passed back to the DODS FreeForm ND Server.

❹ On receiving the converted data, the DODS FreeForm ND Server converts the data into the DODS transmission format. The conversion may involve some adjustment of data types; these are listed in Section 6.2 on page 49. The server also applies any constraint expressions the client sent along with the URL.

❺ The server then constructs DDS and DAS files based on the format of the converted data. If the server has access to DDS and DAS files that describe the data, it applies those definition before sending them back to the client.

❻ Finally, the server sends the DDS, DAS, and converted data back to the client.

For information about how to write a FreeForm ND data description, refer to the Chapter 3 for sequence data and Chapter 4 for array data.

For an introduction to DODS, please refer to *The DODS User Guide*.

6.1 Differences between FreeForm ND and the DODS FreeForm ND Server

The DODS FreeForm ND Server is based on the same libraries used to make the FreeForm ND utilities. However, there are some important differences in the resulting software:

➨ The DODS FreeForm ND Server is a FreeForm ND application that converts data *on receiving a client request for that data*, and not before. Data served by the DODS FreeForm ND Server remains in its original format.

➨ The DODS FreeForm ND Server does not produce an output file containing the converted data, but serves it directly over the network to the DODS client. Therefore, the DODS FreeForm ND Server ignores the output section of the format definition file.

➨ To sample a data file, you do not write format definitions that cause the FreeForm ND engine to sample the data file. Instead, you add a DODS "constraint expression" to the URL that the client sends to the DODS FreeForm ND Server.

➨ The DODS FreeForm ND Server performs data conversion on the fly. Conversion only takes place when the client sends a URL requesting data from the DODS FreeForm ND Server.

➨ Unlike FreeForm ND, there is no static file created by the conversion. (If you wish to create or work with such a file, use the FreeForm ND utilities, such as `newform`.)

6.2 Data Type Conversions

The DODS FreeForm ND Server performs data conversions, based on the data it receives from the FreeForm ND engine. Note that DODS does not recommend the use of `int64` and `uint64` in the format definition file.

Table 6.1: DODS Data Type Conversions

FreeForm ND	**DODS**
`text`	`String`
`int8`	`Byte`
`int16`, `int32`, `int64`	`Int32`
`uint8`, `uint16`, `uint32`, `uint64`	`UInt32`
`float32`, `float64`, `enote`	`Float64`

Table 6.2: Basic Data Type Conversions

6.2.1 Conversion Examples

The examples show how the DODS FreeForm ND Server treats data received from the FreeForm ND engine. Please see the DODS FreeForm ND Server distribution for more test data and format definition files, and the Chapter **??** for more information on writing format definitions.

Arrays

If you define a variable as an array in the FreeForm ND format definition file, the DODS FreeForm ND Server produces an array of variables with matching types.

For exmple, this entry in the format definition file:

```
binary_input_data "array"
fvar1 1 4 ARRAY["records" 1 to 101] of int32 0
```

in converted by the DODS FreeForm ND Server to:

```
Int32 fvar1[records = 101]
```

Collections of Variables

If you define several variables in the format definition file, the DODS FreeForm ND Server produces a Sequence of variables with matching types.

For example, this entry in the format definition file:

```
ASCII_input_data "ASCII_data"
fvar1   1 10  int32 2
svar1  13 18  int16 0
usvar1 21 26 uint16 1
lvar1  29 39  int32 0
ulvar1 42 52 uint32 4
```

is converted by the DODS FreeForm ND Server to:

```
Sequence {
  Int32 fvar1;
  Int32 svar1;
  ...
} ASCII_data;
```

Multiple Arrays

If you define a collection of arrays in the format definition file, as you would expect, the DODS FreeForm ND Server produces a dataset containing multiple arrays.

For example, this entry in the format definition file:

```
binary_input_data "arrays"
fvar1 1 4 ARRAY["records" 1 to 101] of int32 0
fvar2 1 4 ARRAY["records" 1 to 101] of int32 0
```

is converted by the DODS FreeForm ND Server to:

```
Dataset {
  Int32 fvar1[records=101]
  Int32 fvar2[records=101]
};
```

7 File Servers

The DODS project uses the DODS FreeForm ND Server to present a catalog of data files to the world as a single dataset. Normally, in the DODS argot, a "dataset" is contained in a single file on a disk. However, this paradigm is often broken by large datasets that may contain many thousands or tens of thousands of data files. The DODS file server is a way to make these dicrete datasets appear to be a single large dataset.

The DODS file server is a DODS server that returns a URL or set of URLs in response to a query containing selection variables. For example, a dataset organized by date and geographic location might provide a file server that allowed you to query the dataset with a range of dates and longitudes. This fileserver would return a list of one or more URLs corresponding to files within that dataset that fell within the given range.

7.1 The Problem

Consider the following (imaginary) list of files:

```
1997360.nc  1998001.nc  1998007.nc  1998013.nc ...
1997361.nc  1998002.nc  1998008.nc  1998014.nc
1997362.nc  1998003.nc  1998009.nc  1998015.nc
1997363.nc  1998004.nc  1998010.nc  1998016.nc
1997364.nc  1998005.nc  1998011.nc  1998017.nc
1997365.nc  1998006.nc  1998012.nc  1998018.nc
```

These appear to be a set of netCDF files, arranged by date[1]

If you want data from the first week of January, 1998, it is fairly clear which files to request. However, a basic DODS server provides no way to request data from more than one file, so your request would have to be split into 7 different requests, from `1998001.nc` to `1998007.nc`. This could be represented as a set of seven DODS URLs:

```
http://dods/cgi-bin/nph-nc/1998001.nc
http://dods/cgi-bin/nph-nc/1998002.nc
http://dods/cgi-bin/nph-nc/1998003.nc
http://dods/cgi-bin/nph-nc/1998004.nc
http://dods/cgi-bin/nph-nc/1998005.nc
http://dods/cgi-bin/nph-nc/1998006.nc
http://dods/cgi-bin/nph-nc/1998007.nc
```

But what if you then uncover another similar dataset whose data you want to compare to the first? Or what if you want to expand the inquiry to cover the entire year? Keeping track of this many URLs will quickly become burdensome.

What's more, another similar dataset could be arranged in two different directories, `1997` and `1998`, each with files:

```
001.nc
002.nc
003.nc
...
```

and so on. Now you have to keep track of two large sets of URLs, in two different forms. But you could also imagine files called:

```
0011998.nc
0021998.nc
0031998.nc
```

or

[1] A serial date, with a year and the day of the year, expressed in an ordinal number from 1 to 365 or 366.

```
00198.nc
00298.nc
00398.nc
```

or

```
1Jan98.nc
2Jan98.nc
3Jan98.nc
```

That is, the number of possible sensible arrangements may not, in fact, be infinite, but it may seem that way to a scientist who is simply trying to find data.

7.2 The DODS Solution

To create a system that allows data providers to assert a degree of uniformity over wildly variable dataset organizations, DODS provides for the installation of a DODS *file server* . The file server is a DODS dataset, containing associations between the names of files within a dataset and some “selectable” data values.

7.2.1 Selectable Data

The concept of *selectable data* requires some explanation. This is used to indicate the data variables you might ordinarily use to narrow your search for data in the first pass at a dataset.

For geophysical data, the selectable data is often the time and location of the data, since typical searches for data often begin by specifying a part of the globe that bears examining, or a date of some event. For other types of data, other data variables will seem more appropriate. Model data, for example, which has no real location or time, might be arranged by the parameters that varied between runs.

A comprehensive definition of selectable data has so far eluded the DODS group, but there are some guidelines, albeit fairly vague ones:

- The selectable data is generally *not* recorded within each data file. However, the selectable data may often include a *range* summarizing some of the data within each file.
- The selectable data should help a user decide whether a particular data file in a dataset is useful. A temperature range might not be as useful as a time range, since data searches more often start with time. (Both would presumably be still more useful, but there is a trade-off between the utility of the file server and the time spent maintaining it.)

7.2.2 What It Looks Like

Consider again the set of data files shown in Section 7.1 on page 52. We could associate each one of these files with a date, and this would provide the rudiments of a file server if we then serve that data with a DODS server such as the DODS FreeForm ND Server.

```
1997/360 http://dods/cgi-bin/nph-nc/1997360.nc
1997/361 http://dods/cgi-bin/nph-nc/1997361.nc
1997/362 http://dods/cgi-bin/nph-nc/1997362.nc
1997/363 http://dods/cgi-bin/nph-nc/1997363.nc
1997/364 http://dods/cgi-bin/nph-nc/1997364.nc
1997/365 http://dods/cgi-bin/nph-nc/1997365.nc
1998/001 http://dods/cgi-bin/nph-nc/1998001.nc
1998/002 http://dods/cgi-bin/nph-nc/1998002.nc
1998/003 http://dods/cgi-bin/nph-nc/1998003.nc
1998/004 http://dods/cgi-bin/nph-nc/1998004.nc
1998/005 http://dods/cgi-bin/nph-nc/1998005.nc
1998/006 http://dods/cgi-bin/nph-nc/1998006.nc
1998/007 http://dods/cgi-bin/nph-nc/1998007.nc
1998/008 http://dods/cgi-bin/nph-nc/1998008.nc
1998/009 http://dods/cgi-bin/nph-nc/1998009.nc
1998/010 http://dods/cgi-bin/nph-nc/1998010.nc
```

This list represents a set of DODS URLs, each identified by a date, given as a year and a serial day. The files appear to be netCDF format files, served by a DODS netCDF server, but that is not important for this discussion.

To use the DODS FreeForm ND Server for your file server, you could use a format description file with an input section like this:

```
ASCII_input_data "File Server Example Input"
year 1 4 short 0
serial_day 6 8 short 0
DODS_Url 10 46 char 0
```

8 FreeForm ND Conventions

File name conventions have been defined for FreeForm ND. If you follow these conventions, FreeForm ND can locate format files through a default search sequence. Using the file name conventions also lets you reduce the number of arguments on the command line. In addition to standard file names, FreeForm ND programs recognize various standard command line arguments.

8.1 File Name Conventions

Naming conventions have been established for files accessed by FreeForm ND. Although you are not required to follow these conventions, using them lets you enter abbreviated commands when you are using FreeForm ND-based programs. FreeForm ND can then automatically execute several operations:

- Determination of input and output formats when they are not explicitly identified in the relevant format descriptions in format files
- Location of format files when they are not specified on the command line

8.2 File Name Extensions

The expected extensions for data files are as follows:

`.dat` For ASCII, e.g., `latlon.dat`

`.dab` For dBASE, e.g., `latlon.dab`

`.bin` binary or anything that is not `.dat` or `.dab`, e.g., `latlon.bin`

The expected extension for format description files is `.fmt`, e.g., `latlon.fmt`. You should not use mixed case extensions for format description files if you want to take advantage of FreeForm ND's default search capabilities. If you explicitly specify the names of format description files on the command line, you can use mixed case extensions.

NOTE: Previous versions of FreeForm ND used variable description files (formerly called format specification files) each of which contained variable descriptions for one file. Expected extensions for these files were `.afm` (ASCII), `.bfm` (binary), and `.dfm` (dBASE). Variable descriptions for one or more files can now be incorporated into a single format description file. It is recommended that you convert and combine (as appropriate) existing variable description files into format description files.

8.3 File Name Relationships

FreeForm ND-based programs expect certain relationships between data file and format description file names as outlined below.

- The data file is named datafile.ext where datafile is the file name of your choosing and ext is the extension.

  Example: `latlon.dat`

- The corresponding format description file should be named datafile.fmt.

  Example: `latlon.fmt`

- If one format description file is used for multiple data files, all with the same extension, the format description file should be named ext.fmt.

  Example: `ll.fmt` is the format description file for `lldat1.ll`, `lldat2.ll`, and `lldat3.ll`.

Again, although not required, it is to your advantage to use these conventions.

8.4 Determining Input and Output Formats

You can optionally include the read/write type ("`input`" or "`output`") in format descriptors, e.g., `ASCII_input_data`. You may not want to specify the read/write type in some circumstances. For example, you may need to translate from native ASCII to binary, then back to ASCII. ASCII is the input format in the first translation and the output format in the second translation, vice versa for binary. You would need to edit the format description file before executing the second translation if you included read/write type in the format descriptors.

> **NOTE:** If you use the -ft option, you do not need to edit the format description file. See Section 8.6.2 on page 61 later in this chapter.

If you do not specify read/write type, FreeForm ND can nevertheless determine which format in a format description file is input and which is output as long as you have adhered to FreeForm ND filenaming conventions.

- If the input format is not specified, and
  - the input data filename extension is `.bin`, assume binary input.
  - the input data filename extension is `.dab`, assume dBASE input.
  - the input data filename extension is `.dat`, assume ASCII input.
  - the input data filename extension is anything else, assume binary input.
- If the output format is not specified, and
  - the input format is binary, the output is ASCII or dBASE, whichever is found first.
  - the input format is dBASE, the output is ASCII or binary, whichever is found first.
  - the input format is ASCII, the output is binary or dBASE, whichever is found first.

> **NOTE:** The appropriate format descriptions must be in the format description file(s) used by FreeForm ND for a translation. If, for example, FreeForm ND determines the input format is binary and the output format is ASCII, there must be a format description for each type.

The checkvar program needs only an input format.

8.5 Locating Format Files

FreeForm ND programs use the following search sequence to find a format file (format or variable description file) for the data file datafile.ext when the format file name is not explicitly specified on the command line. In summary, FreeForm ND searches the directory specified by the GeoVu keyword `format_dir` (defined in a equivalence table or in the environment), the current or working directory, and the data file's home directory. The rules are applied in the order given below until a format file is found or all rules have been exhausted. If the relevant format file does not follow FreeForm ND conventions for name or location, it should be explicitly specified on the command line.

> **NOTE:** GeoVu is a FreeForm ND-based application for data access and visualization. FreeForm ND applications other than GeoVu use GeoVu keywords.

For information about equivalence tables, see the GeoVu Tools Reference Guide, available from the NGDC.

8.5.1 Search Sequence

❶ Search the directory given by the GeoVu keyword `format_dir` for a format description file named datafile.fmt.

❷ Search the directory given by the GeoVu keyword `format_dir` for variable description files named `datafile.afm`, `datafile.bfm`, and `datafile.dfm`.

> **NOTE:** Step 2 is included to accommodate variable description files that were created using previous versions of FreeForm ND. It is recommended that you convert existing variable description files to format description files.

❸ Search the directory given by the GeoVu keyword `format_dir` for a format description file named ext.fmt.

If the GeoVu keyword `format_dir` is not found, FreeForm ND continues the search for a format file as follows.

❹ Search the current (default) directory for a format description file named datafile.fmt.

❺ Search the current directory for variable description files named `datafile.afm`, `datafile.bfm`, and `datafile.dfm`. Use the criteria in step 2 for determining input and output format files.

❻ Search the current directory for a format description file named `ext.fmt`.

If the data file's home directory is not the same as the current directory, FreeForm ND continues the search for a format file with steps 7-9. The data file's home directory is given by the directory path component of the data file name. If the data file name has no directory path component, the home directory search is not done.

❼ Search the data file's home directory for a format description file named `datafile.fmt`.

❽ Search the data file's home directory for variable description files named `datafile.afm`, `datafile.bfm`, and `datafile.dfm`. Use the criteria in step 2 for determining input and output format files.

❾ Search the data file's home directory for a format description file named `ext.fmt`.

8.5.2 Case Sensitivity

FreeForm ND adheres to the following rules for case sensitivity (in applicable operating systems) when it searches for a format file for the data file datafile.ext.

➨ FreeForm ND preserves the case of datafile, for example, the default format file for the data file `LATLON.BIN` is `LATLON.fmt` (or `LATLON.bfm`).

➨ FreeForm ND searches for a format file with a lower case extension. That is, the format file must have its extension in lower case no matter what the case of datafile. For example, the default format file for the data file `LatLon.dat` is `LatLon.fmt` (or `LatLon.afm`), and `TIMEDATE.fmt` (or `TIMEDATE.bfm`) is the default format file for `TIMEDATE.bin`.

➨ In searching for a format description file of type `ext.fmt`, FreeForm ND preserves the case of ext. For example, for data files named `lldat1.LL`, `lldat2.LL`, and `latlon3.LL`, the default format description file is `LL.fmt`.

8.6 Command Line Arguments

FreeForm ND programs can take various command line arguments. The most widely used or standard arguments are discussed in this section. They are used for several different purposes: identifying input and output files, identifying format files and titles, changing run-time operation parameters, and defining data filters.

The only required argument for any FreeForm ND program is the name of the input file or file to be processed. All other arguments are optional and can be in any order following the input file name. The command line of a FreeForm ND program with the standard arguments has the following form:

> application_name *input_file* [`-f` *format_file*] [`-if` *input_format_file*]
> [`-of` *output_format_file*] [`-ft "`*title*`"`] [`-ift "`*title*`"`] [`-oft "`*title*`"`]
> [`-b` *local_buffer_size*] [`-c` *count*] [`-v` *var_file*] [`-q` *query_file*] [`-o` *output_file*]

> **NOTE:** To see a summary of command line usage for a FreeForm ND program, enter the program's name on the command line without any arguments.

8.6.1 Specifying Input and Output Files

input_file Name of the file to be processed. Following FreeForm ND naming conventions, the standard extensions for data files are `.dat` for ASCII format, `.bin` for binary, and `.dab` for dBASE.

`-o` ***output_file*** Option flag followed by the name of the output file. The standard extensions are the same as for input files.

8.6.2 Specifying Format Description Source

FreeForm ND offers a number of command line options for specifying the source of the format descriptions that a program must find in order to process data. The proper option or combination of options to use depends on how you have constructed your format files.

`-f` ***format_file*** Option flag followed by the name of the format description file describing both input and output data.

`-if` ***input_format_file*** Option flag followed by the name of the format description file describing the input data. Also use this option for an input variable description file written using earlier versions of FreeForm ND.

`-of` ***output_format_file*** Option flag followed by the name of the format description file describing the output data. Also use this option for an output variable description file written using earlier versions of FreeForm ND.

`-ft "`***title***`"` Option flag followed by the title (enclosed in quotes) of the format to be used for both input and output data, in which case there is no reformatting. The title follows format type on the first line of a format description in a format description file.

`-ift "`***title***`"` Option flag followed by the title (enclosed in quotes) of the desired input format.

`-oft "`***title***`"` Option flag followed by the title (enclosed in quotes) of the desired output format.

> **NOTE:** Previous versions of FreeForm ND used variable description files (`.afm`, `.bfm`, `.dfm`). It is recommended that you convert and combine (as appropriate) existing variable description files into format description files.

The various options available for specifying the source of a format description offer you a great deal of flexibility-in naming files, setting up format description files, and on the command line. In using these options, you need to consider the content of your format description files and how FreeForm ND will interpret the arguments on the command line.

8.6.3 Changing Run-time Parameters

FreeForm ND includes three arguments that let you change run-time parameters according to your needs. One argument lets you specify local buffer size, another indicates the number of records to process, and the third indicates which variables to process.

`-b` ***local_buffer_size*** Option flag followed by the size of the memory buffer used to process the data and format files.

Default buffer size is 32,768. You many want to decrease the buffer size if you are running with low memory. Keep in mind that too small a buffer may result in unexpected behavior.

`-c` ***count*** Option flag followed by a number that specifies how many data records at the head or tail of the file to process.

If $count > 0$, *count* records at the beginning of the file are processed.

If $count < 0$, *count* records at the tail or end of the file are processed.

`-v` ***var_file*** Option flag followed by the name of a variable file. The file contains names of the variables in the input data file to be processed by the FreeForm ND program. Variable names in var˙file can be separated by one or more spaces or each name can be on a separate line.

8.6.4 Defining Filters

The query option lets you define data filters via a query file so you can precisely specify which data to process. The FreeForm ND program will process only those records meeting the query criteria.

`-q` ***query_file*** Option flag followed by the name of the file containing query criteria. See Chapter **??** for a complete description of the query syntax.

9

Format Conversion

The FreeForm ND utility program newform lets you convert data from one format to another. This allows you to pass data to applications in the format they require. You may also want to create binary archives for efficient data storage and access. With newform, conversion of ASCII data to binary format is straightforward. If you wish to read the data in a binary file, you can convert it to ASCII with newform, or use the interactive program readfile. You can also convert data from one ASCII format to another ASCII format with newform.

9.1 newform

The FreeForm ND-based program `newform`[1] is a general tool for changing the format of a data file. The only required command line argument, if you use FreeForm ND naming conventions, is the name of the input data file. The reformatted data is written to standard output (the screen) unless you specify an output file. If you reformat to binary, you will generally want to store the output in a file.

You must create a format description file (or files) with format descriptions for the data files involved in a conversion before you can use `newform` to perform the conversion. The standard extension for format description files is `.fmt`. If you do not explicitly specify the format description file on the command line, which is unnecessary if you use FreeForm ND naming conventions, `newform` follows the FreeForm ND search sequence to find a format file.

For details about FreeForm ND naming conventions and the search sequence, see Chapter 8.

The `newform` command has the following form:

> `newform` *input_file* [`-f` *format_file*] [`-if` *input_format_file*] [`-of` *output_format_file*]
> [`-ft "`*title*`"`] [`-ift "`*title*`"`] [`-oft "`*title*`"`] [`-b` *local_buffer_size*] [`-c` *count*]
> [`-v` *var_file*] [`-q` *query_file*] [`-o` *output_file*]

For descriptions of the arguments, see Section 8.6 on page 61.

If you want to convert an ASCII file to a binary file, and you follow the FreeForm ND naming conventions, the command is simply:

```
newform datafile.dat -o datafile.bin
```

where datafile is the file name of your choosing.

If data files and format files are not in the current directory or in the same directory, you can specify the appropriate path name. For example, if the input data file is not in the current directory, you can enter:

```
newform /path/datafile.dat -o datafile.bin
```

To read the data in the resulting binary file, you can reformat back to ASCII using the command:

```
newform datafile.bin -o datafile.ext
```

or you can use the `readfile` program, described in Section 9.3 on page 69.

[1] There is a Solaris utility called `newform`, supplied with the operating system. If you use a Solaris machine, you may want to change the name of the FreeForm ND tool to `newfor` or something similar, to avoid name collisions.

9.2 chkform

Though `newform` is useful for checking data formats, it is limited by requiring a format file to specify an output format. Since some FreeForm ND applications (such as the DODS FreeForm Server) do not require an output format, this is extra work for the dataset administrator. For these occasions, FreeForm ND provides a simpler format-checking program, called `chkform`.

The `chkform` program attempts to read an ASCII file, using the specified input format. If the format allows the file to be read properly, `chkform` says so. However, if the input format contains errors, or does not accurately reflect the contents of the given data file, `chkform` delivers an error message, and attempts to provide a rudimentary diagnosis of the problem.

You must create a format description file (or files) with format descriptions for the data files involved before you can use `chkform` to chack the format. As with `newform`, the standard extension for format description files is `.fmt`. If you do not explicitly specify the format description file on the command line (unnecessary if you use FreeForm ND naming conventions) `chkform` follows the FreeForm ND search sequence to find a format file.

For details about FreeForm ND naming conventions and the search sequence, see Chapter 8.

The `chkform` command has the following form:

`chkform` *input_file* [`-if` *input_format_file*] [`-ift` "*title*"] [`-b` *local_buffer_size*]
[`-c` *count*] [`-q` *query_file*] [`-ol` *log_file*] [`-el` *error_log_file*] [`-ep`]

Most of the arguments are described in Section 8.6 on page 61. The following are specific to `chkform`:

`-ol` ***log_file*** Puts a log of processing information into the specified *log_file*.

`-el` ***error_log_file*** Creates an error log file that contains whatever error messages are issued by `chkform`.

`-ep` In normal operation, `chkform` asks you to manually acknowledge each important error by typing something on the keyboard. If you use this option, `chkform` will not stop to prompt, but will continue processing until either the file is procesed, or there is an error preventing more processing.

As in the above examples, if you have an ASCII data file called `datafile.dat`, supposedly described in a format file called `datafile.fmt`, you can use `chkform` like this:

```
chkform datafile.dat
```

If processing is successful, you will see something like the following:

```
Welcome to Chkform release 4.2.3 -- an NGDC FreeForm ND application


(llmaxmin.fmt) ASCII_input_file_header  "Latitude/Longitude Limits"
File llmaxmin.dat contains 1 header record (71 bytes)
Each record contains 6 fields and is 71 characters long.

(llmaxmin.fmt) ASCII_input_data "lat/lon"
File llmaxmin.dat contains 10 data records (230 bytes)
Each record contains 3 fields and is 23 characters long.

100% processed     Elapsed time - 00:00:00

No errors found (11 lines checked)
```

Table 9.1: The `readfile` program options

`c`	char — 1 byte character
`s`	short — 2 byte signed integer
`l`	long — 4 byte signed integer
`f`	float — 4 byte single-precision floating point
`d`	double — 8 byte double-precision floating point
`uc`	uchar — 1 byte unsigned integer
`us`	ushort — 2 byte unsigned integer
`ul`	ulong — 4 byte unsigned integer
`b`	Toggle between “big-endian” and your machine’s native byte order
`p`	Set new file position
`P`	Show present file position and length
`h`	Display this help screen
`q`	Quit

9.3 `readfile`

FreeForm ND includes **`readfile`**, a simple interactive binary file reader. The program has one required command line argument, the name of the file to be read. You do not have to write format descriptions to use **`readfile`**.

The **`readfile`** command has the following form:

```
readfile binary_data_file
```

When the program starts, it shows the available options, shown in table 9.1. At the **`readfile`** prompt, type these option codes to view binary encoded values. (Pressing return repeats the last option.)

The options let you interactively read your way through the specified binary file. The first position in the file is 0. You must type the character(s) indicating variable type (e.g., us for unsigned short) to view each value, so you need to know the data types of variables in the file and the order in which they occur. If successive variables are of the same type, you can press Return to view each value after the first of that type.

You can toggle the byte-order switch on and off by typing b. The byte-order option is used to read a binary data file that requires byte swapping. This is the case when you need cross-platform access to a file that is not byte-swapped, for example, if you are on a Unix machine reading data from a CD-ROM formatted for a PC. When the switch is on, type s or l to swap short or long integers respectively, or type f or d to swap floats or doubles. The `readfile` program does not byte swap the file itself (the file is unchanged) but byte swaps the data values internally for display purposes only.

To go to another position in the file, type p. You are prompted to enter the new file position in bytes. If, for example, each value in the file is 4 bytes long and you type 16, you will be positioned at the first byte of the fifth value. If you split fields (by not repositioning at the beginning of a field), the results will probably be garbage. Type P to find out your current position in the file and total file length in bytes. Type q to exit from `readfile`.

You can also use an input command file rather than entering commands directly. In that case, the `readfile` command has the following form:

```
readfile binary_data_file < input_command_file
```

9.4 Creating a Binary Archive

By storing data files in binary, you save disk space and make access by applications more efficient. An ASCII data file can take two to five times the disk space of a comparable binary data file. Not only is there less information in each byte, but extra bytes are needed for decimal points, delimiters, and end-of-line markers.

It is very easy to create a binary archive using `newform` as the following examples show. The input data for these examples are in the ASCII file `latlon.dat` (shown below). They consist of 20 random latitude and longitude values. The size of the file on a Unix system is 460 bytes.

Here is the `latlon.dat` file:

```
-47.303545 -176.161101
 -0.928001    0.777265
-28.286662   35.591879
 12.588231  149.408117
-83.223548   55.319598
 54.118314 -136.940570
 38.818812   91.411330
-34.577065   30.172129
 27.331551 -155.233735
 11.624981 -113.660611
 77.652742  -79.177679
 77.883119  -77.505502
-65.864879  -55.441896
-63.211962  134.124014
 35.130219 -153.543091
 29.918847  144.804390
-69.273601   38.875778
-63.002874   36.356024
 35.086084  -21.643402
-12.966961   62.152266
```

9.4.1 Simple ASCII to Binary Conversion

In this example, you will use `newform` to convert the ASCII data file `latlon.dat` into the binary file `latlon.bin`. The input and output data formats are described in `latlon.fmt`.

Here is the `latlon.fmt` file:

```
/ This is the format description file for data files latlon.bin
/ and latlon.dat. Each record in both files contains two fields,
/ latitude and longitude.

binary_data "binary format"
latitude 1 8 double 6
longitude 9 16 double 6

ASCII_data "ASCII format"
latitude 1 10 double 6
longitude 12 22 double 6
```

The binary and ASCII variables both have the same names. The binary variable latitude occupies positions 1 to 8 and longitude occupies positions 9-16. The corresponding ASCII variables occupy positions 1-10 and 12-22. Both the binary and ASCII variables are stored as doubles and have a precision of 6.

9.4.2 Converting to Binary

To convert from an ASCII representation of the numbers in `latlon.dat` to a binary representation:

❶ Change to the directory that contains the FreeForm ND example files.

❷ Enter the following command:

```
newform latlon.dat -o latlon.bin
```

Because FreeForm ND filenaming conventions have been used, `newform` will locate and use `latlon.fmt` for the translation. The `newform` program creates a new data file (effectively a binary archive) called `latlon.bin`. The size of the archive file is 2/3 the size of `latlon.dat`. Additionally, the data do not have to be converted to machine-readable representation by applications.

There are two methods for checking the data in `latlon.bin` to make sure they converted correctly. You can reformat back to ASCII and view the resulting file, or use `readfile` to read `latlon.bin`.

9.4.3 Reconverting to Native Format

Use the following `newform` command to reformat the binary data in `latlon.bin` to its native ASCII format:

```
newform latlon.bin -o latlon.rf
```

The ASCII file `latlon.rf` matches (but does not overwrite) the original input file `latlon.dat`. You can confirm this by using a file comparison utility. The `diff` command is generally available on Unix platforms.

To use `diff` to compare the `latlon` ASCII files, enter the command:

```
diff latlon.dat latlon.rf
```

The output should be something along these lines:

```
Files are effectively identical.
```

Several implementations of the `diff` utility don't print anything if the two input files are identical.

> **NOTE:** The `diff` utility may detect a difference in other similar cases because FreeForm ND adds a leading zero in front of a decimal and interprets a blank as a zero if the field is described as a number. (A blank described as a character is interpreted as a blank.)

9.4.4 Reading the Binary File

To use `readfile` to read the data in `latlon.bin`:

❶ Enter the following command:

```
readfile latlon.bin
```

❷ The data are stored as doubles, so enter d to view each value (or press Return to view each value after the first).

❸ Enter q to quit `readfile`.

9.4.5 Conversion to a More Portable Binary

In this example, you will use `newform` to reformat the latitude and longitude values in the ASCII data file `latlon.dat` into binary longs in the binary file `latlon2.bin`. The input and output data formats are described in `latlon2.fmt`.

This is what's in `latlon2.fmt`:

```
/ This is the format description file for data files latlon.dat
/ and latlon2.bin. Each record in both files contains two fields,
/ latitude and longitude.

ASCII_data "ASCII format"
latitude 1 10 double 6
longitude 12 22 double 6

binary_data "binary format"
latitude 1 4 long 6
longitude 5 8 long 6
```

The ASCII and binary variables both have the same names. The ASCII variable latitude occupies positions 1-10 and longitude occupies positions 12-22. The ASCII variables are defined to be of type double. The binary variables occupy four bytes each (positions 1-4 and 5-8) and are of type long. The precision for all is 6.

9.4.6 Converting to Binary Long

In the previous example, both the ASCII and binary variables were defined to be doubles. Binary longs, which are 4-byte integers, may be more portable across different platforms than binary doubles or floats.

To convert the ASCII data in `latlon.dat` to binary longs:

❶ Change to the directory that contains the FreeForm ND example files.

❷ Enter the following command:

```
newform latlon.dat -f latlon2.fmt -o latlon2.bin
```

It creates the binary archive file `latlon2.bin` with the 20 latitude and longitude values in `latlon.dat` stored as binary longs.

This example duplicates one in chapter 2. If you completed that example, an error message will indicate that `latlon2.bin` exists. You can rename, move, or delete the existing file.

The size of the archive file `latlon2.bin` is about 1/3 the size of `latlon.dat`. Also, the data do not have to be converted to machine representation by applications. The main tradeoff in achieving savings in space and access time is that although binary longs are more portable than binary doubles or floats, any binary representation is less portable than ASCII.

> **CAUTION:** There may be a loss of precision when input data of type double is converted to long.

9.4.7 Reading the Binary File

Once again, you can use `readfile` to check the data in the binary archive you created.

❶ Enter the following command:

```
readfile latlon2.bin
```

❷ The data are stored as longs, so enter l to view each value (or press Return to view each value after the first).

❸ Enter q to quit `readfile`.

If desired, you can enter the commands to `readfile` from an input command file rather than directly from the command line. The example command file `latlon.in` is shown next.

Here is `latlon.in`:

```
llllllp0 llPq
```

The 6 l’s (l for `long`) cause the first 6 values in the file to be displayed. The sequence p0 causes a return to the top (position 0) of the file. A position number (0) must be followed by a blank space. The 2 l’s display the first two values again. The P displays the current file position and length, and q closes `readfile`.

If you enter the following command:

```
readfile latlon2.bin < latlon.in
```

you should see the following output on the screen:

```
long:  -47303545
long: -176161101
long:    -928001
long:     777265
long:  -28286662
long:   35591879
New File Position = 0
long:  -47303545
long: -176161101
File Position: 8       File Length: 160
```

The floating point numbers have been multiplied by 106, the precision of the long variables in `latlon2.fmt`.

9.4.8 Including a Query

You can use the query option (`-q` *query_file*) to specify exactly which records in the data file `newform` should process. The query file contains query criteria. Query syntax is summarized in Appendix C.

In this example, you will specify a query so that `newform` will reformat only those value pairs in `latlon.dat` where latitude is positive and longitude is negative into the binary file `llposneg.bin`. The input and output data formats are described in `latlon2.fmt`.

The query criteria are specified in the following file, called `llposneg.qry`:

```
[latitude] > 0 & [longitude] < 0
```

To convert the desired data in `latlon.dat` to binary and then view the results:

❶ Enter the following command:

```
newform latlon.dat -f latlon2.fmt -q llposneg.qry
       -o llposneg.bin
```

The `llposneg.bin` file now contains the positive/negative latitude/longitude pairs in binary form.

❷ To view the data, first convert the data in `llposneg.bin` back to ASCII format:

newform llposneg.bin -f latlon2.fmt -o llposneg.dat

❸ Enter the appropriate command to display the data in `llposneg.dat`, e.g. `more`:

The following output appears on the screen:

```
54.118314 -136.940570
27.331551 -155.233735
11.624981 -113.660611
77.652742  -79.177679
77.883119  -77.505502
35.130219 -153.543091
35.086084  -21.643402
```

NOTE: As demonstrated in the examples above, you can check the data in a binary file either by using `readfile` or by converting the data back to ASCII using `newform` and then viewing it.

9.5 File Names and Context

In the preceding examples, the read/write type (input or output) was not included in the format descriptors (`ASCII_data` and `binary_data`). FreeForm ND naming conventions were used, so `newform` can determine from the context which format should be used for input and which for output. Consider the command:

```
newform latlon.dat -o latlon.bin
```

The input file extension is `.dat` and the output file extension is `.bin`. These extensions provide context indicating that ASCII should be used as the input format and binary should be used as the output format. The format description file that `newform` will look for is the file with the same name as the input file and the extension `.fmt`, i.e., `latlon.fmt`.

If you use the following command:

```
newform latlon.bin
```

to translate the binary archive `latlon.bin` back to ASCII, `newform` identifies the input format as binary and uses the ASCII format for output. The ASCII data is written to the screen because an output file was not specified.

For information about FreeForm ND file name conventions, see Chapter 8.

9.5.1 "Nonstandard" Data File Names

If you are working with data files that do not use FreeForm ND naming conventions, you need to more explicitly define the context. For example, the files `lldat1.ll`, `lldat2.ll`, `lldat3.ll`, `lldat4.ll`, and `lldat5.ll` all have latitude and longitude values in the ASCII format given in the format description file `lldat.fmt`. If you wanted to archive these files in binary format, you could not use a command of the form used in the previous examples, i.e., `newform datafile.dat` -o `datafile.bin` with `datafile.fmt` as the default format description file.

First, the ASCII data files do not have the extension `.dat`, which identifies them as ASCII files. Second, you would need five separate format description files, all with the same content: `lldat1.fmt`, `lldat2.fmt`, `lldat3.fmt`, `lldat4.fmt`, and `lldat5.fmt`. Creating the format description file `ll.fmt` solves both problems.

Here is the `ll.fmt` file:

```
/ This is the format description file that describes latlon
/ data in files with the extension .ll

ASCII_input_data "ASCII format for .ll latlon data"
latitude 1 10 double 6
longitude 12 22 double 6

binary_output_data "binary format for .ll latlon data"
latitude 1 4 long 6
longitude 5 8 long 6
```

The name used for the format description file, `ll.fmt`, follows the FreeForm ND convention that one format description file can be utilized for multiple data files, all with the same extension, if the format description file is named `ext.fmt`. Also, the read/write type (input or output) is made explicit by including it in the format descriptors `ASCII_input_data` and `binary_output_data`. This provides the context needed for FreeForm ND programs to determine which format to use for input and which for output.

Use the following commands to produce binary versions of the ASCII input files:

```
newform lldat1.ll -o llbin1.ll
newform lldat2.ll -o llbin2.ll
newform lldat3.ll -o llbin3.ll
newform lldat4.ll -o llbin4.ll
newform lldat5.ll -o llbin5.ll
```

If you want to convert back to ASCII, you can switch the words input and output in the format description file `ll.fmt`. You could then use the following commands to convert back to native ASCII format with output written to the screen:

```
newform llbin1.ll
newform llbin2.ll
newform llbin3.ll
newform llbin4.ll
newform llbin5.ll
```

It is also possible to convert back to ASCII without switching the read/write types input and output in `ll.fmt`. You can specify input and output formats by title instead. In this case, you want to use the output format in `ll.fmt` as the input format and the input format in `ll.fmt` as the output format. Use the following command to convert `llbin1.ll` back to ASCII:

```
newform llbin1.ll -ift "binary format for .ll latlon data"
       -oft "ASCII format for .ll latlon data"
```

Notice that `newform` reports back the read/write type actually used. Since `ASCII_input_data` was used as the output format, `newform` reports it as `ASCII_output_data`.

Now assume that you want to convert the ASCII data file `llvals.asc` (not included in the example file set) to the binary file `latlon3.bin`, and the input and output data formats are described in `latlon.fmt`. The data file names do not provide the context allowing `newform` to find `latlon.fmt` by default, so you must include all file names on the command line:

```
newform llvals.asc -f latlon.fmt -o latlon3.bin
```

9.5.2 “Nonstandard” Format Description File Names

If you are using a format description file that does not follow FreeForm ND file naming conventions, you must include its name on the command line. Assume that you want to convert the ASCII data file `latlon.dat` to the binary file `latlon.bin`, and the input and output data formats are both described in `llvals.frm` (not included in the example file set). The data file names follow FreeForm ND conventions, but the name of the format description file does not, so it will not be located through the default search sequence. Use the following command to convert to binary:

```
newform latlon.dat -f llvals.frm -o latlon.bin
```

Suppose now that the input format is described in `latlon.fmt` and the output format in `llvals.frm`. You do not need to explicitly specify the input format description file because it will be located by default, but you must specify the output format description file name. In this case, the command would be:

```
newform latlon.dat -of llvals.frm -o latlon.bin
```

You can always unambiguously specify the names of format description files and data files, whether or not their names follow FreeForm ND conventions. Assume you want to look only at longitude values in `latlon.bin` and that you want them defined as integers (longs) which are right-justified at column 30. You will reformat the specified binary data in `latlon.bin` into ASCII data in `longonly.dat` and then view it. The input format is found in `latlon.fmt`, the output format in `longonly.fmt`.

Here is `longonly.fmt`:

```
/ This is the format description file for viewing longitude as an
/ integer value right-justified at column 30.

ASCII_data "ASCII output format, right-justified at 30"
longitude 20 30 long 6
```

In this case, you have decided to look at the first 5 longitude values. Use the following command to unambiguously designate all files involved:

```
newform latlon.bin -if latlon.fmt -of longonly.fmt -c 5
       -o longonly.dat
```

When you view `longonly.dat`, you should see the following 5 values:

```
         1         2         3         4
1234567890123456789012345678901234567890

                    -176161101
                        777265
                      35591879
                     149408117
                      55319598
```

9.6 Changing ASCII Formats

You may encounter situations where a specific ASCII format is required, and your data cannot be used in its native ASCII format. With `newform`, you can easily reformat one ASCII format to another. In this example, you will reformat California earthquake data from one ASCII format to three other ASCII formats commonly used for such data.The file `calif.tap` contains data about earthquakes in California with magnitudes ¿ 5.0 since 1980. The data were initially distributed by NGDC on tape, hence the `.tap` extension. The data format is described in `eqtape.fmt`:

Here is the `eqtape.fmt` file:

```
/ This is the format description file for the NGDC .tap format,
/ which is used for data distributed on floppy disks or tapes.

ASCII_data ".tap format"
source_code 1 3 char 0
century 4 6 short 0
year 7 8 short 0
month 9 10 short 0
day 11 12 short 0
hour 13 14 short 0
minute 15 16 short 0
second 17 19 short 1
latitude_abs 20 24 long 3
latitude_ns 25 25 char 0
longitude_abs 26 31 long 3
longitude_ew 32 32 char 0
depth 33 35 short 0
magnitude_mb 36 38 short 2
MB 39 40 constant 0
isoseismal 41 43 char 0
intensity 44 44 char 0

/ The NGDC record check format includes
/ six flags in characters 45 to 50. These
/ can be treated as one variable to allow
/ multiple flags to be set in a single pass,
/ or each can be set by itself.

ngdc_flags 45 50 char 0
diastrophic 45 45 char 0
tsunami 46 46 char 0
seiche 47 47 char 0
volcanism 48 48 char 0
non_tectonic 49 49 char 0
infrasonic 50 50 char 0

fe_region 51 53 short 0
magnitude_ms 54 55 short 1
MS 56 57 char 0
```

```
z_h 58 58 char 0
cultural 59 59 char 0
other 60 60 char 0
magnitude_other 61 63 short 2
other_authority 64 66 char 0
ide 67 67 char 0
depth_control 68 68 char 0
number_stations_qual 69 71 char 0
time_authority 72 72 char 0
magnitude_local 73 75 short 2
local_scale 76 77 char 0
local_authority 78 80 char 0
```

Three other formats used for California earthquake data are hypoellipse, hypoinverse, and hypo71. Subsets of these formats are described in the format description file `hypo.fmt`. The format descriptions include the parameters required by the AcroSpin program that is distributed as part of the IASPEI Software Library (Volume 2). AcroSpin shows 3D views of earthquake point data.

Here is the `hypo.fmt` file:

```
/ This format description file describes subsets of the
/ hypoellipse, hypoinverse, and hypo71 formats.

ASCII_data "hypoellipse format"
year 1 2 uchar 0
month 3 4 uchar 0
day 5 6 uchar 0
hour 7 8 uchar 0
minute 9 10 uchar 0
second 11 14 ushort 2
latitude_deg_abs 15 16 uchar 0
latitude_ns 17 17 char 0
latitude_min 18 21 ushort 2
longitude_deg_abs 22 24 uchar 0
longitude_ew 25 25 char 0
longitude_min 26 29 ushort 2
depth 30 34 short 2
magnitude_local 35 36 uchar 1

ASCII_data "hypoinverse format"
year 1 2 uchar 0
month 3 4 uchar 0
day 5 6 uchar 0
hour 7 8 uchar 0
minute 9 10 uchar 0
second 11 14 ushort 2
latitude_deg_abs 15 16 uchar 0
latitude_ns 17 17 char 0
latitude_min 18 21 ushort 2
longitude_deg_abs 22 24 uchar 0
longitude_ew 25 25 char 0
longitude_min 26 29 ushort 2
depth 30 34 short 2
magnitude_local 35 36 uchar 1
```

```
number_of_times 37 39 short 0
maximum_azimuthal_gap 40 42 short 0
nearest_station 43 45 short 1
rms_travel_time_residual 46 49 short 2

ASCII_data "hypo71 format"
year 1 2 uchar 0
month 3 4 uchar 0
day 5 6 uchar 0
hour 8 9 uchar 0
minute 10 11 uchar 0
second 12 17 float 2
latitude_deg_abs 18 20 uchar 0
latitude_ns 21 21 char 0
latitude_min 22 26 float 2
longitude_deg_abs 27 30 uchar 0
longitude_ew 31 31 char 0
longitude_min 32 36 float 2
depth 37 43 float 2
magnitude_local 44 50 float 2
number_of_times 51 53 short 0
maximum_azimuthal_gap 54 57 float 0
nearest_station 58 62 short 1
rms_travel_time_residual 63 67 float 2
error_horizontal 68 72 float 1
error_vertical 73 77 float 1
s_waves_used 79 79 char 0
```

The parameters from the California earthquake data in the NGDC format needed for use with the AcroSpin program can be extracted and converted using the following commands:

```
newform calif.tap -if eqtape.fmt -of hypo.fmt
        -oft "hypoellipse format" -o calif.he
newform calif.tap -if eqtape.fmt -of hypo.fmt
        -oft "hypoinverse format" -o calif.hi
newform calif.tap -if eqtape.fmt -of hypo.fmt
        -oft "hypo71 format" -o calif.h71
```

If you develop an application that accesses seismicity data in a particular ASCII format, you need only to write an appropriate format description file in order to convert NGDC data into the format used by the application. This lets you make use of the data that NGDC provides in a format that works for you.

10
Data Checking

The FreeForm ND-based utility program `checkvar` creates variable summary files, lists of maximum and minimum values, and summaries of processing activity. You can use this information to check data quality and to examine the distribution of the data.

10.1 Generating the Summaries

A variable summary file (or list file), which contains histogram information showing the variable's distribution in the data file, is created for each variable (or designated variables) in the specified data file. You can optionally specify an output file in which a summary of processing activity is saved.

Variable summaries (list files) can be helpful for performing quality control checks of data. For example, you could run `checkvar` on an ASCII file, convert the file to binary, and then run `checkvar` on the binary file. The output from `checkvar` should be the same for both the ASCII and binary files. You can also use variable summaries to look at the data distribution in a data set before extracting data.

The `checkvar` command has the following form:

```
checkvar input_file [-f format_file] [-if input_format_file]
    [-of output_format_file] [-ft "title"] [-ift "title"] [-oft "title"]
    [-b local_buffer_size] [-c count] [-v var_file] [-q query_file] [-p precision]
    [-m maxbins] [-md missing_data_flag] [-mm] [-o processing_summary]
```

The `checkvar` program needs to find only an input format description. Output format descriptions will be ignored. If conversion variables are included in input or output formats, no conversion is performed when you run `checkvar`, since it ignores output formats.

For descriptions of the standard arguments (first eleven arguments above), see Section 8.6 on page 61.

`-p` ***precision*** Option flag followed by the number of decimal places. The number represents the power of 10 that data is multiplied by prior to binning. A value of 0 bins on one's, 1 on tenth's, and so on. This option allows an adjustment of the resolution of the `checkvar` output.

The default is 0; maximum is 5.

> **NOTE:** If you use the `-p` option on the command line, the precision set in the relevant format file is overridden. The precision in the format file serves as the default.

`-m` ***maxbins*** Option flag followed by the approximate maximum number of bins desired in `checkvar` output. The `checkvar` program keeps track of the number of bins filled as the data is processed. The smaller the number of bins, the faster `checkvar` runs. By keeping the number of bins small, you can check the gross aspects of data distribution rather than the details.

The number of bins is adjusted dynamically as `checkvar` runs depending on the distribution of data in the input file. If the number of filled bins becomes ¿ 1.5 * maxbins, the width of the bins is doubled to keep the total number near the desired maximum.

The default is 100 bins; minimum is 6. Must be ¡ 10,000.

> **NOTE:** The precision (-p) and maxbins (-m) options have no effect on character variables.

`-md` ***missing_data_flag*** Option flag followed by a flag value that `checkvar` should ignore across all variables in creating histogram data. Missing data flags are used in a data file to indicate missing or meaningless data. If you want `checkvar` to ignore more than one value, use the query (`-q`) option in conjunction with the variable file (`-v`) option.

`-mm` Option flag indicating that only the maximum and minimum values of variables are calculated and displayed in the processing summary. Variable summary files are not created.

`-o` ***processing_summary*** Option flag followed by the name of the file in which summary information displayed during processing is stored.

10.2 Example

You will use `checkvar` with a precision of 3 to create a processing summary file and summary files for the two variables latitude and longitude in the file `latlon.dat`.

Here is `latlon.dat`:

```
-47.303545 -176.161101
 -0.928001    0.777265
-28.286662   35.591879
 12.588231  149.408117
-83.223548   55.319598
 54.118314 -136.940570
 38.818812   91.411330
-34.577065   30.172129
 27.331551 -155.233735
 11.624981 -113.660611
 77.652742  -79.177679
 77.883119  -77.505502
-65.864879  -55.441896
-63.211962  134.124014
 35.130219 -153.543091
 29.918847  144.804390
-69.273601   38.875778
-63.002874   36.356024
 35.086084  -21.643402
-12.966961   62.152266
```

To create the summary files, enter the following command:

```
checkvar latlon.dat -p 3 -o latlon.sum
```

A summary of processing information and the maximum and minimum for each variable are displayed on the screen. The following three files are created:

- `latlon.sum` recaps processing activity, maximums and minimums
- `latitude.lst` shows distribution of the latitude values in `latlon.dat`
- `longitude.lst` shows distribution of the longitude values in `latlon.dat`.

10.3 Interpreting the Summaries

The processing and variable summary files output by `checkvar` from the example in the previous section are shown and discussed below.

10.3.1 Processing Summary

If you specify an output file on the command line, it stores the information that is displayed on the screen during processing. The file `latlon.sum` was specified as the output file in the example above.

Here is `latlon.sum`:

```
Input file : latlon.dat
Requested precision = 3, Approximate number of sorting bins = 100

Input data format       (latlon.fmt)
ASCII_input_data       "ASCII format"
The format contains 2 variables; length is 24.

Output data format       (latlon.fmt)
binary_output_data       "binary format"
The format contains 2 variables; length is 16.

Histogram data precision: 3, Number of sorting bins: 20
latitude: 20 values read
minimum: -83.223548 found at record  5
maximum:  77.883119 found at record 12
Summary file: latitude.lst

Histogram data precision: 3, Number of sorting bins: 20
longitude: 20 values read
minimum: -176.161101 found at record 1
maximum:  149.408117 found at record 4
Summary file: longitude.lst.
```

The processing summary file `latlon.sum` first shows the name of the input data file (`latlon.dat`). If you specified precision and a maximum number of bins on the command line, those values are given as Requested precision, in this case 3, and Approximate number of sorting bins, in this case the default value of 100. If precision is not specified, No requested precision is shown.

A summary of each format shows the type of format (in this case, Input data format and Output data format) and the name of the format file containing the format descriptions (`latlon.fmt`), whether specified on the command line or located through the default search sequence (as detailed in chapter 4). In this case, it was located by default. Since `checkvar` only needs an input format description, it ignores output format descriptions. Next, you see the format descriptor as resolved by FreeForm ND (e.g., `ASCII_input_data`) and the format title (e.g., “ASCII format”). Then the number of variables in a record and total record length are given; for ASCII, record length includes the end-of-line character (1 byte for Unix).

A section for each variable processed by `checkvar` indicates the histogram precision and actual number of sorting bins. Under some circumstances, the precision of values in the histogram file may be different than the precision you specified on the command line. The default value for precision, if none is specified on the command line, is the precision specified in the relevant format description file or 5, whichever is smaller. The second line shows the name of the variable (latitude, longitude) and the number of values in the data file for the variable (20 for both latitude and longitude).

The minimum and maximum values for the variable are shown next (-83.223548 is the minimum and 77.883119 is the maximum value for latitude). The maximum and minimum values are given here with a precision of 6, which is the precision specified in the format description file. The locations of the maximum and minimum values in the input file are indicated. (-83.223548 is the fifth latitude value in `latlon.dat` and 77.883119 is the twelfth). Finally, the name of the histogram data (or variable summary) file generated for each variable is given (`latitude.lst` and `longitude.lst`).

10.3.2 Variable Summaries

The name of each variable summary file (list file) output by `checkvar` is of the form `variable.lst` for numeric variables and `variable.cst` for character variables. The data in `*.lst`, and `*.cst` files can be loaded into histogram plot programs for graphical representation. (You must be familiar enough with your program of choice to manipulate the data as necessary in order to achieve the desired result.) In Unix, there is no need to abbreviate the base file name.

> **NOTE:** If you use the -v option, the order of variables in var˙file has no effect on the numbering of base file names of the variable summary files.

The two example variable summary files, `latitude.lst` and `longitude.lst`, are shown next.

```
latitude.lst                  longitude.lst
-83.224  1                    -176.162  1
-69.274  1                    -155.234  1
-65.865  1                    -153.544  1
-63.212  1                    -136.941  1
-63.003  1                    -113.661  1
-47.304  1                     -79.178  1
-34.578  1                     -77.506  1
-28.287  1                     -55.442  1
-12.967  1                     -21.644  1
 -0.929  1                       0.777  1
 11.624  1                      30.172  1
 12.588  1                      35.591  1
 27.331  1                      36.356  1
 29.918  1                      38.875  1
 35.086  1                      55.319  1
 35.130  1                      62.152  1
 38.818  1                      91.411  1
 54.118  1                     134.124  1
 77.652  1                     144.804  1
 77.883  1                     149.408  1
```

The variable summary files consist of two columns. The first indicates boundary values for data bins and the second gives the number of data points in each bin. Because a precision of 3 was specified in the example, each boundary value has three decimal places. The boundary values are determined dynamically by `checkvar` and often do not correspond to data values in the input file, even if the `checkvar` and data file precisions are the same.

The first data bin in `latitude.lst` contains data points in the range -83.224 (inclusive) to -69.274 (exclusive); neither boundary number exists in `latlon.dat`. The first bin has one data point, -83.223548. The fourth data bin contains latitude values from -63.212 (inclusive) to -63.003 (exclusive), again with neither boundary value occurring in the data file. The data point in the fourth bin is -63.211962.

A
HDF Utilities

FreeForm ND includes three utilities for use with HDF (hierarchical data format) files: `makehdf`, `splitdat`, and `pntshow`. These programs were built using both the FreeForm library and the HDF library, which was developed at the National Center for Supercomputer Applications (NCSA).

The `makehdf` program converts binary and ASCII data files to HDF files and converts multiplexed band interleaved by pixel image files into a series of single parameter files. The `splitdat` program is used to separate and reformat data files containing headers and data into separate header and data files, or to translate them into HDF files. The `pntshow` program extracts point data from HDF files into binary or ASCII format.

It is assumed in this chapter that you have a working familiarity with HDF terminology and conventions. See the HDF user documentation for detailed information.

NOTE: Do not try the examples in this chapter. The example file set is incomplete.

A.1 `makehdf`

Using `makehdf` you can convert data files with formats described in a FreeForm format file into HDF files. You should follow FreeForm naming conventions for the data and format files. For details about FreeForm conventions, see Chapter 8.

> **NOTE:** A dBASE input file must be converted to ASCII or binary using `newform` before you can run `makehdf` on it.

The HDF file resulting from a conversion consists either of a group of scientific datasets (SDS's), one for each variable in the input data file, or of a *vgroup* containing all the variables as one *vdata.* If you are working with grid data, you will want SDS's (the default) in the output HDF file. A vdata (`-vd` option) is the appropriate choice for point data.

The `makehdf` command has the following form:

```
makehdf input_file [-r rows] [-c columns] [-v var_file] [-d HDF_description_file]
    [-xl x_label -yl y_label] [-xu x_units -yu y_units]
    [-xf x_format -yf y_format] [-id file_id] [-vd [vdata_file]] [-dmx [-sep]]
    [-df] [-md missing_data_file] [-dof HDF_file]
```

input_file Name of the input data file. Following FreeForm naming conventions, the standard extensions for data files are `.dat` for ASCII format and `.bin` for binary.

`-r` ***rows*** Option flag followed by the number of rows in each resulting scientific dataset. The number of rows must be specified through this option on the command line, or in an equivalence table, or in a header (`.hdr`) file defined according to FreeForm standards.

`-c` ***columns*** Option flag followed by the number of columns in each resulting scientific dataset. The number of columns must be specified through this option on the command line, or in an equivalence table, or in a header (`.hdr`) file defined according to FreeForm standards.

For information about equivalence tables, see the GeoVu Tools Reference Guide.

`-v` ***var_file*** Option flag followed by the name of the variable file. The file contains names of the variables in the input data file to be processed by `makehdf`. Variable names in *var_file* can be separated by one or more spaces or each name can be on a separate line.

`-d` ***HDF_description_file*** Option flag followed by the name of the file containing a description of the input file. The description will be stored as a file annotation in the resulting HDF file.

`-xl` ***x_label*** `-yl` ***y_label*** Option flags followed by strings (labels) describing the x and y axes; labels must be in quotes (`" "`) if more than one word.

`-xu` ***x_units*** `-yu` ***y_units*** Option flags followed by strings indicating the measurement units for the x and y axes; strings must be in quotes (`" "`) if more than one word.

`-xf` ***x_format*** `-yf` ***y_format*** Option flags followed by strings indicating the formats to be used in displaying scale for the x and y dimensions; strings must be in quotes (`" "`) if more than one word.

`-id` ***file_id*** Option flag followed by a string that will be stored as the ID of the resulting HDF file.

`-vd` **[*vdata_file*]** Option flag indicating that the output HDF file should contain a vdata. The optional file name specifies the name of the output HDF file; the default is `input_file.HDF`.

`-dmx [-sep]` The option flag `-dmx` indicates that input data should be demultiplexed from band interleaved by pixel to band sequential form in `input_file.dmx`. If `-dmx` is followed by `-sep`, the input data are demultiplexed into separate variable files called `data_file.1` ...`data_file.n`

`-df` To use this option, the input file (`data_file.ext`) must be a binary demultiplexed (band sequential) file. For each input variable in the applicable FreeForm format description file, there is a corresponding demultiplexed section in the output HDF file.

`-md` ***missing_data_file*** Option flag followed by the name of the file defining missing data (data you want to exclude). Use this option only along with the vdata (-vd) option. Each line in the missing data file has the form:

```
variable_name lower_limit upper_limit
```

The precision of the upper and lower limits matches the precision of the input data.

`-dof` ***HDF_file*** Option flag followed by the name of the output HDF file. If you do not use the `-dof` option, the default output file name is `input_file.HDF`.

A.1.1 Example

You will use `makehdf` to store `latlon.dat` as an HDF file. The HDF file will consist of two SDS's, one each for the two variables latitude and longitude. Each SDS will have four rows and five columns.

To convert `latlon.dat` to an HDF file, enter the following command:

```
makehdf latlon.dat -r 4 -c 5
```

As `makehdf` translates `latlon.dat` into HDF, processing information is displayed on the screen:

```
1   Caches (1150 bytes) Processed: 800 bytes written to latlon.dmx

Writing latlon.HDF and calculating maxima and minima ...

Variable latitude:
Minimum: -86.432712  Maximum 89.170904
Variable longitude:
Minimum: -176.161101  Maximum 165.066193
```

The output from `makehdf` is an HDF file named `latlon.HDF` (by default). It contains the minimum and maximum values for the two variables as well as the two SDS's.

A temporary file named `latlon.dmx` was also created. It contains the data from latlon.dat in demultiplexed form . The data was converted from its original multiplexed form to enable `makehdf` to write sections of data to SDS's.

If you start with a demultiplexed file such as `latlon.dmx`, the translation process is much quicker, particularly for large data files. As an illustration, try this. Rename `latlon.dmx` to `latlon.bin` (renaming is necessary for `makehdf` to find the format description file `latlon.fmt` by default). Enter the following command:

```
makehdf latlon.bin -df -r 4 -c 5
```

The output file again is `latlon.HDF`, but notice that no demultiplexing was done.

A.2 splitdat

The splitdat program translates files with headers and data into separate header and data files or into HDF files. If the translation is to separate header and data files, the header file can include indexing information.

The combination of header and data records in a file is often used for point data sets that include a number of observations made at one or more stations or locations in space. The header records contain information about the stations or locations of the measurements. The data records hold the observational data. A station record usually indicates how many data records follow it. The structure of such a file is similar to the following:

```
Header for Station 1
Observation 1 for Station 1
Observation 2 for Station 1
       .
       .
       .
Observation N for Station 1

Header for Station 2
Observation 1 for Station 2
Observation 2 for Station 2
       .
       .
       .
Observation N for Station 2

Header for Station 3
       .
       .
       .
```

Many applications have difficulty reading this sort of heterogeneous data file. One solution is to split the data into two homogeneous files, one containing the headers, the other containing the data. With splitdat, you can easily create the separate data and header files. To use splitdat for this purpose, the input and output formats for the record headers and the data must be described in a FreeForm format description file. To use splitdat for translating files to HDF, the input format must be described in a FreeForm format description file. You should follow FreeForm naming conventions for the data and format files. For details about FreeForm conventions, see Chapter 8.

The splitdat command has the following form:

splitdat *input˙file* [*output˙data˙file* ¿ *output˙header˙file*]

input_file Name of the file to be processed. Following FreeForm naming conventions, the standard extensions for data files are `.dat` for ASCII format and `.bin` for binary.

output_data_file Name of the output file into which data are transferred with the format specified in the applicable FreeForm format description file. The standard extensions are the same as for input files. If an output file name is not specified, the default is standard output.

output_header_file Name of the output file into which headers from the input file are transferred with the format specified in the applicable FreeForm format description file. If an output header file name is not specified, the default is standard output.

A.2.1 Index Creation

You can use the two variables begin and extent (described below) in the format description for the output record headers to indicate the location and size of the data block associated with each record header. If you then use `splitdat`, the header file that results can be used as an index to the data file.

`begin` Indicates the offset to the beginning of the data associated with a particular header. If the data is being translated to HDF, the units are records; if not, the units are bytes.

`extent` Indicates the number of records (HDF) or bytes (non-HDF) associated with each header record.

Example

You will use `splitdat` to extract the headers and data from a rawinsonde (a device for gathering meteorological data) ASCII data file named `hara.dat` (HARA = Historic Arctic Rawinsonde Archive) and create two output files-`23338.dat` containing the ASCII data and `23338hdr.dat` containing the ASCII headers. The format description file `hara.fmt` should contain the necessary format descriptions.

Here is `hara.fmt`:

```
ASCII_input_record_header "ASCII Location Record input format"
WMO_station_ID_number 1 5 char 0
latitude 6 10 long 2
longitude_east 11 15 long 2
year 17 18 uchar 0
month 19 20 uchar 0
day 21 22 uchar 0
hour 23 24 uchar 0
flag_processing_1 28 28 char 0
flag_processing_2 29 29 char 0
flag_processing_3 30 30 char 0
station_type 31 31 char 0
sea_level_elev 32 36 long 0
instrument_type 37 38 uchar 0
number_of_observations 40 42 ushort 0
identification_code 44 44 char 0

ASCII_input_data "Historical Arctic Rawinsonde Archive input format"
atmospheric_pressure 1 5 long 1
geopotential_height 7 11 long 0
temperature_deg 13 16 short 0
dewpoint_depression 18 20 short 0
wind_direction 22 24 short 0
wind_speed_m/s 26 28 short 0
flag_qg 30 30 char 0
flag_qg1 31 31 char 0
flag_qt 33 33 char 0
flag_qt1 34 34 char 0
flag_qd 36 36 char 0
flag_qd1 37 37 char 0
flag_qw 39 39 char 0
flag_qw1 40 40 char 0
flag_qp 42 42 char 0
flag_levck 43 43 char 0

ASCII_output_record_header "ASCII Location Record output format"
       .
       .
       .

ASCII_output_data "Historical Arctic Rawinsonde Archive output format"
       .
       .
       .
```

To “split” `hara.dat`, enter the following command:

```
splitdat hara.dat 23338.dat > 23338hdr.dat
```

The data values from `hara.dat` are stored in `23338.dat` and the headers in `23338hdr.dat`.

Because the variables begin and extent were used in the header output format in `hara.fmt` to indicate data offset and number of records, `23338hdr.dat` has two columns of data showing offset and extent. Thus, it can serve as an index into `23338.dat`.

A.2.2 HDF Translation

If output files are not specified on the `splitdat` command line, a file named `input_file.HDF` is created. It is hierarchically named and organized as follows:

```
              vgroup
         input_file_name
             /      \
            /        \
        vdata1       vdata2
   "PointIndex"      "input_file_name"
```

- `vdata1` contains the record headers
- `vdata2` contains the data
- If writing to a Vset (represented by a vgroup), both output formats are converted to binary, if not binary already.

Example

To create the file `hara.HDF` from `hara.dat`, enter the following abbreviated command:

```
splitdat hara.dat
```

The output formats in `hara.fmt` are automatically converted to binary, and subsequently the ASCII data in `hara.dat` are also converted to binary for HDF storage.

A.3 pntshow

The **pntshow** program is a versatile tool for extracting point data from HDF files containing scientific datasets and Vsets. The extraction can be done into any binary or ASCII format described in a FreeForm format description file. Before using **pntshow** on an HDF file, you should pack the file using the NCSA-developed HDF utility hdfpack.

You can use **pntshow** to extract headers and data from an HDF file into separate files or to extract just the data. It's a good idea to define GeoVu keywords in an equivalence table to facilitate access to HDF objects. For information about equivalence tables, see the GeoVu Tools Reference Guide. The input and output formats must be described in a FreeForm format description file. You should follow FreeForm naming conventions for the data and format files. For details about FreeForm conventions, see Chapter 8.

If a format description file is not specified on the command line, the output format is taken by default from the FreeForm output format annotation stored in the HDF file. If there is no annotation, a default ASCII output format is used.

> **NOTE:** An equivalence table takes precedence over everything. (vdata=1963, SDS=702)

If you have not specified an HDF object in an equivalence table, **pntshow** uses the following sequence to determine the appropriate source for output:

❶ Output the first vdata with class name Data.

❷ Output the largest vdata.

❸ Output the first SDS.

If no vdatas exist in the file, but an SDS is found, it is extracted and a default ASCII output format is used.

A.3.1 Extracting Headers and Data

The **pntshow** command takes the following form when you want to extract headers and data from HDF files into separate files.

```
pntshow input_HDF_file [-h [output_header_file]] [-hof output_header_format_file]
    [-hof output_header_format_file] [-d [output_data_file]]
    [-dof output_data_format_file]
```

input_HDF_file Name of the input HDF file, which has been packed using `hdfpack`.

`-h` **[*output_header_file*]** Option flag followed optionally by the name of the file designated to contain the record headers currently stored in a vdata with a class name of Index. If an output header file name is not specified, the default is standard output.

`-hof` ***output_header_format_file*** Option flag followed by the name of the FreeForm format file that describes the format for the headers extracted to standard output or output_header_file.

`-d` **[*output_data_file*]** Option flag followed optionally by the name of the file designated to contain the data currently stored in a vdata with a class name of Data. If an output file name is not specified, the default is standard output.

`-dof` ***output_data_format_file*** Option flag followed by the name of the FreeForm format file that describes the format for data extracted to standard output or *output_data_file*.

Example

You will extract data and headers from `hara.HDF` (created by `splitdat` in a previous example). This file contains two vdatas: one has the class name Data and the other has the class name Index. Because this file is extremely small, no appending links were created in the file, so there is no need to pack the file before using `pntshow`, though you can if you wish.

To extract data and headers from `hara.HDF`, enter the following command:

```
pntshow hara.HDF -d haradata.dat -h harahdrs.dat
```

The data from the vdata designated as Data in `hara.HDF` are now stored in `haradata.dat`. The data are in their original format because the original output format was stored by `splitdat` in the HDF file. The header data from the vdata designated as Index in `hara.HDF` are now stored in `harahdrs.dat`. In addition to the original header data, the variables begin and extent have also been extracted to `harahdrs.dat`.

A.3.2 Extracting Data Only

The `pntshow` command takes the following form when you want to extract just the data from an HDF file:

`pntshow` *input_HDF_file* [`-of` *default_output_format_file*] [¿ *output_file*]

input_HDF_file Name of the input HDF file, which has been packed using `hdfpack`.

`-of` ***default_output_format_file*** Option flag followed by the name of the FreeForm format file that describes the format for data extracted to standard output or *output_file*.

output_file Name of the output file into which data is transferred. If an output file name is not specified, the default is standard output.

Examples

You can use `pntshow` to extract designated variables from an HDF file. In this example, you will extract temperature and pressure values from `hara.HDF` to an ASCII format. First, the following format description file must exist.

Here is `haradata.fmt`:

```
ASCII\_output_data "ASCII format for pressure, temp"
atmospheric_pressure 1 10 long 1
temperature_deg 15 25 float 1
```

To create a file named `temppres.dat` containing only the temperature and pressure variables, enter either of the following commands:

```
pntshow hara.HDF -of haradata.fmt > temppres.dat
```

or

```
pntshow hara.HDF -d temppres.dat -dof haradata.fmt
```

If you use the first command, `pntshow` searches `hara.HDF` for a vdata named Data. Since `hara.HDF` contains only one vdata named `Data`, this vdata is extracted by default with the format specified in `haradata.fmt`.

The results are the same if you use the second command. Now, try running `pntshow` on the previously created file `latlon.HDF`, which contains two SDS's. Use the following command:

```
pntshow latlon.HDF > latlon.SDS
```

The `latlon.SDS` file now contains the latitude and longitude values extracted from `latlon.HDF`. They have the default ASCII output format. You could have used the -of option to specify an output format included in a FreeForm format description file.

B
Error Handling

The FreeForm ND error handling system captures errors, such as improper usage, code problems, and system errors, and places them in an error queue. For each error captured, error type and a short message are placed in the message queue. If a fatal error occurs, the program stops executing and displays all error messages in the queue.

B.1 Error Messages

The following is a list of some possible error messages with suggestions for corrections.

Problem opening, reading, or writing to file Check that all file names and paths are correct.

Problem making format Make sure there is a format file describing the data file formats.

Check that input and output format descriptions in the format file accurately describe the data.

Problem making header format If a header exists in the data file, it must be described in a format file.

Check that the header description accurately describes the header in your data file.

Problem getting value

Problem processing variable list The data formats may not be described correctly or there may be some inconsistencies in the data.

Check also for unprintable characters at the end of the data file.

File length / Record length mismatch

Record Length or CR Problem This usually happens because the input format description is not correct.

Make sure the format description's last position is the last character before the end-of-line character. If you have a header, make sure it is described correctly.

The header's length must include all characters up until the last end-of-line-character before the data begins.

Binary Overflow Try using a larger output variable type such as a long instead of a short.

Be sure you have given enough space for the values to be written.

See Chapter **??** for more information.

Variable not found The variable names in your output format must match the variable names in the input format unless you are using conversion variables.

Data Overflow

****** Data overflow does not usually cause a fatal error and FreeForm ND functions try to anticipate them. If overflow occurs for a particular value, `***`'s are written to that value's location.

If you find these in your output, check your variable positions and precision. Increase field width or use a "larger" data type.

Be sure the output format specifies space for the output variable. For instance, FreeForm ND adds a leading zero in front of decimal points. If the original data did not have a leading zero, the output will have one more digit than the input.

Insufficient memory allocation The application has run out of memory. Try using the `-b` (local buffer size) option, or modify `autoexec.bat` and `config.sys` and comment out devices, TSR's, etc.

Index

H

I

L

M

N

O

P

Q

R

S

T

V

X

Y