Chapter 5 Programs That Run on Remote Hosts

Chapter Overview

This chapter describes DataScope's networking capabilities and its relationship with Telnet 2.3MacTCP.

Networking Introduction

Installing Your Network

NCSA DataScope version 2.0 can be used with or without a TCP/IP network connection. If you do not install the network functions, all local functions of DataScope continue to work as before. However, if you do install it, you must set up MacTCP correctly on your Macintosh.

NOTE: You should install NCSA Telnet 2.3 -MacTCP- on your Macintosh before attempting to use any of the DataScope network capabilities. The configuration file is used by DataScope to resolve the addresses of remote machines.

After installing NCSA Telnet 2.3 -MacTCP-, make sure you can connect to the appropriate host machines with NCSA Telnet, and that you can log in via Telnet. If you cannot, consult the NCSA Telnet documentation, contact your local networking support organization, or contact NCSA to get NCSA Telnet working first. (See the Bugs and ReadMe pages of this document.)

NOTE: NCSA DataScope does not support name server functions. If the name of the machine is not in the config.tel file explicitly, you will have to use the machine's Internet address.

The only change you may want to make to the config.tel file is to include all names and IP addresses of machines you wish to connect to from DataScope.

NOTE: You should place your config.tel file from NCSA Telnet 2.3 -MacTCP- in the system folder. If you don't, an error message stating that there is a problem with the config.tel file will appear on the screen.

DataScope 2.0 is no longer limited to the processing power of the Macintosh alone. Notebook functions, which you may not have used before because they were too slow or you didn't have access to a local compiler, can now automatically be executed on a remote compute server. Although you do have to wait for the data to be sent there and back, this inconvenience is negligible when compared to the increased computational speed of the supercomputer.

If you have an existing FORTRAN or C application running on a supercomputer or other UNIX computer with the TCP/IP network protocols, DataScope can be used as a flexible, interactive output device for two dimensional arrays. When the program or a specific time step finishes, a simple subroutine call transfers the array to your Macintosh.

Figures 5.1 and 5.2 show the new network functions in DataScope 2.0. There are two completely separate functions which may be used. You can:

- send datasets from your mainframe to DataScope on demand from a mainframe application that you've written.
- use a remote compute server to access notebook functions that are too time-consuming to calculate or that you cannot compile on your Macintosh.

Figure 5.1 One-way Passage of Data from a Remote Host to DataScope on the Mac

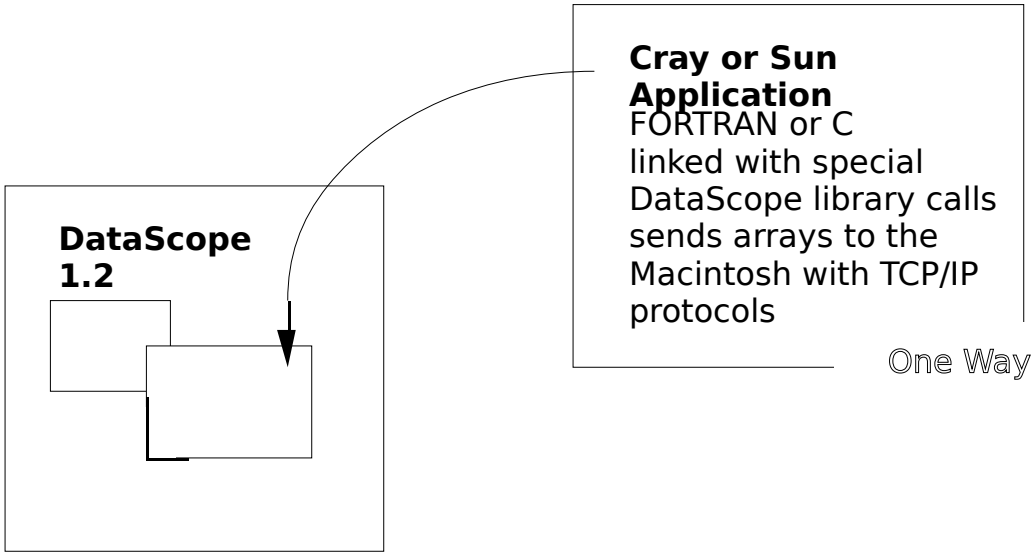
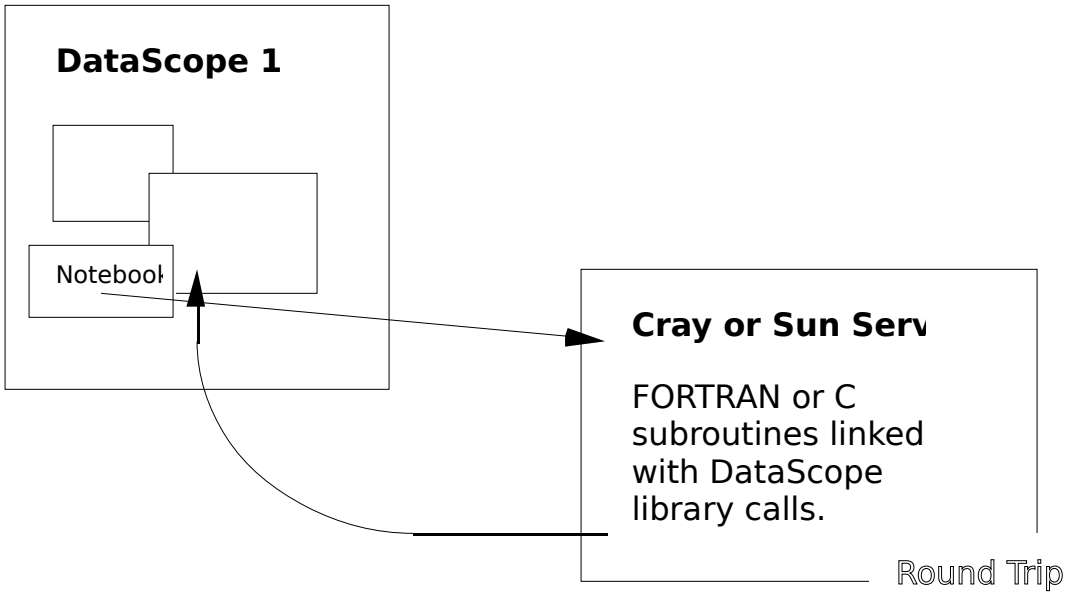


Figure 5.2 Two-way Passage of Data from a Remote Host to DataScope on the Mac



One-Way Process

An example of a one-way process in DataScope (Figure 5.1) would be a FORTRAN application used to simulate a growing, moving thunderstorm. After each simulation time step, your application might send out one or more arrays which "pop up" in the background of your Macintosh screen. This is a one way transfer; the data originates in your remote simulation and ends up in DataScope on your Macintosh.

Two-Way Process

The need for a two-way process of data in the program (Figure 5.2) might come later when you are looking at your array in DataScope and need to perform a Fourier Transform on one of the arrays to analyze it in frequency space. Once you've prepared a FFT routine on a supercomputer, all you have to do is select the Calculate From Notes command from the Numbers menu on the following notebook formula. (See Chapter 4, "Notebook Calculations.")

```
transform = fft(mydata)
```

This kind of compute serving requires a round trip for the data. DataScope delivers the array to the remote computer, waits for the response, and then receives the answer back.

Checking Your Networking Setup

NCSA DataScope does not contain diagnostics or error messages concerning the operation of the network. Therefore, your best bet to debug any network problem is to make sure NCSA Telnet works first.

NOTE: In the worst case, you may have to enter the IP address of the machine to connect to. If NCSA Telnet can connect to a machine by name, you can always connect to the machine from NCSA DataScope by using its IP number.

DataScope provides two kinds of diagnostics to let you know whether the network has been initialized successfully or not.

- In the About DataScope dialog under the Apple menu, the information includes an indicator of "Network Enabled" or "Network Disabled".
- If the network is disabled and you attempt to execute a notebook function which is not local, the "Use Network Server" button will be dimmed and cannot be selected.

Using DataScope for Remote Computing: Delivering Data to DataScope with TCP/IP

Most supercomputer simulations write out results into ASCII or binary data files, which you then can use to print out or create graphs or images from the numbers. With DataScope 2.0, you can bypass viewing the intermediate data file since the program displays the data on your screen immediately, as values are generated. The following section describes this process and give examples.

In order to display an array, DataScope needs the data information listed in Table 5.1.

Table 5.1 Required Information to Display an Array

• name	the name of the variable to display
• flags	the array's purpose (Flags specifies what is to be done with the data when returned to DataScope.)
• nrows	the number of rows in the array
• ncols	the number of columns in the array
• rows	the scale values (independent variable), one for each row
• cols	the scale values (independent variable), one for each column
• max	the largest value of interest in the array
• min	the smallest value of interest in the array
• vals	the array of floating-point data values

With the appropriate linkable library (source provided on DataScope 2.0 distribution disk), you can write a FORTRAN program which combines all of this information into one call statement as shown in Figure 5.3.

Figure 5.3 FORTRAN Call Statement

```
call
ds_send(mac,name,flags,max,min,nrows,ncols,rows,cols,vals)
```

The additional "mac" parameter is present so that the remote computer can indicate which Macintosh is ready to receive the array. This call may be used anywhere in a FORTRAN or C program written for a UNIX host computer where the special DataScope linkable library will operate.

The `ds_send` call connects the remote computer to your Macintosh and sends all of the relevant information over the TCP/IP connection to DataScope. When DataScope finishes receiving all of the data, the array window appears on the screen, just as if you had used the "Open" command on a data file. Of course this means that DataScope 2.0 must be ready and waiting, running under MultiFinder before the transfer attempt starts.

Generating an Image Automatically

DataScope 2.0 can generate an image each time it receives a data array from another computer. The flags field from the `ds_send` command is used to indicate that an image should be generated as soon as the whole data array arrives. For example, you can use the '`G`' flag to generate an image by entering the boxed code segment in Figure 5.4.

Figure 5.4 Call Statement using 'G' Flag

```
flags = 'G'
call
ds_send(mac,name,flags,max,min,nrows,ncols,rows,cols,vals)
```

As soon as this array arrives in DataScope, the "Generate Image" command is invoked just as if you had selected it from the Image menu. The new image window appears on top of the text window.

Making Movies with a Sequence of Arrays

The '`R`' flag performs a special "replace" operation in DataScope. When using the '`R`' flag, all of the members of its associated sequence must be the same size (cols,rows), and they must have the same or similar names. In your FORTRAN program, use the same `ds_send` call and add the '`R`' flag to the flags parameter (Figure 5.5). This flag causes any previously sent dataset to be replaced by subsequently sent datasets.

Figure 5.5 ds_send Call with 'R' Flag

```
flags = 'RG'
call
ds_send(mac,name,flags,max,min,nrows,ncols,rows,cols,vals)
```

DO Loop

Using a DO loop around this same `ds_send` call with different arrays of the same size cycles each array through the same DataScope window(s). You can easily create a movie effect with a top speed of approximately one frame every two seconds by using small (50 x 50) arrays and high-speed Ethernet links. (See the this chapter's section, "Making Faster Movies" for more information.)

Variable Names

A special component of the movie sequencing is the use of the variable name.

A sequence of variable names can be created from a base name and an extension. The base name must be identical in each variable name entry for the replace flag to work, but the extension may be arbitrary (e.g., _001, _002, _003, etc.). For example, with a base name of "`pressure`" and numbers used as extensions, the sequence of names in Figure 5.6 can be generated and used to create a movie sequence in DataScope .

Figure 5.6 Example Code to Create a Movie Sequence

```
pressure_001
pressure_002
pressure_003
...
```

The separator between the base name and the extension can be any non-alphanumeric character. A period, underscore, or even a space may be used. An underscore is recommended since DataScope translates all non-alphanumerics to underscores before using the specified name.

To sum up:

- If you use the *identical* variable name for `ds_send`, then the replace flag works as expected with each new array appearing in the same window.
- If you use a *brand new* name, a new window appears whether the '`R`' flag is used or not.
- If you use the replace flag with *more than one* variable name alternately, you allow two movies to proceed in two different windows at the same time. DataScope matches the windows by variable name.

Making Faster Movies

A primary factor in the delay between frames is the time it takes to make a network connection between the remote computer and the Macintosh. With a slight variation on the `ds_send` call, you can remove the need to open and close the connection between each frame of a movie sequence. This new form involves the `ds_send1` call. The "`mac`" parameter has been moved to a new call, `ds_open`. Your code would resemble that in Figure 5.7.

Figure 5.7 Using ds_send1 and ds_open for Faster Movies

```
mac = '192.17.20.10'
call ds_open(mac)
flags = 'RGS'
do 100 i=1,10
c     calculate new vals array
c
100       call ds_send1(name,flags,max,min,nrows,ncols,rows,cols,vals)

call ds_close
```

Saving to Disk

To save an array, use the flag '`S`' for Save. Each time an array arrives with the '`S`' flag set, the array and/or image(s) will be displayed on the screen as usual, and then saved to disk in an HDF file. This save operation is identical to selecting the Save command from the File menu.

NOTE: If you repeat the use of a variable name, the new file will be saved on top of the old file, thus replacing the former file. Use a

base name plus an extension as discussed in this chapter's section "Making Movies with a Sequence of Arrays" to avoid overwriting files while saving a movie.

With the 'G' flag to generate an image, the 'R' flag to cause a replacement each time, and the 'S' flag to save the files to disk, the remote FORTRAN or C program can create an entire animated movie sequence on disk. This sequence may be animated from disk with NCSA Image, NCSA ImageIP, or other programs in the NCSA suite of Macintosh software, which can redisplay the animation while adjusting speed and direction of the animation.

To make the use of NCSA Image easier, run the animation once without the 'S' flag on. Then save the last frame, or a representative frame, into an empty folder. This sets up the default folder for the movie sequence. Run the animation again with the 'S' flag and DataScope fills that folder with a complete set of animation files.

Command Summary

Listed below is a summary of the routines used in DataScope.

DS_SEND

The `ds_send` call is shown in Figure 5.8, as defined in FORTRAN.

Figure 5.8 ds_send in FORTRAN

```
SUBROUTINE DS_SEND(MAC,NAME,FLAGS,MAX,MIN,NROWS,NCOLS,ROWS,COLS,VALS)

CHARACTER *80 MAC,NAME,FLAGS
INTEGER NROWS,NCOLS
REAL MAX,MIN,ROWS(NROWS),COLS(NCOLS),VALS(NCOLS,NROWS)
```

where:

- mac — the name or IP address of the Macintosh to connect with
- name — the name of the variable to display
- flags — the function of the array (Flags specifies what is to be done with the data when its returned to DataScope.)
- max — the largest value of interest in the array
- min — the smallest value of interest in the array
- nrows — the number of rows in the array
- ncols — the number of columns in the array

- rows the scale values (independent variable), one for each row
- cols the scale values (independent variable), one for each column
- vals the array of floating-point data values

DS_SEND1

The `DS_SEND1` version of the `ds_send` call is the same, except for the `MAC` parameter. The MAC parameter is used in the separate `DS_OPEN` call (Figure 5.9).

Figure 5.9 ds_send1 in FORTRAN

```
SUBROUTINE DS_SEND1(NAME,FLAGS,MAX,MIN,NROWS,NCOLS,ROWS,COLS,VALS)
SUBROUTINE DS_OPEN(MAC)
SUBROUTINE DS_CLOSE
```

Flag Summary

The *flags* field gives the remote computer application several options about what DataScope should do after it has received an array. The fields are all set by using upper-case letters corresponding to the Command-keys in the DataScope menus. Except for the addition of the `'R'` replace flag, the capital letters have the same functions as the menu commands they represent.

Table 5.2 Flags and Their Functions

Flags	Function
G	Generate Image
I	Interpolated Image
P	Polar Image
S	Save
R	Replace

Flags may be used in any combination and in any order. The entire flags string is scanned for the occurrence of these letters. To disable a function, omit it from the flags string. The default performs none of these functions.

Using C Calls

The C form of the `ds_send` call (Figure 5.10) has the same parameters as the FORTRAN call. Below is the C syntax. Strings are zero-terminated C strings of any (appropriate) length.

Figure 5.10 ds_send in C

```
ds_send(mac,name,flags,max,min,nrows,ncols,rows,cols,vals)
  char *mac,*name,*flags;
  int nrows,ncols;
  float max,min,*rows,*cols,*vals;
```

ds_open, ds_send1, and ds_close are similar to their FORTRAN counterparts as shown in Figure 5.11.

Figure 5.11 ds_open, ds_send, ds_close in C

```
ds_open(mac)
  char *mac;
ds_close()

ds_send1(name,flags,max,min,nrows,ncols,rows,cols,vals)
  char *name,*flags;
  int nrows,ncols;
  float max,min,*rows,*cols,*vals;
```

NOTE: You may only have one connection open at a time. Entering COMMAND-period (⌘-.) at any time will interrupt the network activity and reset the network.

Linking in dscall.o

The ds_send calls for both FORTRAN and C are contained in the same object file, dscall.o compiled from the C source in dscall.c. You should include this file on the cc or f77 compile lines directly. On UNICOS, dscall.o references routines in libnet.a, so it requires that you include that library on the link line. To create dscall.o on any UNIX system, just compile it with the -c flag.

To compile dscall.o on Sun and UNICOS systems, enter the commands in Figure 5.12.

Figure 5.12 Code to Compile dscall.o

```
cc -DSUN -c dscall.c
cc -DUNICOS -c dscall.c
```

UNICOS compile examples (Figure 5.13):

Figure 5.13 UNICOS Compile Examples

```
cc myprog.c dscall.o -lnet -o myprog
cf77 myprog.f dscall.o -lnet -o myprog
```

Sun UNIX compile examples, direct version (Figure 5.14):

Figure 5.14 UNIX Compile Examples

```
cc myprog.c dscall.o -o myprog
f77 myprog.c dscall.o -o myprog
```

Array Order

The declaration order of the arrays must be arranged according to the storage order of the language you are working with. The network connection to DataScope transfers the array in storage order, across from left to right as you go down the page (the same as reading English text). The matching declarations for the vals array in FORTRAN and C are presented in Figures 5.15 and 5.16, respectively.

Figure 5.15 **Vals Array in FORTRAN**

```
REAL VALS(NCOLS,NROWS)
```

Figure 5.16 **Vals Array in C**

```
float vals[nrows][ncols];
```

Using a Remote Network Server for Notebook Functions

Basics

Thus far we've discussed DataScope's capacity for remote computing. The second network function available in NCSA DataScope 2.0 is the ability to off-load Notebook computations to another computer. Previous versions of NCSA DataScope had several built-in functions which could manipulate arrays in local memory. They could combine the arrays to make more powerful formulas, but there was a limit to the number of functions provided and a limit to the processing power of the Macintosh. Now, as in previous versions, you may write external functions for DataScope which perform calculations that you write and compile for yourself. In addition, DataScope 2.0 allows access to functions which are compiled and run on other computers.

DataScope's server routines run on Cray UNICOS and Sun UNIX. They are written in portable C and use the `rexecd` mechanism provided by many UNIX systems, so they should be portable to other types of computers. As a user, you do not have to work with the communications routines directly. You write the function subroutines in FORTRAN or C as described below and then DataScope takes care of transferring the arrays back and forth.

NOTE: SunOS version 4.0 and later will not allow communication to be established with the `rexecd` daemon UNLESS the machine attempting to connect is in the remote machine's hosts file. Talk to your Sun administrator to get an entry for your Macintosh made in the hosts file.

DS_serve, the DataScope Server

`DS_serve` is the name of the program which DataScope accesses on your remote computer. For UNIX, the Makefile and example subroutines are provided on the DataScope distribution disk. When DataScope invokes the remote server, it looks for a program called `DS_serve` in your home directory and runs it. This program must be set up to receive DataScope function calls with the special `DS_serve` library provided in DataScope.

On the NCSA systems, home directories are limited in their space allocation. You can easily create a tiny shell script called `DS_serve` in your home directory which selects the proper `DS_serve` program in another directory. This arrangement only requires one line in the shell script; i.e., the full path name of the actual program to run (Figure 5.17).

Figure 5.17 Making a Shell Script with DS_serve

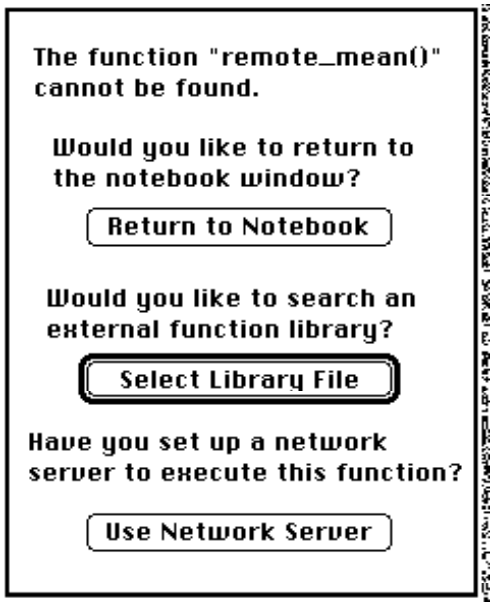
```
/scr4/u14013/DS_serve
```

Using Remote Functions in DataScope

Remote functions in DataScope are initiated the same way that local functions are initiated. To use remote functions, write a formula in the Notebook window and select it. Then choose the "Calculate From Notes" command in the Numbers menu. Any function name encountered in the formula which is on the list of local functions is performed locally while any other name is a candidate for an external or network function.

When a function name is not on the local list, DataScope leaves it up to the user to find the function. For example, if you were not satisfied with the speed at which DataScope calculates the mean of an array locally, you might define a remote function called `remote_mean()` to do the same task as the local mean() function. When DataScope looks for `remote_mean()`, it presents the choices in the dialog shown in Figure 5.18.

Figure 5.18 External Function Library Dialog Box

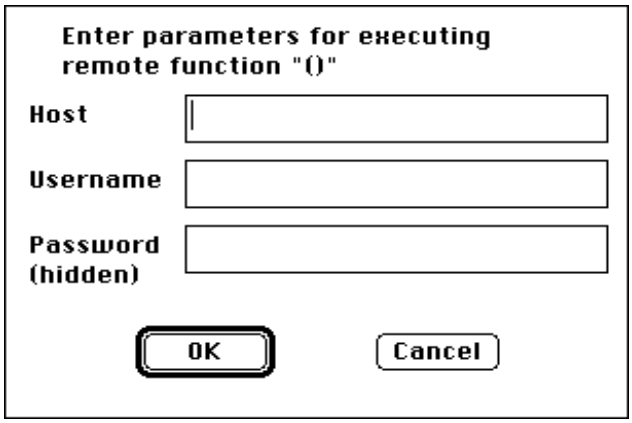


Select the top button if you have accidentally misspelled a function name or want to exit the dialog box. Press the Select Library File button when you have programmed an external function on your Macintosh. The new option, the Use Network Server button, invokes `DS_serve` over the network. Each of these options proceeds with the appropriate dialog boxes or returns you back to the notebook.

Figure 5.19 shows the Network Server dialog box. The information you enter in it is the same information you've entered when you logged into the remote machine using NCSA Telnet. When you

enter your password, it will echo with dummy characters. Backspace deletes the entire password, not just one character. Press OK or hit return to initiate the remote server function.

Figure 5.19 Network Server Dialog Box



NOTE: If the host you intend to connect to is not in your config.tel file, you will have to enter the IP address of the host rather than its name.

When you start the remote function, the appearance of the watch cursor indicates that the network transfers are taking place. The cursor continues to tick while the remote computer works on the problem and returns the answer. Then the new array pops up on the screen, just as it does for local computations.

NOTE: If at any time you want to cancel the operation, press COMMAND-period (⌘-.), the universal Macintosh cancel key.

Problems with rexecd

The `rexecd` mechanism can fail for many reasons. NCSA DataScope attempts to capture the error message and display it for you. Some of the more common errors are listed below.

- Cannot connect
  The remote function server cannot be reached over the network. Check that you have entered the name correctly and that NCSA Telnet can reach this host.

- Login incorrect
  The user name provided is not valid for the remote computer.

- Password incorrect
  The password given is not valid for the user to log in.

- No remote directory
  The home directory of the user could not be located.

- Command failed
  `DS_serve` cannot be found. It must be placed in the user's home directory.

- Function not found
  `DS_serve` does not have a function of the name you requested. Check your spelling and make sure the correct version of `DS_serve` is available in your home directory. Also check the declarations of your functions as described below.

NOTE: If at any time you want to cancel the operation, press COMMAND-period (⌘-.), the universal Macintosh cancel key.

Writing your own functions

Writing C functions for remote UNIX systems is exactly the same as writing DataScope external functions for the Macintosh. See Chapter 4 "Notebook Calculations" for an example. The structures passed to and from the user-written subroutine are defined in the file `DScope.h`. In fact, most C functions written for DataScope compile and run the same unmodified source code on Apple Macintosh, Sun UNIX, and Cray UNICOS systems.

The only extra step needed is to declare your function to the `DS_serve` program and compile it in with the provided Makefile. The declaration file is `dsfn.h`, an example is shown in Figure 5.20.

Figure 5.20 **dsfn.h File**

```
/*
*  Declarations for externally callable routines for the DataScope server
*  on UNIX machines.
*
*  Add your external function to each of the lists.
*  The internal routine, when compiled, will automatically register the
*  routine name and which function to call.
*
*  The routine name does not need to match the function name string.
*/

int
       NORM(),                                  /* FORTRAN declaration */
       exmean(),                                /* C declaration */
       puts();                                  /* dummy entry, anchors list */


struct flist {
       char *namestring;
       int (*fncall)();

};


struct flist dsc[] = {                    /* C calls list */
       "remote_mean",exmean,
       "",puts
```

Figure 5.20 **dsfn.h File (Continued)**

```
        };



        "norm",NORM,
        "",puts

        };
```

Each of your functions must appear in the C calls list in order to have it be available as a remote call in DataScope. The function name does not have to be the same as the compiled function name as shown in the example. `exmean()` is the name of the function, but it is referenced by the string "`remote_mean`".

Compiling and linking

The Makefile provided for Cray UNICOS is shown in Figure 5.21.

Figure 5.21 **Makefile for UNICOS**

```
DS_serve:  dsfn.h DScope.h dsfns.c DS_serve.c dsfnF.o
          cc -DUNICOS DS_serve.c dsfns.c dsfnF.o -lf -lu -lm
          -lnet -o DS_serve

dsfnF.o:  dsfnF.f
         cf77 -c dsfnF.f
```

The Makefile provided for Sun UNIX is shown in Figure 5.22.

Figure 5.22 **Makefile for UNIX**

```
DS_serve:  DS_serve.c dsfns.c dsfn.h DScope.h dsfnF.o
          cc -D SUN DS_serve.c dsfns.c dsfnF.o -lF77 -lI77 -lm     -f68881 -o
DS_serve

dsfnF.o: dsfnF.f
          f77 dsfnF.f -c -o dsfnF.o
```

This compile and link process references three source files.

DS_serve.c

`DS_serve.c` is the only file used without modification. It contains the number translation and array transfer code which communicates with DataScope on the Macintosh. The `-DUNICOS` or `-DSUN` flags in the Makefile are used by `DS_serve.c` to determine whether to use code specific to either UNICOS or Sun UNIX systems. `DS_serve.c` contains the `main()` routine, so you should not include any other `main()` in your other C files.

dsfns.c

`dsfns.c` is the example C source file that contains example functions. Add your own functions to this file, or create similar files and link them in. Make sure you declare these functions in `dsfn.h` first.

dsfnF.f

`dsfnF.f` is the example FORTRAN source file that contains an example function. Add your own functions to this file, or create similar files and link them in. Make sure you declare these functions in `dsfn.h` first.

Once these three files are compiled and linked together as the executable program `DS_serve`, install `DS_serve` in your home directory on the remote computer. You can use the UNIX ln, mv or cp commands to put `DS_serve` in place. It is now ready to be accessed as described in the section above.

Writing FORTRAN Functions

FORTRAN functions are declared in the same file as the C functions: `dsfn.h`. The organization of the data for the subroutine call is quite different, because FORTRAN cannot use structures. You also cannot use COMMON since the arrays are of variable size. The call is complex, but well-organized, so you should start by taking the example FORTRAN source and copying it for your own functions.

An example FORTRAN function is shown in Figure 5.23. It simply divides every element of an array by 254 and returns the result. It contains the full declaration of the arrays which can be used to perform a variety of one-parameter and two-parameter calculations.

Figure 5.23 Example FORTRAN Function

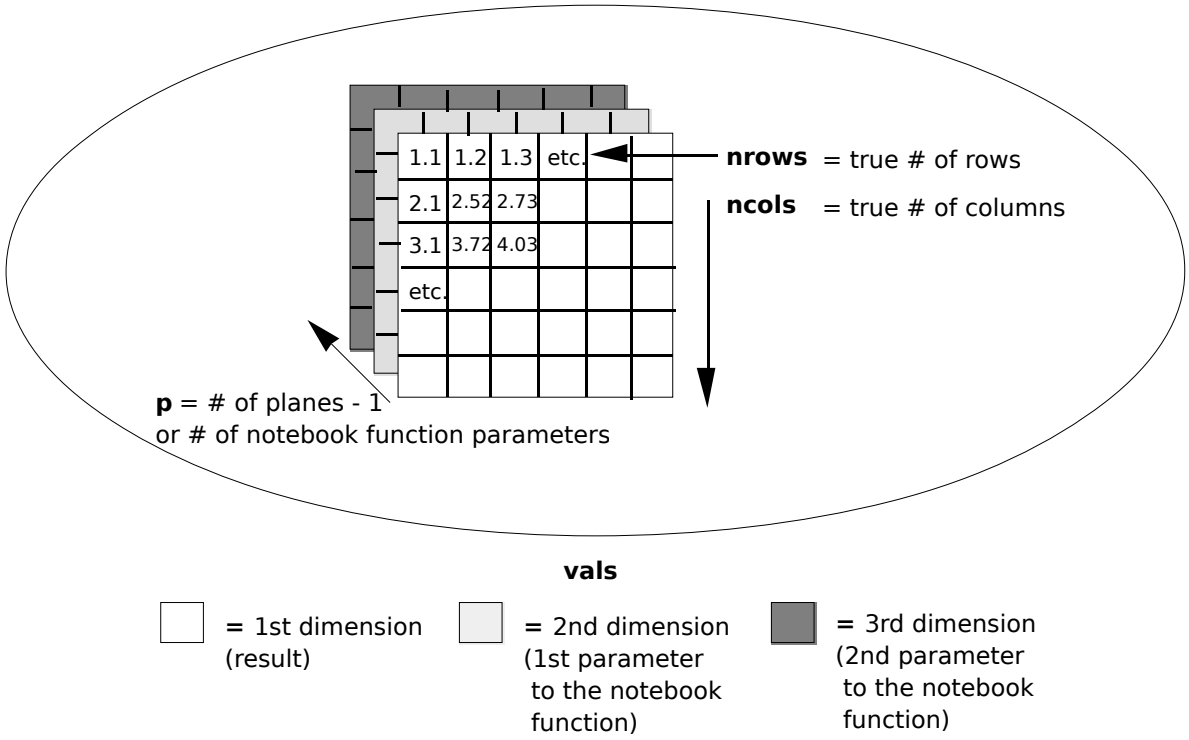
```
subroutine norm(vals,rows,cols,nrows,ncols,maxr,maxc,p)
integer maxr,maxc,p
integer nrows(0:p),ncols(0:p)
real vals(maxc,maxr,0:p)
real rows(maxr,0:p), cols(maxc,0:p)

do 100 j=1,nrows(0)
  do 100 i=1,ncols(0)
     vals(i,j,0) = vals(i,j,1)/255.0
100    continue
return
end
```

The arrays are laid out to allow for any size of two dimensional array(s) to be used. The arguments of the call all depend on `p`, the number of parameters to the notebook function. Figure 5.24 shows a pictorial representation of some of the arrays in a FORTRAN function.

- vals
  `Vals` represents all data. This matrix is three-dimensional, even though DataScope is dealing only with two-dimensional data sets. Like the other dimensions, the third dimension is used to store values that you provided as parameters (datasets) to the original notebook function call. In addition, it stores the result values. The first plane of the third dimension is used to store the result of the computation.

Figure 5.24 **Arrays in a FORTRAN Function**



The following arrays individually determine the mesh spacing characteristics for each of the planes of the `vals` matrix.

- rows
  This array has an one-dimensional array for each of the parameters to the DataScope function. Each one-dimensional array contains the offset values for each row of each plane in the `vals` matrix.

- cols
  This array also has an array for each of the planes in the `vals` matrix. Each array contains the offset values for each column of each plane in the vals matrix.

The next two arguments determine the *true size* of each plane in the `vals` matrix.

- nrows
  This array contains the number of rows in each plane of the `vals` matrix.

- ncols
  This array contains the number of columns in each plane of the `vals` matrix.

NOTE: Often, `nrows(1)` and `ncols(1)` will be equal to one. This indicates that the value in the plane is a constant. If there is only one row and one column, then there can only be one value. If your function returns a constant value, set `nrows(0)` and `ncols(0)` equal to one. The flexibility in the size of the planes in the vals matrix make matrix operations on matrix of different sizes possible.

- maxr,maxc
  These arrays represent the maximum number of rows and columns in any plane of the `vals` matrix. Use the `nrows`() and `ncols`() arrays for loop indices, but don't be surprised if `nrows`(0) is equal to `maxr` and `ncols`(0) is equal to `maxc`. However, make sure your code handles the cases where they are not equal.

- p
  This value is the number of notebook function parameters and is equal to the number of planes in vals minus 1 (the result plane will always be there). DataScope supports 0, 1, or 2 parameter notebook function calls, so `p` is used to dimension the `vals` array and the associated scales.

NOTE: The maximum value for `p` is 2 in NCSA DataScope 1.2. DataScope currently can only support two parameters for each function.

Using a Sample Program

This section outlines a sample program (Figure 5.25) which you can run on a remote host and from which you can route the resulting output back to DataScope on your Macintosh. Examples provided with DataScope include `dsfnF.f`, a sample FORTRAN simulation of a propagating wave. The output from `dsfnF.f` creates a movie sequence in DataScope. Instructions to compile and link `dsfnF.f` are included in the source file.

Also included in the sample program is `hdf2ds.c`, the source code for an HDF utility which demonstrates calling `ds_send` from C. This code requires NCSA's HDF library to be linked in and works only on HDF files.

Figure 5.25 Source Code for dsfnF.f

```
      subroutine norm(vals,rows,cols,nrows,ncols,maxr,maxc,p)
            integer maxr,maxc,p
            integer nrows(0:p),ncols(0:p)
            real vals(maxc,maxr,0:p)
            real rows(maxr,0:p), cols(maxc,0:p)

c
c  this example takes the first parameter array #1 and computes
c  the answer and places that in answer #0
c

      do 100 j=1,nrows(0)
               do 100 i=1,ncols(0)
                   vals(i,j,0) = vals(i,j,1)/255.0
      100    continue
      return
      end
```

DataScope's Interactions with a Remote Host

Ultimately, it is your responsibility to be familiar with the remote host. Each different host and each different operating system may have characteristics that will affect the way that DataScope works. For example, when using FORTRAN subroutines called from C on Sun systems, references to the FORTRAN subroutines within the C code must have an underscore appended to the name (thus, a subroutine named "norm" would be referred to as "norm_" in C code). Also in reference to the Sun OS 4.0, data is passed back and forth between DataScope on the Macintosh and the Sun via the `rexecd` system call; this call REQUIRES that the Macintosh's IP number be hardcoded in the system's host table.