PFF — A Portable File Format

Fortran Programmer's Reference Guide

David B. Seidel

DRAFT

This manual is not yet complete, but is provided as-is in its present form.

Last updated: August 14, 2014

Table of Contents

1.0 Introduction to PFF

The Portable File Format (PFF) was developed in order to make the transport of large, mixed integer and floating point data files between various computers possible, efficient, and extensible. It has been designed to store data in a **compact, binary** form in such a manner that it can be read **without translation** on a wide variety of computers. Originally developed for the QUICKSILVER family of codes, it has since been extended to handle the requirements of a wide range of diverse applications.

One of the basic assumptions upon which PFF is built is that most floating point data written to a PFF file needs to retain only graphics-like precision, i.e., arrays of data require only limited precision over the dynamic range of the data in the array. Consequently, most floating point array data can be converted to arrays of 2-byte integers before being written to the file. From this converted data, the original data can be reconstructed to approximately 16 bits of precision (1 part in 64,000) over the dynamic range of the array. The PFF format also supports higher precision floating point data; however, the benefit of data compaction is obtained primarily through the use of this reduced-precision floating-point array data.

PFF software is designed in a hierarchical manner. PFF format files are read or written using a few low-level (and possibly platform-dependent) routines to actually read and write the data to and from the file. Upon this base, machine-independent routines are available for handling some general, primitive-level data constructs. Finally the user-interface level provides routines for reading and writing several general-purpose dataset types, which are themselves constructed of combinations of the aforementioned PFF data primitives. This interface also provides routines to manage PFF files and their datasets. PFF currently supports several of these general-purpose dataset types. However, since the interface is built upon an intermediate layer of primitive, machine-independent data constructs, it possible to extend the capabilities of PFF data files without worrying about machine-dependencies.

PFF currently runs on most common computers and operating systems, including various Unix implementations, Linux, and MS Windows systems. This document describes the FORTRAN interface to PFF. A similar interface based on the C programming language is also available.

1.1 History of PFF

The PFF format grew from an effort to move 3D Particle-In-Cell (PIC) simulation data from Cray supercomputers to workstations with graphics precision. The goal was to minimize both the size of the data file and the difficulty of transporting the files between computers.

The first attempt to do this actually used an ASCII text file format in which floating point array data was converted to 16-bit integers that were then written to the file as four digit hexadecimal numbers. The advantages to this approach were that it was straightforward; techniques for translating text files between various machines were readily available. In addition, its utilization of file space was 50% (compared to much lower utilizations associated with typical text data representations, i.e., 18% for FORTRAN E11.5 formatting). Of course, it had the disadvantages that it was **only** 50% efficient and was also very slow.

Very soon after this original effort, it was determined that by encoding all data as 16-bit integers and using (from FORTRAN) ANSI-standard direct access unformatted I/O, files could be read/written on many different machines with only minimal coding variations. In addition, the files could easily be read and written from C, again with only minimal variations between machines. This approach worked quite well; it was fast, very efficient in terms of file utilization, and is in fact the basic technique employed in the present version of the PFF format. However, at the user-interface level, the format was designed with a single application in mind. Consequently, it did not provide sufficient generality to support the many other applications that could benefit from the use of machine-transportable, compact binary data files.

The current version of the PFF format has been the result of an attempt to correct some of the deficiencies of the previously described version. A large effort has been expended in generalizing the format to meet a wide variety of application requirements. In addition, the format has been made more easily extensible at the application level and is now more maintainable.

1.2 General Overview of the PFF File Format

The basic feature of the PFF file format that allows for its portability is that all data is ultimately written to (read from) the file in the form of 16-bit integers. Furthermore, these integers are written to the file as a byte-stream, with no record control. Additionally, files contain an integral number of 4096-byte blocks to facilitate efficient buffering at the machine-dependent level. Unused portions of the final block are filled with PFF's End-of-File (EOF) flag. A PFF file is composed of a PFF file header, which is followed by one or more datasets that contain user data. The last dataset in the file is followed by the file's directory (or dataset catalog). This directory contains an entry corresponding to each dataset in the file. Each directory contains the information required to identify, locate, and access its associated dataset.

1.2.1 File Header

The first 16 words (2-byte integers) of every PFF file comprise the PFF file header. This header contains information identifying the file as a PFF file, as well as providing the beginning address of the directory information that is appended to the end of the file.

1.2.2 Datasets

The user (or application) accesses data from a PFF file in self-contained units called datasets, of which there are several types. Several general-purpose dataset types are available which are designed to meet the needs of most users. However, if these dataset types are not sufficient for a specific application, PFF is extensible in the sense that it provides a means to add application-specific dataset types to the general-purpose types already available.

Each dataset type has an associated dataset format. However, each dataset type's first element is a dataset header, the format of which is independent of dataset type. This header contains some basic information about the dataset, most notably a dataset type identifier, and an optional user-supplied title. The remainder of the dataset is specific to the particular dataset

type.

1.2.3 File Directory

After the last dataset in a PFF file, the file's directory or catalog is appended. This directory is composed of a separate directory entry corresponding to each dataset in the contained in the file. (These directory entries are actually datasets in their own right; however, to avoid confusion with the other datasets in the file, they will henceforth be referred to as directory entries.) Each directory contains information required to identify, locate, and access its associated dataset. This identification includes the dataset type identifier as well as the optional title string found in the associated dataset's header section.

1.3 Using PFF

The remainder of this document provides the information needed to use the PFF format (via the FORTRAN interface) as well as to a description of how to extend its capabilities to meet the requirements of a specific application. An attempt has been made to divide the information according to the complexity of an application's demands upon the PFF format. For example, one chapter will describe using PFF for simple sequential read and write applications while another will discuss the features of PFF that facilitate user-controlled, random access to a PFF file's datasets.

Chapter 2 of this document describes in detail how one would incorporate PFF in a program that writes data to a file as well a description of how one would use PFF in another program to subsequently read that data in a sequential manner. Chapter 3 will describe how to add random, user-controlled access capability to a program that reads data from a PFF file. Chapter 4 will describe in detail how one would go about designing and implementing an application-specific PFF dataset type. Finally, Chapter 5 describes the machine-dependencies in PFF and provides a porting guide for the software.

This document also contains an Appendix section. Appendix A describes the available PFF dataset primitives. Appendix B describes the format of PFF's available dataset types. Appendix C provides documentation on many of the available PFF functions and subroutines. Appendix D contains templates for the read and write routines that must be written to support an application-specific dataset type.

2.0 Basic PFF Usage

This chapter will describe in detail how one would incorporate PFF in a program that writes data to a file. It will also describe how another program can easily access that data in a sequential manner via calls to the PFF library. It will also show how PFF's error handling facility can be directed to route PFF error messages to a user-specified file. The approach outlined here should provide a means to very rapidly include basic PFF file transport capability in an existing or new application. A discussion of more advanced PFF features that facilitate the setup of a user-friendly interface to PFF file access will be deferred to the next chapter.

Before continuing, it is useful to describe the naming convention for PFF FORTRAN modules. First, all module names follow the ANSI standard limiting module names to six or fewer characters. Further, in order to avoid conflicts with other software, the first two characters of all PFF modules are **pf**. The next (third) character is reserved for a single-letter code that categorizes the basic function of the module. For example, the letters **d**, **f**, **g**, **r**, **s**, **u**, and **w** are used for modules with **d**irectory-, **f**ile-, **g**et-, **r**ead-, **s**et-, **u**tility-, and **w**rite-related functionality, respectively. Finally, the last three characters are used to provide a mnemonic of the module's specific function.

2.1 Writing Data to a PFF file

There are three basic steps needed to write data to a PFF file. First, the file must be opened. Next, the data needs to be written to the file as one or more datasets. Finally, the file *must* be explicitly closed. PFF provides the necessary routines to perform these tasks.

2.1.1 Opening a PFF file for writing

To open a new PFF file for writing, an application need only call the **pfuopn** function. For example,

```
integer pfuopn, fid, ierr, status, dirflg
character  fname
. . .
ierr = 0
status = 1
fname = 'newfile.pff'
fid = pfuopn( fname, status, ierr, dirflg )
```

where **fname** is the name of the file to be opened. The variable **fid** returned by **pfuopn** is a PFF file ID that will be used in all subsequent PFF calls related to that file. The variable **status** describes to PFF the mode in which the file is to be opened: 0 for read, 1 for write-only, and 2 for read/write. These values are available as the parameters **RE**, **WR**, and **RW**, respectively, in the PFF include file ***pfstfg.inc***. This file is located in the PFF source directory. The variable **dirflg** is not used for files opened in write-only mode and a discussion of its use will be deferred until the section that describes reading from a PFF file. The variable **ierr** returns the error status; if nonzero, an error has occurred. A detailed description of the **pfuopn** function and its associated error codes can be found in Appendix C.

At this point, it is useful to briefly outline the use of the PFF error status flag, **ierr**. Its use is consistent throughout the PFF library. All PFF routines, on entry, examine the value of **ierr**; if nonzero, they return *immediately* without performing the requested operation. Consequently, **ierr** must be initialized to zero before the first PFF call. In addition, after a PFF call that returns a nonzero error flag, the calling program should reset **ierr** to zero before attempting a subsequent PFF call.

2.1.2 Writing data to an open PFF file

Now that the file has been successfully opened, we need to write data to it. The first step is to identify one or more of the PFF dataset types that can be used to package the data. The formats of these dataset types are described in detail in Appendix B. If some or all of the data does not naturally fit into these general dataset types, the user may want to consider developing application-specific dataset type/types, as outlined in Chapter 4.

As an example, suppose an application needed to write out some initialization information that could be loaded into an integer array **iinit(1:ni)** and two floating-point arrays **flist(1:nl)** and **finit(1:nf)**. The **flist** array needs to retain its full precision, whereas the **finit** array needs only limited precision over its dynamic range. The remainder of the data consists of scalar values from a 2D nonuniform grid which are saved at several times through the course of the application's execution. The application's code would need the following PFF library calls (assuming the file-opening calls from the previous example).

```
integer   MX, MY, MAXP, MAXS
parameter   ( MX=10, MY=20, MAXP=MX*MY, MAXS=MX+MY )
integer   iinit(ni), nx(2)
real   finit(nf), flist(nl), x(MAXS), sdata(MX,MY)
character   alabel(2)
data  nx   /  MX , MY /
data  alabel   /  'x' , 'y' /
. . .
call pfwifl( fid, 0,'initialization info', ' ', ni, nl, nf, iinit, flist, finit, ierr )
. . .
do itime=1,maxtim
   . . .
   call pfwngd ( fid, 1, '2D data', ' ', 2,1, nx, MAXP, 0, 0, alabel, ' ', x, sdata, ierr )
   . . .
end do
```

Each call to **pfwifl** or **pfwngd** (or any other PFF dataset-write routine) writes a single dataset to the PFF file identified by the file ID **fid**. In the example above, the **x** array contains the grid points in the two coordinate directions and the **sdata** array contains the actual scalar data that is located on the 2D grid defined by **x**. A complete description of the IFL and NGD dataset types, as well as a description of the arguments of their subroutine calls can be found in Appendices B and C, respectively.

2.1.3 Closing a PFF file

To enhance its performance, PFF maintains its own internal buffers. Consequently, *any PFF file that has been written to since its opening must be properly closed; otherwise data at the*

end of the file will be lost. Thus it is essential that a PFF file that has been written to be closed with PFF's file closing routine. For example, to close the file opened and written to in the previous example, use

call pfucls (fid, ierr).

This will insure that the any partial buffers associated with the file are in fact written to the file, and that the file's directory information is properly appended to the end of the file.

2.2 Sequentially Reading Data from an Existing PFF file

As in writing a PFF file, there are three basic steps needed to read data from an existing PFF file. The file is opened by the same PFF routine, only the access status need be changed. For each PFF routine that writes a particular dataset type, there is a corresponding routine to read that dataset type from a PFF file. If the order in which datasets were originally written to a PFF file is known, then it is a simple matter to call these dataset-read routines to extract the data from the file. Finally, the file should be closed. Although this is not strictly necessary for files opened in read-only mode, it is a good practice to close such files anyway. This is particularly true when dealing with multiple PFF files, since there is a limited number of PFF files which may be open at any one time. The following section of code demonstrates how one might read the file that was written in the example of the previous section. For brevity, the typing of variables used in the previous example in not repeated and is assumed to be the same.

```
integer   IDIM, MDIM, FLDIM, FADIM, NDIM, XDIM, VDIM
integer   ierr, status, dirflg, ni, nl, nf, fid, tapp, flo, fao, xoff, soff, m, n, nvp, ispare
integer   iinit(IDIM), ioff(MDIM), nx(MDIM)
real      flist(FLDIM), finit(FADIM), x(XDIM), sdata(VDIM, NDIM)
character  st*16, sc*64, vlabel*16, fname*16
. . .
ierr = 0
status = 0
fname = 'newfile.pff'
fid = pfuopn( fname, status, ierr, dirflg )
. . .
call pfrifl( fid, .FALSE., IDIM, FLDIM, FADIM, 1, ni, nl, nf, tapp, st, sc, iinit, flist, flo,
&              finit, fao, ierr )
. . .
do while (ierr.eq.0)
              . . .
   call pfrngd ( fid, .FALSE., MDIM, NDIM, XDIM, VDIM, 1, tapp, st, sc, m, n, nx, nvp,
&                    1, ispare, alabel, vlabel, x, xoff, sdata, soff, ioff, ierr )
   if (ierr.eq.0) then
      . . .
   endif
end do
if (ierr.ne.-1) then
   print *, 'Error encountered reading grid data
endif
call pfucls ( fid, ierr )
```

As before, the file must first be open; in this case the **status** flag in the **pfuopn** call indicates read-only mode. The variable **dirflg** returns the number of datasets that are in the file.

Then the PFF routine to read an IFL dataset is called to access the initialization information. Notice that PFF dataset-read routines take array dimensioning information as input in order to avoid overwriting the dimensioned array space of the calling program. Next the PFF routine to read NGD data is called repeatedly* until the End-of-File error code (-1) is returned. For a complete description of the arguments to these PFF routines, see Appendix C.

2.3 Using PFF's Error-Handling Facility

As described in Section 2.1.1 above, all PFF routines return with no operation if the entry value of the error flag **ierr** is non-zero. While this allows the user to safely avoid checking the error flag after every PFF call, it makes subsequent processing of a returned error flag potentially ambiguous. PFF provides three subroutines to help the user deal with this problem.

First is the routine **pfgerr**, which returns information about the last error detected by any PFF routine. This information includes the name of the routine that encountered the error, the error number, the File ID associated with the error (if any), as well as other more detailed information. This routine is described in detail in Appendix C.

The PFF library also provides two functions that will write detailed error and/or debugging information to a specified file. The PFF routine **pfsvrb** (set-verbose) allows the user to specify a FORTRAN unit number to which any PFF routine encountering an error will write an error message. The PFF routine **pfsdbg** (set-debug) allows the user to specify a unit to which every PFF routine writes messages detailing the entry and exit values of the error flag **ierr**, in addition to messages that detail any errors that are encountered. It should be noted that such a unit must be open at the time of the call in the sense that the **opened** value returned by a FORTRAN inquire statement will indicate that the unit is in fact open.

Note that the PFF routines **pfsvrb** and **pfsdbg** can be used to route messages to the terminal screen by using a unit number that is associated with terminal output. However, since implicit or explicit mapping of units to terminal output is not standard, this will require different implementations on different computers. In addition, on many computers, although a unit might be implicitly associated with terminal output, some *explicit* action might be required to insure that the file will be open. On many systems this can be accomplished by writing a null character to the unit before calling **pfsvrb** or **pfsdbg**. For example, on such systems, the following code:

```
open ( unit=10, file='debug')
write ( 6, '(a,$)' ) char(0)
call pfsvrb ( 6, ierr )
call pfsdbg ( 10, ierr )
```

will send error messages to the terminal and route the more detailed output from **pfsdbg** to a file named ***debug***.

---

* Note that a **do while** loop is used to perform this repeated operation in the example. Strictly, the **do while** statement is not part of the Fortran 77 standard, but is accepted by all modern Fortran 77 compilers, and is included in the Fortran 90 standard. We use it here to avoid the use of statement labels and **goto** statements, which is the only way to accomplish this using standard Fortran 77.

3.0 Using Advanced Features of PFF

This chapter will describe how one might use some of the more advanced features of PFF. It will provide examples of how to add user-controlled, random access capability to a program that reads data from one or more PFF files and describe various routines designed to make the implementation of PFF file handling in a user-friendly command interface. In addition, the sample application **bldpff**, distributed with the PFF software, provides a useful template for writing applications that used these features of PFF.

3.1 PFF File Management

In the previous chapter on basic PFF usage, the routines to open and close files, **pfuopn** and **pfucls**, were described. However, PFF provides many other routines to assist the user in the management of PFF files. However, before demonstrating the use of these routines it is helpful to briefly describe the way in which PFF handles files internally.

As described earlier, each PFF format file is assigned a unique File ID (or **fid**) when it is opened. This **fid** is permanently associated with the file until that file is closed. Once a file has been closed, PFF's file handler will reuse the released **fid**. If one imagines an application that allows the user to open and close files interactively, it is easy to see that by allowing the user to manipulate files by their **fid**'s could become quite confusing. To avoid this problem, PFF maintains an open file list upon which all currently-open files are assigned consecutive file entry numbers (**fen**'s). As files are opened and closed, a file's **fen** will change, but its **fid** will remain constant. However, the user can always be presented with a consecutively-numbered list of files that remain ordered in the chronological sequence in which they were opened. In addition to this list, PFF also maintains a pointer to the currently-active file entry and provides routines by which an application can use and manipulate this pointer.

One difficulty that can arise when developing a PFF application is that typically the application will need to read and write other files that are not in PFF format. Since the application has no control (and in fact no knowledge) of the FORTRAN logical unit numbers that PFF is using for the files that it is managing, the application must have some way to avoid using a currently active unit number. The PFF library provides the function **pfg1lu** that returns a unit number, within an application-specified range, that is guaranteed to be currently unused. Since PFF uses this same routine to obtain its own unit numbers, it will never attempt to open a unit that the application already has open.

3.2 PFF Dataset Management

3.3 Other Useful PFF Features

4.0 Design and Implementation of Application-Specific PFF Dataset Types

Appendix A PFF Data Primitives

Appendix B PFF Dataset Formats

Appendix C PFF Fortran Interface Functions and Subroutines

C.1 PFF Utility Functions

pfuopn

Open a pff file — This function is used to open a *directoried* PFF file. It finds an available open PFF File ID (*fid*), assigns the file "fname" to that *fid*, opens the file, loads in its directory, and returns *fid*.

```
integer function pfuopn ( fname, status, ierr, dirflg )
integer      status, ierr, dirflg
character    fname*(*)
```

Input:

fname	character variable containing name of file to be opened
status	file status: 0 for read (RE) 1 for write (WR) 2 for read/write (RW)
ierr	if not zero, return with no operation

Output:

dirflg	number of directory entries loaded from file (RE-RW mode only)
ierr	error status flag:0, No error 1, No Available PFF File ID's 2, No Available Logical Units 3, Error Opening File 4, Invalid Status 5, File opened for read is empty 6, File has no PFF header

Return Value:

< 0,	error has occurred, returns (**-ierr**)
0,	error has occurred in lower level PFF routine
> 0,	returns PFF File ID # of file

pfucls

Close an open pff file — This routine is used to close a *directoried* PFF file. It closes PFF file **fid** and resets all the buffer management pointers, etc. For a file in write mode, it also pads and flushes the buffer if it is not empty. If the extend flag has been set, the file's directory is appended to the end of the file.

```
subroutine pfucls ( fid, ierr )
integer     fid, ierr
```

Input:

fid File ID of PFF file to be closed. If fid = 0, then close all open PFF files.

ierr If not zero, return with no operation.

Output:

ierr error status flag:0, No error
1, Illegal PFF File ID
2, File Not Open
3, File not on FID map

pfsvrb

Route error messages to a Fortran logical unit — This routine is used to turn VERBOSE mode on and off. In VERBOSE mode, error messages are written to the user-supplied Fortran logical unit.

```
subroutine pfsvrb (lun,ierr)
integer    lun, ierr
```

Input:

lun	if **lun** = -100, turn off VERBOSE mode, otherwise, turn on VERBOSE mode, using logical unit **lun** for VERBOSE output
ierr	if not zero, return with no operation

Output:

ierr	error status flag:0, No error 1, VERBOSE File not Opened

pfgerr

Returns error information for the last error encountered by PFF — This routine is used to retrieve the error flags in the PFERRS common block.

```
subroutine pfgerr (modul, number, finfo, fname, clear)
integer    finfo(5), number
character  fname*(*), modul*(*)
logical    clear
```

Input:

clear if true, clear the error flags after reading them

Output:

modul module name of PFF function or routine last encountering an error.

number error #.

finfo array containing information about file associated with error:

- **finfo(1)** PFF File ID on which the error occurred (if .le. 0, no file was associated with the error)
- **finfo(2)** buffer loc. of file
- **finfo(3)** Unit # of file
- **finfo(4)** record # of file
- **finfo(5)** I/O status of file

fname file name associated with last reported error

C.2 PFF Dataset Write Functions

pfwifl

Write an integer/float list dataset to a PFF file — This routine is a WRITE routine that writes a IFL (Integer/Float List) dataset to a PFF file. Floating data is divided into two groups:

1) **Float List** — each value is encoded independently as a **<FLOAT>** at the full precision of this data type.
2) **Float Array** — the entire array is encoded as an **<FARRAY>**. This uses less space but has dynamic range limitations for data with multi-order-of-magnitude variations.

This operation is ***only allowed*** in WRITE mode. The format for this dataset type is described in detail in Appendix B.

```
subroutine pfwifl ( fid, tapp, st, sc, ni, nfl, nfa, iarray, flist, farray, ierr )
integer     fid, ierr, nfa, nfl, ni, tapp
integer     iarray(*)
real        farray(*), flist(*)
character   st*(*), sc*(*)
```

Input:

fid	File ID of PFF file
farray	floating point array
flist	floating point list
iarray	integer array
nfa	length of float array (**farray**)
nfl	length of float list (**flist**)
ni	length of integer array (**iarray**)
sc	character string containing dataset comment
st	character string containing dataset type label
tapp	# associated with application dataset type
ierr	if not zero, return with no operation

Output:

ierr	error status flag	0, No error
		otherwise, Error returned by called PFF routine

pfwuf1

Write a 1D, multiple-block, uniform-grid, scalar dataset to a PFF file — This routine is used to write a UF1 (uniform, 1D, Floating) dataset to a PFF file. The input array is assumed to be blocked. The dimensioned size of each block must be provided as well as a do-list for each block describing the subset of its data to be written to file. The value **x0** is associated with the lowest dimensioned point (**low**) in **farray** for each block. Similarly, **dx** is the grid separation of **farray** for each block. For any block, if **idlist(1)** > **low**, the value of **x0** written to the file is adjusted to correspond to the grid location **idlist(1)**. Similarly, if **idlist(3)** > 1, the value of **dx** written to the file is adjusted to be **idlist(3)** times the input value of **dx**. This operation is ***only allowed*** in WRITE mode. The format for this dataset type is described in detail in Appendix B.

```
subroutine pfwuf1 ( fid, tapp, st, sc, nblks, ispare, x0, dx, xlabel, blabel, locb, low,
                    imax, idlist, farray, ierr )
integer     fid, ierr, low, nblks, tapp
integer     ispare(5,nblks), locb(nblks)
integer     imax(nblks), idlist(3,nblks)
real        farray(low:*), dx(nblks), x0(nblks)
character   st*(*), sc*(*)
character   xlabel(nblks)*(*), blabel(nblks)*(*)
```

Input:

blabel	Title label for each block
dx	grid separation for each block
farray	linear array containing floating point data
fid	File ID of PFF file
ispare	5 spare words per block reserved for the use of the application
idlist	do-list array for each block of **farray**, e.g., for the **m**th block, **do i=idlist(1,m),idlist(2,m),idlist(3,m)**
imax	upper dimension of **farray** for each block
locb	indices into linear array **farray** identifying the first data value in each block
low	lower dimension of each block of **farray**
nblks	number of data blocks contained in **farray**
sc	character string containing dataset comment
st	character string containing dataset type label
tapp	number associated with application dataset type
x0	initial grid value for each block
xlabel	Grid label for each block
ierr	if not zero, return with no operation

Output:

ierr	error status flag:0, No error 1, Illegal DO-List

pfwuf3

Write a 3D, multiple-block, uniform-grid, scalar dataset to a PFF file — This routine is used to write a UF3 (uniform, 3D, Floating) dataset to a PFF file. The input array is assumed to be composed for multiple blocks, each representing a 3D set of data. The dimensioned size of each 3D block must be provided as well as a do-list in each of the three coordinate directions for each block describing the subset of its data to be written to file. The value **x0** is associated with the lowest dimensioned point (**low**) in the *x* direction of **farray** for each block. Similarly, **dx** is the grid separation in the *x* direction of **farray** for each block. For any block, if **idlist(1)** > **low**, the value of **x0** written to the file is adjusted to correspond to the grid location **idlist(1)**. Similarly, if **idlist(3)** > 1, the value of **dx** written to the file is adjusted to be **idlist(3)** times the input value of **dx**. The arrays **y0**, **z0**, **dx**, **dy**, **jdlist**, and **kdlist** are similarly defined for the *y* and *z* coordinate directions. This operation is ***only allowed*** in WRITE mode. The format for this dataset type is described in detail in Appendix B.

```
subroutine pfwuf3 ( fid, tapp, st, sc, nblks, ispare, x0, dx, y0, dy, z0, dz, xlabel, ylabel,
                    zlabel, blabel, locb, low, imax, jmax, kmax, idlist, jdlist, kdlist, farray,
                    ierr )
integer     fid, ierr, low, nblks, tapp
integer     ispare(5,nblks), locb(nblks)
integer     imax(nblks), jmax(nblks), kmax(nblks)
integer     idlist(3,nblks), jdlist(3,nblks), kdlist(3,nblks)
real        farray(low:*)
real        x(nblks), dy(nblks), dz(nblks)
real        x0(nblks), y0(nblks), z0(nblks)
character   st*(*), sc*(*)
character   xlabel(nblks)*(*), ylabel(nblks)*(*)
character   zlabel(nblks)*(*), blabel(nblks)*(*)
```

Input:

blabel	Title label for each block
dx	grid separation in the *x* direction for each block
dy	grid separation in the *y* direction for each block
dz	grid separation in the *z* direction for each block
farray	linear array containing floating point data
fid	File ID of PFF file
ispare	5 spare words per block reserved for the use of the application
idlist	do-list array for the *x* coordinate direction of each block of **farray**, e.g., for the **m**th block, **do i=idlist(1,m),idlist(2,m),idlist(3,m)**
imax	upper dimension for the *x* coordinate direction of **farray** for each block
jdlist	do-list array for the *y* coordinate direction of each block of **farray**, e.g., for the **m**th block, **do j=jdlist(1,m),jdlist(2,m),jdlist(3,m)**
jmax	upper dimension for the *y* coordinate direction of **farray** for each block
kdlist	do-list array for the *z* coordinate direction of each block of **farray**, e.g., for the **m**th block, **do k=kdlist(1,m),kdlist(2,m),kdlist(3,m)**
kmax	upper dimension for the *z* coordinate direction of **farray** for each block
locb	indices into linear array **farray** identifying the first data value in each block

low	lower dimension for all three coordinate directions of each block of **farray**
nblks	number of data blocks contained in **farray**
sc	character string containing dataset comment
st	character string containing dataset type label
tapp	number associated with application dataset type
x0	initial *x* grid value for each block
xlabel	*x* grid label for each block
y0	initial *y* grid value for each block
ylabel	*y* grid label for each block
z0	initial *z* grid value for each block
zlabel	*z* grid label for each block
ierr	if not zero, return with no operation

Output:

ierr	error status flag:0, No error 1, Illegal DO-List

pfwng3

Write a 3D, multiple-block, nonuniform-grid dataset to a PFF file — This routine is used to write a NG3 (nonuniform, 3D, Grid) dataset to a PFF file. The input array is assumed to be composed for multiple blocks, each representing a 3D set of data. The dimensioned size of each 3D block must be provided as well as a do-list in each of the three coordinate directions for each block describing the subset of its data to be written to file. This operation is ***only allowed*** in WRITE mode. The format for this dataset type is described in detail in Appendix B.

```
subroutine pfwng3 ( fid, tapp, st, sc, nblks, mspare, nspare, ispare, x, y, z, xlabel, ylabel,
                    zlabel, blabel, locx, locy, locz, low, max, jmax, kmax, idlist, jdlist,
                    kdlist, ierr )
integer     fid, ierr, low, nblks, mspare, nspare, tapp
integer     ispare(mspare,nblks)
integer     locx(nblks), locy(nblks), locz(nblks)
integer     imax(nblks), jmax(nblks), kmax(nblks)
integer     idlist(3,nblks), jdlist(3,nblks), kdlist(3,nblks)
real        x(low:*), y(low:*), z(low:*)
character   st*(*), sc*(*)
character   xlabel(nblks)*(*), ylabel(nblks)*(*)
character   zlabel(nblks)*(*), blabel(nblks)*(*)
```

Input:

blabel	Title label for each block
id	File ID of PFF file
ispare	multiple spare words reserved for application for each block
idlist	do-list array for each block of **x**, e.g., for the **m**th block, **do i=idlist(1,m),idlist(2,m),idlist(3,m)**
imax	upper dimension for the *i* coordinate direction of grid for each block
jdlist	do-list array for each block of **y**, e.g., for the **m**th block, **do k=jdlist(1,m),jdlist(2,m),jdlist(3,m)**
jmax	upper dimension for the *j* coordinate direction of grid for each block
kdlist	do-list array for each block of **z**, e.g., for the **m**th block, **do k=kdlist(1,m),kdlist(2,m),kdlist(3,m)**
kmax	upper dimension for the *k* coordinate direction of grid for each block
locx	indices into linear array **x** identifying the first data value in each block
locy	indices into linear array **y** identifying the first data value in each block
locz	indices into linear array **z** identifying the first data value in each block
low	lower dimension for each block of **x**, **y**, and **z** arrays
nblks	number of data blocks to be written to file
mspare	1st dimension of **ispare** array (**mspare** ≥ **nspare**)
nspare	number of spare words used in each block
sc	character string to be loaded with dataset comment
st	character string to be loaded with dataset type label
tapp	number associated with application dataset type

x	linear array containing grid data in *x* direction for each block
xlabel	*x* grid direction label for each block
y	linear array containing grid data in *y* direction for each block
ylabel	*y* grid direction label for each block
z	linear array containing grid data in *z* direction for each block
zlabel	*z* grid direction label for each block
ierr	if not zero, return with no operation

Output:

ierr	error status flag:0, No error 1, Illegal DO-List

pfwnf3

Write a 3D, multiple-block, nonuniform-grid, scalar dataset to a PFF file — This routine is used to write a NF3 (nonuniform, 3D, Floating) dataset to a PFF file. The input array is assumed to be composed for multiple blocks, each representing a 3D set of data. The dimensioned size of each 3D block must be provided as well as a do-list in each of the three coordinate directions for each block describing the subset of its data to be written to file. This operation is ***only allowed*** in WRITE mode. The format for this dataset type is described in detail in Appendix B.

```
subroutine pfwnf3 ( fid, tapp, st, sc, nblks, ispare, x, y, z, xlabel, ylabel, zlabel, blabel, locx,
                    locy, locz, locb, low, imax, jmax, kmax, idlist, jdlist, kdlist, farray, ierr )
integer     fid, ierr, low, nblks, tapp
integer     ispare(5,nblks)
integer     locb(nblks), locx(nblks), locy(nblks), locz(nblks)
integer     imax(nblks), jmax(nblks), kmax(nblks)
integer     idlist(3,nblks), jdlist(3,nblks), kdlist(3,nblks)
real        farray(low:*)
real        x(low:*), y(low:*), z(low:*)
character   st*(*), sc*(*)
character   xlabel(nblks)*(*), ylabel(nblks)*(*)
character   zlabel(nblks)*(*), blabel(nblks)*(*)
```

Input:

blabel	Title label for each block
farray	linear array containing floating point data
id	File ID of PFF file
ispare	5 spare words reserved for application for each block
idlist	do-list array for each block of **x** and the *x* coordinate direction of **farray**, e.g., for the **m**th block, **do i=idlist(1,m),idlist(2,m),idlist(3,m)**
imax	upper dimension for the *i* coordinate direction of grid for each block
jdlist	do-list array for each block of **y** and the *y* coordinate direction of **farray**, e.g., for the **m**th block, **do k=jdlist(1,m),jdlist(2,m),jdlist(3,m)**
jmax	upper dimension for the *j* coordinate direction of grid for each block
kdlist	do-list array for each block of **z** and the *z* coordinate direction of **farray**, e.g., for the **m**th block, **do k=kdlist(1,m),kdlist(2,m),kdlist(3,m)**
kmax	upper dimension for the *k* coordinate direction of grid for each block
locb	indices into linear array **farray** identifying the first data value in each block
locx	indices into linear array **x** identifying the first data value in each block
locy	indices into linear array **y** identifying the first data value in each block
locz	indices into linear array **z** identifying the first data value in each block
low	lower dimension for each block of **x**, **y**, and **z** arrays and for the *i*, *j*, and *k* directions in each block of **farray**
nblks	number of data blocks to be written to file
sc	character string to be loaded with dataset comment
st	character string to be loaded with dataset type label

tapp	number associated with application dataset type
x	linear array containing grid data in *x* direction for each block
xlabel	*x* grid direction label for each block
y	linear array containing grid data in *y* direction for each block
ylabel	*y* grid direction label for each block
z	linear array containing grid data in *z* direction for each block
zlabel	*z* grid direction label for each block
ierr	if not zero, return with no operation

Output:

ierr	error status flag:0, No error 1, Illegal DO-List

pfwnv3

Write a 3D, multiple-block, nonuniform-grid, 3D vector dataset to a PFF file — This routine is used to write a NF3 (nonuniform, 3D, Vector) dataset to a PFF file. The input array is assumed to be composed for multiple blocks, each representing a 3D set of data. The dimensioned size of each 3D block must be provided as well as a do-list in each of the three coordinate directions for each block describing the subset of its data to be written to file. This operation is ***only allowed*** in WRITE mode. The format for this dataset type is described in detail in Appendix B.

```
subroutine pfwnv3 ( fid, tapp, st, sc, nblks, ispare, x, y, z, xlabel, ylabel, zlabel, blabel,
                    imax, jmax, kmax, idlist, jdlist, kdlist, vx, vy, vz, ierr )
integer     fid, ierr, low, nblks, tapp
integer     ispare(5,nblks)
integer     locb(nblks), locx(nblks), locy(nblks), locz(nblks)
integer     imax(nblks), jmax(nblks), kmax(nblks)
integer     idlist(3,nblks), jdlist(3,nblks), kdlist(3,nblks)
real        vx(low:*), vy(low:*), vz(low:*)
real        x(low:*), y(low:*), z(low:*)
character   st*(*), sc*(*)
character   xlabel(nblks)*(*), ylabel(nblks)*(*)
character   zlabel(nblks)*(*), blabel(nblks)*(*)
```

Input:

blabel	Title label for each block
fid	File ID of PFF file
ispare	Array of 5 spare words reserved for application for each block
idlist	do-list array for each block of **x** and the *x* coordinate direction of **vx**, **vy**, and **vz**, e.g., for the **m**th block, **do i=idlist(1,m),idlist(2,m),idlist(3,m)**
imax	upper dimension for the *i* coordinate direction of grid for each block
jdlist	do-list array for each block of **y** and the *y* coordinate direction of **vx**, **vy**, and **vz**, e.g., for the **m**th block, **do k=jdlist(1,m),jdlist(2,m),jdlist(3,m)**
jmax	upper dimension for the *j* coordinate direction of grid for each block
kdlist	do-list array for each block of **z** and the *z* coordinate direction of **vx**, **vy**, and **vz**, e.g., for the **m**th block, **do k=kdlist(1,m),kdlist(2,m),kdlist(3,m)**
kmax	upper dimension for the *k* coordinate direction of grid for each block
locb	indices into linear array **vx**, **vy**, and **vz** identifying the first data value in each block
locx	indices into linear array **x** identifying the first data value in each block
locy	indices into linear array **y** identifying the first data value in each block
locz	indices into linear array **z** identifying the first data value in each block
low	lower dimension for each block of **x**, **y**, and **z** arrays and for the *i*, *j*, and *k* directions in each block of **vx**, **vy**, and **vz**
nblks	number of data blocks to be written to file
sc	character string to be loaded with dataset comment
st	character string to be loaded with dataset type label

tapp	number associated with application dataset type
vx	linear array containing *x* component of vector floating point data
vy	linear array containing *y* component of vector floating point data
vz	linear array containing *z* component of vector floating point data
x	linear array containing grid data in *x* direction for each block
xlabel	*x* grid direction label for each block
y	linear array containing grid data in *y* direction for each block
ylabel	*y* grid direction label for each block
z	linear array containing grid data in *z* direction for each block
zlabel	*z* grid direction label for each block
ierr	if not zero, return with no operation

Output:

ierr	error status flag:0, No error 1, Illegal DO-List

pfwngd

Write a dataset containing an *n*-dimensional vector of an *m*-dimensional, nonuniformly gridded space to a PFF file — This routine is used to write a NGD (nonuniform grid data) dataset to a PFF file. The input array can only be composed of a single block. This single block represents *n*-dimensional vectors on a parallelepiped region of an *m*-dimensional space This operation is ***only allowed*** in WRITE mode. The format for this dataset type is described in detail in Appendix B.

```
subroutine pfwngd ( fid, tapp, st, sc, m, n, nx, mg, nspare, ispare, alabel, vlabel, x, v, ierr )
integer     fid, ierr, m, mg, n, nspare, tapp
integer     ispare(1:nspare), nx(1:m)
real        x(1:*)
real        v(1:mg,1:n)
character   st*(*), sc*(*)
character   alabel(m)*(*), vlabel(n)*(*)
```

Input:

alabel	Axis labels for each axis of the *m*-dimensional space
fid	File ID of PFF file
ispare	Array of multiple spare words reserved for application
m	space dimensionality
n	vector dimensionality
mg	Dimension size of first index of **v** array. It must exceed the product of the **m** values in the **nx** array.
nspare	number of spare words
nx	number of grid points along each axis of space
sc	character string to be loaded with dataset comment
st	character string to be loaded with dataset type label
tapp	number associated with application dataset type
x	Packed Grid array. Each dimension's grid is stored consecutively, i.e., x_1(1:**nx**(1)) -> **x**(1:**nx**(1)), x_2(1:**nx**(2)) -> **x**(**nx**(1)+1,**nx**(1)+**nx**(2)) etc. Consequently, **x** must be dimensioned ≥ **nx**(1) + ... + **nx**(**m**)
v	Packed vector data. Data for each vector component is stored as a packed *m*-dimensional array, i.e., **v**(**nx**(1),**nx**(2),**nx**(3),...). Consequently, **mg** must be ≥ **nx**(1) * **nx**(2) * ... * **nx**(**m**)
vlabel	labels for each of the **n** components of the data in **v**
ierr	if not zero, return with no operation

Output:

ierr	error status flag:0, No error 1, Vector array dimension **mg** too small otherwise, Error in called PFF routine

pfwvtx

Write a list of *m*-dimensional points, each with *n* attributes, to a PFF file — This routine is used to write a VTX (vertex data) dataset to a PFF file. This operation is ***only allowed*** in WRITE mode. The format for this dataset type is described in detail in Appendix B.

```
subroutine pfwvtx ( fid, tapp, st, sc, m, n, nv, ispare, vlabel, alabel, vdim, vert, attr,   ierr )
integer     fid, ierr, m, n, nv, tapp, vdim
integer     ispare(5)
real        vert(1:m,1:nv), attr(1:vdim,1:n)
character   st*(*), sc*(*)
character   vlabel(m)*(*), alabel(n)*(*)
```

Input:

alabel	Array of **n** attribute labels
attr	2D array containing attribute data
fid	File ID of PFF file
ispare	Array of 5 spare words reserved for application
m	spatial dimensionality of each vertex
n	number of attributes defined at each vertex
nv	number of vertices in vertex list
sc	character string to be loaded with dataset comment
st	character string to be loaded with dataset type label
tapp	number associated with application dataset type
vdim	dimensioned size of vertex list in calling program [**attr**(1:**vdim**,*)]
vert	packed *m*-dimensional vertex list array [(1:**m**,1:**nv**)]
vlabel	Array of **m** grid coordinate labels
ierr	if not zero, return with no operation

Output:

ierr	error status flag:	0, No error
		otherwise, Error in called PFF routine

C.3 PFF Dataset Read Functions

pfrifl

Read a 1D, multiple-block, uniform-grid, scalar dataset from a PFF file — This routine is a READ routine that reads a IFL (Integer/Float List) dataset from a PFF file. Floating data is divided into two groups:

3) **Float List** — each value is encoded independently as a **<FLOAT>** at the full precision of this data type.
4) **Float Array** — the entire array is encoded as an **<FARRAY>**. This uses less space but has dynamic range limitations for data with multi-order-of-magnitude variations.

This operation is ***only allowed*** in READ mode. The format for this dataset type is described in detail in Appendix B.

```
subroutine pfrifl ( fid, keep, idim, fldim, fadim, offdim, ni, nfl, nfa, tapp, st, sc, iarray, flist,
                    floff, farray, foff10, ierr )
integer     fid, fadim, fldim, foff10, idim, ierr, nfa, nfl, ni, offdim, tapp
integer     floff(*), iarray(*)
logical     keep
real        farray(*), flist(*)
character   st*(*), sc*(*)
```

Input:

fid	File ID of PFF file
fadim	dimensioned length of floating array **farray** in calling program
fldim	dimensioned length of floating list **flist** in calling program
idim	dimensioned length of integer array **iarray** in calling program
keep	logical flag indicating whether or not to keep a non-zero value for floating data in the case of underflow.
offdim	dimensioned length of float list offsets **floff**
ierr	if not zero, return with no operation

Output:

farray	floating point array
flist	floating point list

floff	power-of-ten offset for floating list elements:
	for i=1,min(**nfl**,**fldim**) {
	if (i ≤ **offdim**) then {
	floff(i) = "*power-of-10 offset of* **flist**(i)"
	}
	else if ("*power-of-10 offset of* **flist**(i)" ≠ 0) then {
	floff(**offdim**) = 1
	}
	}

foff10	ower-of-ten offset for floating array **farray**
iarray	integer array

nfa	length of float array (**farray**)
nfl	length of float list (**flist**)
ni	length of integer array (**iarray**)
sc	character string containing dataset comment
st	character string containing dataset type label
tapp	# associated with application dataset type
ierr	error status flag 0, No error -1, EOF marker encountered (Not really an error) 1, Incorrect dataset type 2, Insufficient array dimension -- array truncated

pfruf1

Read a 1D, multiple-block, uniform-grid scalar dataset from a PFF file — This routine is used to read a UF1 (uniform, 1D, Floating) dataset from a PFF file. The output array is assumed to be blocked. This operation is ***only allowed*** in READ mode. The format for this dataset type is described in detail in Appendix B.

```
subroutine pfruf1 ( fid, mode, keep, mblks, mdim, tapp, st, sc, nblks, nx, ispare, x0, dx,
                    goff10, xlabel, blabel, locfa, farray, foff10, ierr )
integer     fid, foff10, goff10, ierr, mode, mblks, mdim, nblks, tapp
integer     ispare(5,nblks), lenfa(nblks), locfa(nblks), nx(nblks)
logical     keep
real        farray(low:*), dx(nblks), x0(nblks)
character   st*(*), sc*(*)
character   xlabel(nblks)*(*), blabel(nblks)*(*)
```

Input:

fid	File ID of PFF file
keep	logical flag indicating whether or not to keep a non-zero value for floating data in the case of underflow.
mblks	dimensioned length of arrays over blocks in calling program
mdim	dimensioned length of floating array **farray** in calling program
mode	subroutine mode flag: if **mode** = 0, return floating array in **farray** otherwise, return pointer and length in **locfa** and **lenfa**
ierr	if not zero, return with no operation

Output:

blabel	Title label for each block
dx	grid separation for each block
farray	linear array containing floating data (**mode** = 0 only)
foff10	power-of-ten offset for each block of floating array **farray**
goff10	power-of-ten offset for each block's grid definition values **x0** and **dx**
ispare	5 spare words per block reserved for the use of the application
locfa	pointer to floating array for each block: **mode** = 0, indices into linear array **farray** identifying the first data value in each block **mode** = 1, PFF file pointer to **<FARRAY>** primitive for this block
nblks	number of data blocks read from file
nx	number of grid values for each block
sc	character string containing dataset comment
st	character string containing dataset type label
tapp	number associated with application dataset type
x0	initial grid value for each block
xlabel	Grid label for each block
ierr	error status flag 0, No error

-1, EOF marker encountered (Not really an error)
1, Incorrect dataset type
2, Insufficient array dimension -- array truncated

pfruf3

Read a 3D, multiple-block, uniform-grid scalar dataset from a PFF file — This routine is used to read a UF3 (uniform, 3D, Floating) dataset from a PFF file. This operation is ***only allowed*** in READ mode. The format for this dataset type is described in detail in Appendix B. This routine operates in two modes:

```
subroutine pfruf3 ( fid, mode, keep, mblks, mdim, tapp, st, sc, nblks, nx, ny, nz, ispare,
                    x0, dx, y0, dy, z0, dz, goff10, xlabel, ylabel, zlabel, blabel, locfa, lenfa,
                    farray, foff10, ierr )
integer     fid, foff10(nblks), goff10(nblks), ierr, mode, mblks, mdim, nblks, tapp
integer     ispare(5,nblks), lenfa(nblks), locfa(nblks), nx(nblks) , ny(nblks) , ny(nblks)
logical     keep
real        farray(low:*), dx(nblks), dy(nblks), dz(nblks), x0(nblks) , y0(nblks) , z0(nblks)
character   st*(*), sc*(*)
character   xlabel(nblks)*(*), ylabel(nblks)*(*), zlabel(nblks)*(*), blabel(nblks)*(*)
```

Input:

fid File ID of PFF file

keep logical flag indicating whether or not to keep a non-zero value for floating data in the case of underflow.

mblks dimensioned length of arrays over blocks in calling program

mdim dimensioned length of floating array **farray** in calling program

mode subroutine mode flag:
if **mode** = 0, return floating array in **farray**
otherwise, return pointer and length in **locfa** and **lenfa**

ierr if not zero, return with no operation

Output:

blabel Title label for each block

dx grid separation for the *x* coordinate of each block

dy grid separation for the *y* coordinate of each block

dz grid separation for the *z* coordinate of each block

farray linear array containing floating data (**mode** = 0 only)

foff10 power-of-ten offset for each block of floating array **farray**

goff10 power-of-ten offset for each block's grid definition values **x0**, **y0**, **z0**, **dx**, **dy**, and **dz**

ispare 5 spare words per block reserved for the use of the application

lenfa length of floating array for each block

locfa pointer to floating array for each block:
mode = 0, indices into linear array **farray** identifying the first data value in each block
mode = 1, PFF file pointer to **<FARRAY>** for this block

nblks number of data blocks read from file

nx number of grid values in the *x* coordinate for each block

ny number of grid values in the *y* coordinate for each block

nz	number of grid values in the *z* coordinate for each block
sc	character string containing dataset comment
st	character string containing dataset type label
tapp	number associated with application dataset type
x0	initial grid value for the *x* coordinate of each block
xlabel	Grid label for the *x* coordinate of each block
y0	initial grid value for the *y* coordinate of each block
ylabel	Grid label for the *y* coordinate of each block
z0	initial grid value for the *z* coordinate of each block
zlabel	Grid label for the *z* coordinate of each block
ierr	error status flag 0, No error -1, EOF marker encountered (Not really an error) 1, Incorrect dataset type 2, Insufficient array dimension -- array truncated

pfrng3

Read a 3D, multiple-block, nonuniform-grid dataset from a PFF file — This routine is used to read a NG3 (nonuniform, 3D, Grid) dataset from a PFF file. This operation is ***only allowed*** in READ mode. The format for this dataset type is described in detail in Appendix B.

```
subroutine pfrng3 ( fid, keep, mblks, mgrdx, mgrdy, mgrdz, mspare, tapp, st, sc, nblks, nx,
                    ny, nz, nspare, ispare, x, y, z, goff10, xlabel, ylabel, zlabel, blabel, locx,
                    locy, locz, ierr )
logical     keep
integer     fid, ierr, mblks, mgrdx, mgrdy, mgrdz, mspare
integer     nblks, tapp, nspare
integer     goff10(mblks)
integer     ispare(mspare,mblks)
integer     locx(mblks), locy(mblks), locz(mblks)
integer     nx(mblks), ny(mblks), nz(mblks)
real        x(mgrdx), y(mgrdy), z(mgrdz)
character   st*(*), sc*(*)
character   xlabel(mblks)*(*), ylabel(mblks)*(*)
character   zlabel(mblks)*(*), blabel(mblks)*(*)
```

Input:

fid	File ID of PFF file
keep	logical flag indicating whether or not to keep a non-zero value for floating data in the case of underflow.
mblks	dimensioned length of arrays over blocks in calling program
mgrdx	dimensioned length of **x** grid array
mgrdy	dimensioned length of **y** grid array
mgrdz	dimensioned length of **z** grid array
mspare	1st dimension of **ispare** array (**mspare** ≥ **nspare**)
ierr	if not zero, return with no operation

Output:

blabel	Title label for each block
goff10	power-of-ten offset for grid arrays for each block
ispare	spare words reserved for application for each block
locx	index of first value for each block in the **x** grid array
locy	index of first value for each block in the **y** grid array
locz	index of first value for each block in the **z** grid array
nblks	number of blocks read from file
nspare	number of spare words used in each block
nx	number of values in **x** array for each block
ny	number of values in **y** array for each block
nz	number of values in **z** array for each block
sc	character string containing dataset comment

st	character string containing dataset type label
tapp	number associated with application dataset type
x	grid array in *x* coordinate direction for each block
xlabel	Grid label for the *x* coordinate of each block
y	grid array in *y* coordinate direction for each block
ylabel	Grid label for the *y* coordinate of each block
z	grid array in *z* coordinate direction for each block
zlabel	Grid label for the *z* coordinate of each block
ierr	error status flag 0, No error -1, EOF marker encountered (Not really an error) 1, Incorrect dataset type 2, Insufficient array dimension -- array truncated

pfrnf3

Read a 3D, multiple-block, nonuniform-grid scalar dataset from a PFF file — This routine is used to read a NF3 (nonuniform, 3D, Floating) dataset from a PFF file. This operation is ***only allowed*** in READ mode. The format for this dataset type is described in detail in Appendix B.

```
subroutine pfrnf3 ( fid, mode, keep, mblks, mgrdx, mgrdy, mgrdz, mdim, tapp, st, sc, nblks,
                    nx, ny, nz, ispare, x, y, z, goff10, xlabel, ylabel, zlabel, blabel, locx, locy,
                    locz, locfa, lenfa, farray, foff10, ierr )
logical     keep
integer     fid, ierr, mblks, mdim, mgrdx, mgrdy, mgrdz, mode, nblks, tapp
integer     foff10(mblks) , goff10(mblks)
integer     ispare(5,mblks), lenfa(mblks), locfa(mblks)
integer     locx(mblks), locy(mblks), locz(mblks)
integer     nx(mblks), ny(mblks), nz(mblks)
real        x(mgrdx), y(mgrdy), z(mgrdz)
real        farray(mdim), x(mgrdx), y(mgrdy), z(mgrdz)
character   st*(*), sc*(*)
character   xlabel(mblks)*(*), ylabel(mblks)*(*), zlabel(mblks)*(*)
character   blabel(mblks)*(*)
```

Input:

fid	File ID of PFF file
keep	logical flag indicating whether or not to keep a non-zero value for floating data in the case of underflow.
mblks	dimensioned length of arrays over blocks in calling program
mdim	dimensioned length of floating array **farray** in calling program
mgrdx	dimensioned length of **x** grid array
mgrdy	dimensioned length of **y** grid array
mgrdz	dimensioned length of **z** grid array
mode	subroutine mode flag: if **mode** = 0, return floating array in **farray** otherwise, return pointer and length in **locfa** and **lenfa**
ierr	if not zero, return with no operation

Output:

blabel	Title label for each block
farray	linear array containing floating data (**mode** = 0 only)
foff10	power-of-ten offset for each block of floating array **farray**
goff10	power-of-ten offset for grid arrays for each block
ispare	spare words reserved for application for each block
lenfa	length of floating array for each block
locfa	pointer to floating array for each block: **mode** = 0, indices into linear array **farray** identifying the first data value in each block **mode** = 1, PFF file pointer to **<FARRAY>** for this block

locx	index of first value for each block in the **x** grid array
locy	index of first value for each block in the **y** grid array
locz	index of first value for each block in the **z** grid array
nblks	number of blocks read from file
nx	number of values in **x** array for each block
ny	number of values in **y** array for each block
nz	number of values in **z** array for each block
sc	character string containing dataset comment
st	character string containing dataset type label
tapp	number associated with application dataset type
x	grid array in *x* coordinate direction for each block
xlabel	Grid label for the *x* coordinate of each block
y	grid array in *y* coordinate direction for each block
ylabel	Grid label for the *y* coordinate of each block
z	grid array in *z* coordinate direction for each block
zlabel	Grid label for the *z* coordinate of each block
ierr	error status flag 0, No error -1, EOF marker encountered (Not really an error) 1, Incorrect dataset type 2, Insufficient array dimension -- array truncated

pfrnv3

Read a 3D, multiple-block, nonuniform-grid vector dataset from a PFF file — This routine is used to read a NV3 (nonuniform, 3D, Vector) dataset from a PFF file. This operation is ***only allowed*** in READ mode. The format for this dataset type is described in detail in Appendix B.

```
subroutine pfrnv3 ( fid, mode, keep, mblks, mgrdx, mgrdy, mgrdz, mdim, tapp, st, sc,
                    nblks, nx, ny, nz, ispare, x, y, z, goff10, xlabel, ylabel, zlabel, blabel,
                    locx, locy, locz, locfa, lenfa, vx, vy, vz, voff10, ierr )
logical     keep
integer     fid, ierr, mblks, mdim, mgrdx, mgrdy, mgrdz, mode, nblks, tapp
integer     voff10(mblks) , goff10(mblks)
integer     ispare(5,mblks), lenfa(mblks), locfa(mblks)
integer     locx(mblks), locy(mblks), locz(mblks)
integer     nx(mblks), ny(mblks), nz(mblks)
real        x(mgrdx), y(mgrdy), z(mgrdz)
real        vx(mdim), vy(mdim), vz(mdim), x(mgrdx), y(mgrdy), z(mgrdz)
character   st*(*), sc*(*)
character   xlabel(mblks)*(*), ylabel(mblks)*(*), zlabel(mblks)*(*)
character   blabel(mblks)*(*)
```

Input:

fid	File ID of PFF file
keep	logical flag indicating whether or not to keep a non-zero value for floating data in the case of underflow.
mblks	dimensioned length of arrays over blocks in calling program
mdim	dimensioned length of floating arrays **vx**, **vy**, and **vz** in calling program
mgrdx	dimensioned length of **x** grid array
mgrdy	dimensioned length of **y** grid array
mgrdz	dimensioned length of **z** grid array
mode	subroutine mode flag: if **mode** = 0, return vector component arrays in **vx**, **vy**, and **vz** otherwise, return pointer and length in **locfa** and **lenfa**
ierr	if not zero, return with no operation

Output:

blabel	Title label for each block
goff10	power-of-ten offset for grid arrays for each block
ispare	spare words reserved for application for each block
lenfa	length of floating array for each block
locfa	pointer to floating array for each block: **mode** = 0, indices identifying the first data value in each block for the linear arrays **vx**, **vy**, and **vz** **mode** = 1, PFF file pointer to **<FARRAY>** containing **vx** for this block
locx	index of first value for each block in the **x** grid array
locy	index of first value for each block in the **y** grid array

locz	index of first value for each block in the **z** grid array
nblks	number of blocks read from file
nx	number of values in **x** array for each block
ny	number of values in **y** array for each block
nz	number of values in **z** array for each block
sc	character string containing dataset comment
st	character string containing dataset type label
tapp	number associated with application dataset type
voff10	power-of-ten offset for each block of floating arrays **vx**, **vy**, and **vz**
vx	linear array containing *x* component of the vector (**mode** = 0 only)
vy	linear array containing *y* component of the vector (**mode** = 0 only)
vz	linear array containing *z* component of the vector (**mode** = 0 only)
x	grid array in *x* coordinate direction for each block
xlabel	Grid label for the *x* coordinate of each block
y	grid array in *y* coordinate direction for each block
ylabel	Grid label for the *y* coordinate of each block
z	grid array in *z* coordinate direction for each block
zlabel	Grid label for the *z* coordinate of each block
ierr	error status flag 0, No error -1, EOF marker encountered (Not really an error) 1, Incorrect dataset type 2, Insufficient array dimension -- array truncated

pfrngd

Read a dataset containing an *n*-dimensional vector of an *m*-dimensional, nonuniformly gridded space from a PFF file — This routine is used to read a NGD (nonuniform grid data) dataset from a PFF file. This operation is ***only allowed*** in READ mode. The format for this dataset type is described in detail in Appendix B.

```
subroutine pfrngd ( fid, keep, mdim, ndim, xdim, vdim, sprdim, tapp, st, sc, m, n, nx, nvp,
                    nspare, ispare, alabel, vlabel, x, goff10, v, voff10, ioff, ierr )
logical     keep
integer     fid, goff10, ierr, m, mdim, n, ndim, nspare, nvp
integer     sprdim, tapp, vdim, voff10, xdim
integer     ioff(1:*), ispare(1:sprdim), nx(1:mdim)
real        x(1:xdim), v(1:vdim,1:ndim)
character   st*(*), sc*(*)
character   alabel(1:mdim)*(*), vlabel(1:ndim)
```

Input:

fid	File ID of PFF file
keep	logical flag indicating whether or not to keep a non-zero value for floating data in the case of underflow.
mdim	dimensioned length of **nx** and **alabel** arrays in calling program (the maximum value of **m** that can be successfully read)
ndim	dimensioned length of **v** (2^{nd} dimension) and **vlabel** arrays in calling program (the maximum value of **n** that can be successfully read)
sprdim	dimensioned length of **ispare** array in calling program
vdim	dimensioned length of **v** (1^{st} dimension) array in calling program [the maximum value of the product (**nx**(1) × ... × **nx**(**m**)) that can be successfully read]
xdim	dimensioned length of **x** array in calling program [the maximum value of the sum (**nx**(1) + ... + **nx**(**m**)) that can be successfully read]
ierr	if not zero, return with no operation

Output:

alabel	labels of spatial axes
goff10	power-of-ten offset for grid arrays
ispare	spare words reserved for application
m	spatial dimensionality of dataset
n	vector dimensionality of dataset
nspare	number of spare words used by dataset
nvp	number of vector grid points (**nx**(1) × ... × **nx**(**m**))
nx	number of grid point values for each spatial axis
sc	character string containing dataset comment
st	character string containing dataset type label
tapp	number associated with application dataset type

v two-dimensional array containing vector data

vlabel labels of vector components

voff10 power-of-ten offset for vector data

x packed linear array containing grid point values for all spatial coordinates

ierr error status flag
- 0, No error
- -1, EOF marker encountered (Not really an error)
- 1, Incorrect dataset type
- 2, Insufficient array dimension -- array truncated
- 3, Error reading **ispare** array
- 4, size of **g** array read from dataset does not equal the sum (**nx**(1) + ... + **nx**(**m**))
- 5, size of **v** array read from dataset does not equal **nvp**

Needed workspace:

ioff integer array used to compute power-of-ten offsets for **x** and **v** arrays. It must be dimensioned at least as large as the larger of **mdim** and **ndim**

pfrvtx

Read a dataset containing a list of *m*-dimensional spatial vertices, each with *n* attributes, from a PFF file — This routine is used to read a VTX (vertex data) dataset from a PFF file. This operation is ***only allowed*** in READ mode. The format for this dataset type is described in detail in Appendix B.

```
subroutine pfrvtx ( fid, mode, keep, mdim, ndim, vdim, adim, m, n, nv, nvr, tapp, st, sc,
                    ispare, vlabel, alabel, vert, voff10, attr, aoff10, locv, loca, ierr )
logical     keep
integer     adim, fid, ierr, locv, m, mdim, mode, n, ndim, nv, nvr
integer     tapp, vdim, voff10
integer     ispare(1:5), aoff10(1:ndim), loca(1:ndim)
real        attr(1:*), vert(1:vdim)
character   st*(*), sc*(*)
character   alabel(1:mdim)*(*), vlabel(1:ndim)
```

Input:

adim if **mode** = 0, dimensioned length of 1st dimension of 2D **attr** array (maximum value of **nv**)
if **mode** = 1, dimensioned length of entire 1D **attr** array (maximum value of **n** × **nv**)

fid File ID of PFF file

keep logical flag indicating whether or not to keep a non-zero value for floating data in the case of underflow.

mdim dimensioned length of **vlabel** array in calling program (maximum value of **m**)

mode subroutine mode flag:
if **mode** = 0, **attr** is assumed to be two-dimensional: **attr**(1:**adim**,1:**ndim**)
if mode = 1, **attr** is assumed to be one-dimensional: **attr**(1:**adim**), into which the attribute data will be densely packed
otherwise, PFF file pointers to the vertex and attribute arrays are returned in **locv** and **lena**(1:**n**), respectively

ndim dimensioned length of **x**, **loca**, and **vlabel** arrays in calling program (maximum value of **n**)

vdim dimensioned length of **vert** array (maximum value of **m** × **nv**)

ierr if not zero, return with no operation

Output:

alabel labels of attribute data

aoff10 power-of-ten offset for each attribute's data (**attr**)

attr array containing attribute data (see **mode**)

ispare spare words reserved for application

loca PFF file pointers to the attribute **<FARRAY>** primitives (**mode** = 2 only)

locv PFF file pointer to the vertex **<FARRAY>** primitive (**mode** = 2 only)

m dimensionality of vertices in dataset

n	number of attributes in dataset
nv	number of vertices in dataset
nvr	number of vertices in the vertex list that are actually returned
sc	character string containing dataset comment
st	character string containing dataset type label
tapp	number associated with application dataset type
vert	**m**-dimensional vertex list array (packed) (not used for **mode** = 2)
vlabel	labels of vertex coordinates
voff10	power-of-ten offset for vertex data (**vert**)
ierr	error status flag 0, No error -1, EOF marker encountered (Not really an error) 1, Incorrect dataset type 2, Insufficient array dimension -- array truncated