Chapter 4

Vdatas (VS API)

4.1 Chapter Overview

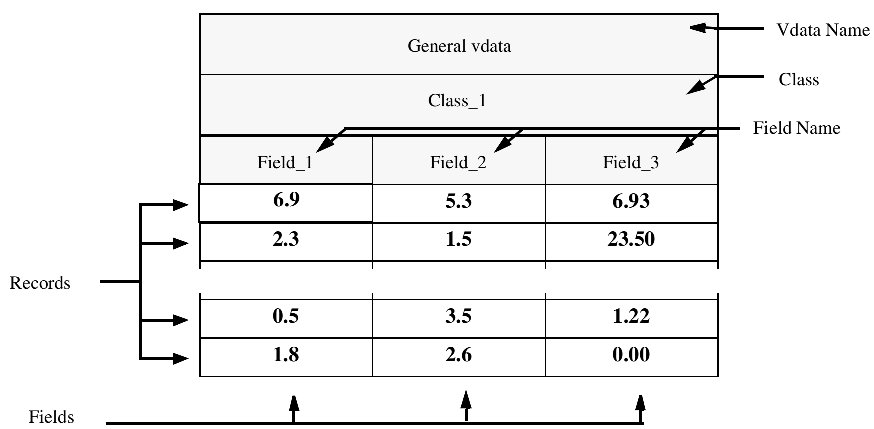
This chapter describes the vdata data model, the vdata API (also called the VS interface) and the vdata programming model. The last section of this chapter introduces command-line utilities that operate on vdata objects.

4.2 The Vdata Model

The vdata object is a collection of records whose values are stored in fixed-length fields. The HDF ***Vdata model*** provides a framework for storing customized tables, or ***vdatas***, in HDF files. The term “vdata” is an abbreviation of “vertex data” which alludes to the fact that, when the object was first implemented in HDF, it was designed specifically for the purpose of storing the vertex and edge information of polygon sets. The vdata design has since been generalized to apply to a broader variety of applications.

Vdatas are uniquely identified by a ***name***, a ***class*** and a series of individual ***field names*** . (See Figure 4a.)

FIGURE 4a **Vdata Table Structure**



A ***vdata name*** is a label typically assigned to describe the contents of a vdata. It often serves as a search key for files containing multiple vdatas. A ***vdata class*** further distinguishes a particular vdata by identifying the purpose or use of its data. Finally, ***vdata field names*** are labels assigned to each field.

4.2.1 Records and Fields

A vdata is a collection of ***records***, which itself is comprised of one or more fixed-length ***fields***. A vdata is like a table in which each row has the same structure and each column contains data of the same data type. (See Figure 4b.) Vdata records and fields are identified by an index. The record and field indexes are zero-based and separately incremented by one for each additional field and record in the vdata object.

Every field in a vdata is assigned a data type when the vdata is created. The data type of a field may be any basic HDF data type: character, 8-bit, 16-bit and 32-bit signed and unsigned integers, and 32-bit and 64-bit floating point numbers. The maximum length of a vdata record is 32,767 bytes.

The Vdata model allows multiple entries per field. The number of entries or ***components*** in a field is more commonly called the ***order*** of the field.

The organizational structure of a vdata is often determined by the data types of its data set or sets. For example, given a data set describing the location (“X, Y”) and temperature (“Temp”) of points in a plane, there are several ways to organize the data. (See Figure 4b.) If the “X”, “Y” and “Temp” values are of the same data type, they could be stored as three single-component fields, as two-component “X, Y” field and a Temp field or as a three-component “X, Y, Temp” field. Generally, the “X,Y” data is kept in a single field, but HDF places no restrictions on the organization of field data and there are no significant HDF performance issues involved in choosing one organizational regime over another.

FIGURE 4b **Three Different Vdata Structures for Data of the Same Number Type**

Simulation Data 1		
2D_Temperature_Grid		
X	Y	Temp
2.30	**1.50**	**23.50**
3.40	**5.70**	**8.03**
0.50	**3.50**	**1.22**
1.80	**2.60**	**0.00**

3 Single-Component Fields

Simulation Data 1	
2D_Temperature_Grid	
X, Y	Temp
2.30, 1.50	**23.50**
3.40, 5.70	**8.03**
0.50, 3.50	**1.22**
1.80, 2.60	**0.00**

1 Multi-Component Field
1 Single-Component Field

Simulation Data 1
2D_Temperature_Grid
X, Y, Temp
2.30, 1.50, 23.50
3.40, 5.70, 8.03
0.50, 3.50, 1.22
1.80, 2.60, 0.00

1 Multi-Component Field

4.2.2 Conventions

To facilitate the sharing of data in an HDF data file among different users and organizations, it is a good idea to establish conventions about vdata attributes such as vdata names, classes, field and record structures and field names. A typical HDF convention describes a configuration of vdata and vgroup objects designed to store polygonal data. These vdatas and vgroups collectively form a structure called a ***vset***. Chapter 5, titled *Vgroups (V API)*, contains additional information on vsets.

4.3 The Vdata API

The VS API consists of routines that are used to store and retrieve information about vdatas and their contents. The V interface, which is described in Chapter 5, titled *Vgroups (V API)*, can also be used to perform certain operations on vdatas.

4.3.1 Vdata Library Routines

Vdata routines begin with the prefixes "VS", "VF", "VSQ" and "VH". Routines in the V interface are prefaced by a capital "V". V routines perform most general vdata operations, VF routines query information about vdata fields and VSQ routines query information about specific vdatas. VH routines are high-level procedures designed to write to single-field vdatas.

Vdata routines let you define, organize and manipulate vdatas, and are categorized as follows:

- ***Access routines*** attach, or allow access, to vdatas. Data transfer can only occur after a vdata has been accessed. These routines also detach from, or properly terminate access to, vdatas when data transfer is completed.
- ***Read and write routines*** read and write the contents of a vdata.
- ***File inquiry routines*** provide information about how vdatas are stored in a file. They are useful for locating vdatas in a file.
- ***Vdata inquiry routines*** provide specific information about a given vdata, including the vdata's name, class, number of fields, number of records, tag and reference number pairs, interlace mode and size.
- ***Field inquiry routines*** provide specific information about the fields in a given vdata, including the field's size, name, order, type and number of fields in the vdata.

TABLE 4A **Vdata Interface Routines**

Category	Routine Name		Description
	C	Fortran-77	
Access	Vend	vfend	Closes the vdata interface.
	Vstart	vfstart	Initializes the vdata interface.
	VSattach	vsfatch	Establishes access to a specified vdata.
	VSdetach	vsfdtch	Terminates access to a specified vdata.
Read and Write	VHstoredata	vhfsd/vhfscd	Writes data to a simple, single-component vdata.
	VHstoredatam	vhfsdm/vhfscdm	Writes a vdata with one multi-component field.
	VSfdefine	vsffdef	Defines a new vdata field.
	VSsetclass	vsfscls	Assigns a class to a vdata.
	VSsetfields	vsfsfld	Specifies the vdata fields to be written to.
	VSsetinterlace	vsfsint	Sets the interlace mode for a vdata.
	VSsetname	vsfsnam	Assigns a name to a vdata.
	VSwrite	vsfwrit	Writes records to a vdata.
	VSread	vsfread	Reads from a vdata.
	VSseek	vsfseek	Seeks to a specified record in a vdata.
	VSsetattr	vsfsnat/vsfscat	Sets the attribute of a vdata field or vdata.
File Inquiry	VSfind	vsffnd	Searches for a given vdata name the opened HDF file.
	VSgetid	vsfgid	Returns the identifier of the next vdata in the file.
	VSlone	vsflone	Returns the vdatas that are not linked into vgroups.

Vdata Inquiry	VSfexist	vsfex	Tests for the existence of fields in the specified vdata.
	VSinquire	vsfinq	Returns information about the specified vdata.
	VSelts	vsfelts	Returns the number of records in the specified vdata.
	VSgetclass	vsfcls	Returns the class name of the specified vdata.
	VSgetfields	vsfgfld	Returns all field names within the specified vdata.
	VSgetinterlace	vsfgint	Retrieves the interlace mode of the specified vdata.
	VSgetname	vsfgnam	Retrieves the name of the specified vdata.
	VSsizeof	vsfsiz	Returns the field sizes of the specified vdata.
	VSQueryclass	None	Returns the class of the specified vdata.
	VSQueryfields	vsgfld	Returns the field names of the specified vdata.
	VSQueryname	vsfgname	Returns the name of the specified vdata.
	VSQueryref	None	Retrieves the reference number of the specified vdata.
	VSQuerytag	None	Retrieves the tag of the specified vdata.
	VSQuerycount	vsfelts	Returns the number of records in the specified vdata.
	VSQueryinterlace	vsfgint	Returns the interlace mode of the specified vdata.
	VSQueryvsize	vsfsiz	Retrieves the local size in bytes of the specified vdata record.
	VSfindex	vsffidx	Queries the index of a vdata field given the field name.
	VSsetattr	vsfsnat/vsfscat	Sets the attribute of a vdata field or vdata.
	VSnattrs	vsfnats	Queries the total number of vdata attributes.
	VSfnattrs	vsffnas	Queries the number of attributes of a vdata or vdata field.
	VSfindattr	vsffdat	Retrieves the index of an attribute given the attribute name.
	VSattrinfo	vsfainf	Queries information on a given attribute.
	VSgetattr	vsfgnat/vsfgcat	Queries the values of a given attribute.
	VSisattr	vsfisat	Determines if the given vdata is an attribute.
Field Inquiry	VFfieldesize	None	Retrieve the field size (as stored in a file) of a specified field.
	VFfieldisize	None	Retrieve the field size (as stored in memory) of a specified field.
	VFfieldname	None	Retrieves the name of the specified field in the given vdata.
	VFfieldorder	None	Retrieves the order of the specified field in the given vdata.
	VFfieldtype	None	Retrieves the data type for the specified field in the given vdata.
	VFnfields	None	Retrieves the total number of fields in the specified vdata.

4.3.2 Vdata Identifiers in the Vdata API

The VS interface identifies vdatas in several ways. Before a vdata is opened or *attached*, it is identified by its name, class and reference number. When a vdata is attached it is assigned a vdata identifier, or ***vdata id***, which is used by the vdata interface routines in accessing vdatas.

The name and class are assigned by the calling program and may be changed throughout the scope of the vdata object.

4.3.3 Programming Model for the Vdata Interface

The programming model for accessing vdatas is as follows:

1. Open a file.
2. Initialize the VS interface.
3. Open a vdata by obtaining a vdata id from a vdata reference number.
4. Perform the desired operations on the vdata.
5. Dispose of the vdata id.

6. Terminate access to the VS interface.
7. Close the file.

To access a vdata, the calling program must contain the following calls:

C:

```
file_id = Hopen(filename, file_access_mode, n_dds);
status = Vstart(file_id);
vdata_id = VSattach(file_id, vdata_ref, vdata_access_mode);

<Optional operations>

status = VSdetach(vdata_id);
status = Vend(file_id);
status = Hclose(file_id);
```

FORTRAN:

```
file_id = hopen(filename, file_access_mode, n_dds)
status = vfstart(file_id)
vdata_id = vsfatch(file_id, vdata_ref, vdata_access_mode)

<Optional operations>

status = vsfdtch(vdata_id)
status = vfend(file_id)
status = hclose(file_id)
```

4.3.4 Accessing Files and Vdatas: Hopen, Vstart and VSattach

The general routine **Hopen** opens an HDF file and obtains a file identifier. Details on the use of **Hopen** are provided in Chapter 2, titled *HDF Fundamentals*.

To perform operations on a vdata, the calling program must call **Vstart** for every file to be accessed and **VSattach** for every vdata to be accessed. **Vstart** initializes the internal vdata structures used by the VS API. **Vstart** has as its only argument the file identifier returned by **Hopen**.

VSattach accesses an individual vdata and must be called before all read and write operations. It has three arguments: `file_id`, `vdata_ref` and the `file_access_mode`. The `file_id` argument is the file identifier returned by **Hopen** and `vdata_ref` is the reference number that identifies to vdata to be accessed. Specifying a nonexistent reference number will return an error, specifying a value of `-1` will create a new vdata and specifying an existing reference number will result in the corresponding vdata being accessed.

The `access_mode` argument specifies the access mode ("`r`" for read-only access or "`w`" for write-only) required for subsequent operations on the specified vdata. Although several HDF user programs may simultaneously read from one vdata object, only one write access is allowed at a time. The "`r`" read-only access mode may only be used with existing vdatas, but the "`w`" access mode is valid with both new vdatas (`vdata_ref = -1`) and existing vdatas.

The parameters for **Vstart** and **VSattach** are further defined below. (See Table 4B.)

4.3.5 Terminating Access to Vdatas and Files: VSdetach, Vend and Hclose

VSdetach terminates access by updating pertinent information and freeing all memory associated with the vdata. Once access to the vdata is terminated, its identifier is invalid and any attempt to access it will result in an error condition.

Vend releases all internal data structures allocated by **Vstart**. It takes one argument the file identifier returned by **Hopen**. **Vend** must be called once per file and only after access to all vdata objects are terminated. Attempts to call VS interface routines after calling **Vend** will result in an error condition.

Successfully terminating access to a vdata requires one **VSdetach** call for each call to **VSattach** and one **Vend** call for each call to **Vstart**. All **VSdetach** calls must be made prior to making the first call to **Vend**.

Hclose terminates access to a file only if no vdata ids remain allocated. It is described in Chapter 2, titled *HDF Fundamentals*.

The parameters for **VSdetach** and **Vend** are further defined in the following table.

TABLE 4B

Vstart, VSattach, VSdetach and Vend Parameter List

Routine Name (Fortran-77)	Parameter	Data Type		Description
		C	Fortran-77	
Vstart (vfstart)	file_id	int32	integer	File identifier.
VSattach (vsfatch)	vdata_id	int32	integer	Vdata identifier.
	vdata_ref	int32	integer	Reference number of the vdata.
	file_access_mode	char *	character*1	Vdata access mode.
VSdetach (vsfdtch)	vdata_id	int32	integer	Vdata identifier.
Vend (vfend)	file_id	int32	integer	File identifier.

EXAMPLE 1.

Accessing a Vdata in an HDF File

These examples show how to write vdata information. It returns information about the specified vdata. The **VSinquire** routine is described in Section 4.9 on page 135.

Note that the “Dummy_HDF_File.hdf” HDF file represents a previously-created file.

C:

```
#include "hdf.h"

main( )
{

int32 file_id, vdata_id, status;
int32n_records, interlace, vdata_size;
charfields[30], vdata_name[30];

/* Open the HDF file. */
file_id = Hopen("Dummy_HDF_File.hdf", DFACC_RDONLY, 0);

/* Initialize the VS interface. */
status = Vstart(file_id);

/* Get information about the vdata. */
status = VSinquire(vdata_id, &n_records, &interlace, fields, \
                &vdata_size, vdata_name);

/* Terminate access to the vdata. */
```

```
status = VSdetach(vdata_id);

/* Terminate access to the VS interface. */
status = Vend(file_id);

/* Close the HDF file. */
status = Hclose(file_id);

}
```

FORTRAN:

```
      PROGRAM VDATA ACCESS

      integer*4 file_id, vdata_id, n_records, interlace, vdata_size
      integer status
      character fields*30, vdata_name*30
      integer hopen, vfatch, vsfinq, vfdtch, hclose

C     DFACC_RDONLY is defined in 'hdf.inc'.
      integer DFACC_RDONLY
      parameter (DFACC_RDONLY = 1)

C     Open an HDF file with read-only access.
      file_id = hopen('Dummy_HDF_File.hdf', DFACC_RDONLY, 0)

C     Initialize the Vset interface.
      status = vfstart(file_id)

C     Attach to the first Vdata.
      vdata_id = vfatch(file_id, -1, 'w')

C     Get information about the Vdata.
      status = vsfinq(vdata_id, n_records, interlace, fields,
     +                            vdata_size, vdata_name)

C     Terminate access to the Vdata.
      status = vfdtch(vdata_id)

C     Terminate access to the Vdata interface.
      status = vfend(file_id)

C     Close the HDF file.
      status = hclose(file_id)

      end
```

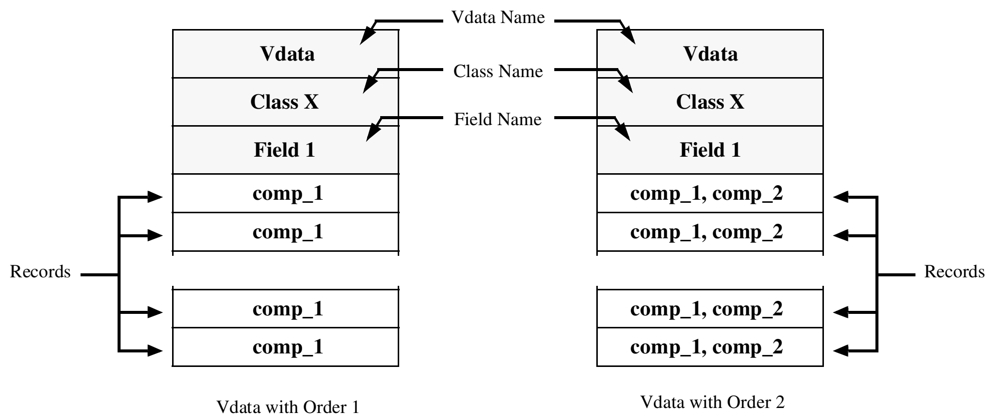
4.4 Creating and Writing to Single-Field Vdatas: VHstoredata and VHstoredatam

There are two methods of writing vdatas that contain one field per record. One requires the use of VS routines and the other involves the use of **VHstoredata** or **VHstoredatam**, two high-level routines that encapsulate several VS routines into one.

The high-level VH routines are useful for writing vdatas when it is only necessary to write one field per vdata and complete information about each vdata to be written is available. If you cannot provide full information about the vdata the VS routines described in the next section must be called.

Figure 4c on page 108 shows two examples of single-field vdatas. The fields can be single-component or multi-component fields, that is, they may contain one or more components of the same data type.

FIGURE 4c **Single- and Multi-Component Vdatas**



VHstoredata writes a vdata with one single-component field. **VHstoredatam** writes a vdata with one multi-component field. In both cases the following steps are involved:

1. Open the file.
2. Initialize the VS interface.
3. Create the vdata.
4. Terminate access to the VS interface.
5. Close the file.

These steps correspond to the following sequence of function calls:

C:

```
file_id = Hopen(filename, file_access_mode, n_dds);
status = Vstart(file_id);
vdata_ref = VHstoredata (file_id, fieldname, buf, n_records,
    data_type, vdata_name, vdata_class);

    OR

vdata_ref = VHstoredatam (file_id, fieldname, buf, n_records,
    data_type, vdata_name, vdata_class, order);
status = Vend(file_id);
status = Hclose(file_id);
```

FORTRAN:

```
file_id = hopen(filename, file_access_mode, n_dds)
status = vfstart(file_id)
vdata_ref = vhfsd (file_id, fieldname, buf, n_records,
    data_type, vdata_name, vdata_class)
status = vfend(file_id)
status = hclose(file_id)

    OR

vdata_ref = vhfsdm (file_id, fieldname, buf, n_records,
    data_type, vdata_name, vdata_class, order)
status = vfend(file_id)
status = hclose(file_id)
```

The first seven parameters of **VHstoredata** and **VHstoredatam** are the same. The `file_id` parameter is the file identifier returned by **Hopen**. The `fieldname` parameter is the name of the single field. The `buf`, `n_records` and `data_type` parameters contain the contents, number of records and data type of the vdata and the `vdata_name` and `vdata_class` parameters specify the name and class of the vdata. The `order` parameter of **VHstoredatam** specifies the order of the multi-component field. The maximum length of the vdata name is given by the `VSNAMELENMAX` definition in the "hlimits.h" header file. Both routines return the reference number of the newly-created vdata or `FAIL` (or `-1`) if the create operation was unsuccessful.

The parameters of **VHstoredata** and **VHstoredatam** are further described in the following table.

TABLE 4C

VHstoredataand VHstoredatam Parameter List

Routine Name (Fortran-77)	Parameter	Data Type		Description
		C	Fortran-77	
VHstoredata (vhfsd/vhfscd)	file_id	int32	integer	File identifier.
	fieldname	char *	character* (*)	String containing the name of the field.
	buf	uint8	integer (*)	Buffer containing the data to be stored.
	n_records	int32	integer	Number of records to create in the vdata.
	data_type	int32	integer	Number type of the stored data.
	vdata_name	char *	character* (*)	Name of the vdata.
	vdata_class	char *	character* (*)	Vdata class.
VHstoredatam (vhfsdm/ vhfscdm)	file_id	int32	integer	File identifier.
	fieldname	char *	character* (*)	String containing the name of the single field.
	buf	uint8 *	integer (*)	Buffer containing the data to be stored.
	n_records	int32	integer	Number of records to create in the vdata.
	data_type	int32	integer	Number type of the stored data.
	vdata_name	char *	character* (*)	Name of the vdata.
	vdata_class	char *	character* (*)	Class describing the type of vdata.
	order	int32	character* (*)	Number of components to store in the field.

EXAMPLE 2.

Creating Single-Field Vsets

In these examples two vdatas are created. The first vdata has five records with one field of order 1 and is created from a 5 x 1 array in memory. The second vdata has six records with one fields of order 4 and is created from a 6 x 4 array in memory.

C:

```
#include "hdf.h"

main( )
{

int32 file_id, vdata_id, vdata_ref1, vdata_ref2, status;
uint8 vset1_data[5] = {1, 2, 3, 4, 5};
uint8 vset2_data[6][4] = {{1, 2, 3, 4}, {2, 4, 6, 8},
                          {3, 6, 9, 12}, {4, 8, 12, 16},
                          {5, 10, 15, 20}, {6, 12, 18, 24}};

/* Create the HDF file. */
```

```
file_id = Hopen("Example2.hdf", DFACC_CREATE, 0);

/* Initialize the VS interface. */
status = Vstart(file_id);

/* Create the first vdata and populate it with data
* from the vset1_data array. */

vdata_ref1 = VHstoredata(file_id, "Vset1 data", (uint8 *)vset1_data, \
                5, DFNT_UINT8, "Second Vset", "5x1 array");

/*
* Create the second vdata and populate it with data
* from the vset2_data array.
*/

vdata_ref2 = VHstoredatam(file_id, "Vset2 data", (uint8 *)vset2_data, \
                6, DFNT_UINT8, "Third Vset", "6x4 array", 4);

/* Terminate access to the VS interface. */
status = Vend(file_id);

/* Close the HDF file. */
status = Hclose(file_id);

}
```

FORTRAN:

```
     PROGRAM VSET CREATE

      integer*4 file_id, vdata_ref1, vdata_ref2
      integer vset_data1(5), vset_data2(6, 4), i
      integer hopen, vhfsd, vhfsdm, hclose

C     DFACC_CREATE is defined in 'hdf.inc'. DFNT_INT32 is
C     defined in 'hntdefs.h'.
      integer DFACC_CREATE, DFNT_INT32
      parameter (DFACC_CREATE = 4, DFNT_INT32 = 24)

C     Generate the vset_data1 data.
      do 10 i = 1, 5
        vset_data1(i) = i
10    continue

C     Generate the vset_data2 data.
      do 20 i = 1, 6
        do 30 j = 1, 4
          vset_data2(i, j) = i + j
30      continue
20    continue

C     Create the HDF file.
      file_id = hopen('Example2.hdf', DFACC_CREATE, 0)

C     Initialize the VS interface.
      status = vfstart(file_id)

C     Create the first vdata and populate it with data
C     from the vset1_data array.
      vdata_ref1 = vhfsd(file_id, 'Vset1 data', vset_data1,
     +               5, DFNT_INT32, 'First Vset', '5x1 array')
```

```
C     Create the second vdata and populate it with data
C     from the vset2_data array.
      vdata_ref2 = vhfsdm(file_id, 'Vset2 data', vset_data2,
     +               6, DFNT_INT32, 'Second Vset', '6x4 array', 4)

C     Terminate access to the VS interface.
      status = vfend(file_id)

C     Close the HDF file.
      status = hclose(file_id)

      end
```

4.5 Creating and Writing to Multi-Field Vdatas

There are three steps involved in creating vdatas with more than one field, or ***general vdatas***: define the vdata, write the data to the vdata then write the vdata to the file. These steps are usually executed within a single program, although it is also possible to define an empty vdata in anticipation of writing data to it at a later time.

4.5.1 Creating Vdatas

Creating an empty vdata involves the following steps:

1. Open a file.
2. Initialize the VS interface.
3. Create the new vdata.
4. Assign a vdata name. (optional)
5. Assign a class. (optional)
6. Define the fields.
7. Set the interlace mode.
8. Dispose of the vdata identifier.
9. Terminate access to the VS interface.
10. Close the file.

Like the high-level VH interface, the VS interface does not retain default settings from one operation to the next or from one file to the next. Each time a vdata is created, its definitions must be explicitly reset.

To create a multi-component vdata, the calling program must contain the following:

C:

```
file_id = Hopen(filename, file_access_mode, n_dds);
status = Vstart(file_id);
vdata_id = VSattach(file_id, vdata_ref, vdata_access_mode);
status = VSsetname(vdata_id, vdata_name);
status = VSsetclass(vdata_id, vdata_class);
status = VSfdefine(vdata_id, fieldname, data_type, order);
status = VSsetfields(vdata_id, fieldname_list);
status = VSsetinterlace(vdata_id, interlace_type);
status = VSdetach(vdata_id);
status = Vend(file_id);
status = Hclose(file_id);
```

FORTRAN:

```
file_id = hopen(filename, file_access_mode, n_dds)
status = vfstart(file_id)
```

```
vdata_id = vsfatch(file_id, vdata_ref, vdata_access_mode)
status = vsfsnam(vdata_id, vdata_name)
status = vsfscls(vdata_id, vdata_class)
status = vsffdef(vdata_id, fieldname, data_type, order)
status = vsfsfld(vdata_id, fieldname_list)
status = vsfsint(vdata_id, interlace_type)
status = vsfdtch(vdata_id)
status = vfend(file_id)
status = hclose(file_id)
```

In the routines that follow, `vdata_id` is the vdata identifier returned by **VSattach.**

4.5.1.1 Creating New Vdatas with VSattach

When a new vdata is created, the value of the **VSattach** parameter `vdata_ref` must be `-1` and the value of the `access_mode` parameter must be `"w"`.

4.5.1.2 Assigning a Vdata Name and Class: VSsetname and VSsetclass

VSsetname assigns a name to a vdata object. If not explicitly named by a call to **VSsetname**, the name of the vdata is set by default to `NULL`. Names may be assigned and reassigned at any time after the vdata is created. The parameter `vdata_name` contains the name to be assigned to the vdata.

VSsetclass assigns a class to a vdata. If **VSsetclass** is not called, the vdata's class is set by default to `NULL`. As with vdata names, the class may be assigned and reassigned any time after the vdata is created. The parameter `vdata_class` contains the class name to be assigned to the vdata.

The parameters for **VSsetname** and **VSsetclass** are further defined below. (See Table 4E on page 113.)

4.5.1.3 Defining a Field Within Vdatas: VSfdefine

VSfdefine defines a field within a newly-created vdata. Each **VSfdefine** call assigns the name contained in the `fieldname` argument, the data type contained in the `data_type` argument and the order contained in the `order` argument to one new field. Fields will appear in the vdata in the order they are defined. Once data is written to a vdata, field definitions may not be modified or deleted.

The VS interface also provides certain ***predefined fields***. (See Table 4D.) A predefined field is assumed to have a specific name, data type and order, so there is no need to call **VSfdefine** to define a predefined field. Some applications may require the use of pre-defined fields in vdatas. (For instance, NCSA Polyview requires the coordinates of vertices to have the predefined field names "PX", "PY" and "PZ".)

The parameters of **VSfdefine** are further described below. (See Table 4E on page 113.)

TABLE 4D **Predefined Data Types and Field Names for Vdata Fields**

Data Type	Coordinate Point Field Names			Normal Component Field Names		
	x-coordinate	y-coordinate	z-coordinate	x-component	y-component	z-component
float	PX	PY	PZ	NX	NY	NZ
integer	IX	IY	IZ	None	None	None

4.5.1.4 Initializing the Fields for Write Access: VSsetfields

VSsetfields initializes write access to the fields in a vdata. It is used to specify the fields to be written and the order they are to be written in . The parameter `fieldname_list` is a comma-separated list of all the fields that will be accessed during the write operation. It may contain any fieldname previously defined by **VSfdefine**, but it may also contain names of predefined fields.

VSsetfields must be called prior to any write operation and may only be called once for each vdata. Any field not named in the `fieldname_list` parameter is removed from the vdata. The combined width of the fields must be less than 32,767 bytes. If an attempt is made to create a larger record than this, **VSsetfields** will fail and an error condition will result.

The parameters of **VSsetfields** are further described below. (See Table 4E on page 113.)

4.5.1.5 Specifying the Interlace Mode: VSsetinterlace

The VS interface supports two modes of interlacing: ***file interlacing*** and ***buffer interlacing***. File interlacing determines how data is stored in a vdata and buffer interlacing determines how data is stored in memory. The VS interface can write data from a buffer to a file in an interlaced or non-interlaced manner. It can also read data from a file in an interlaced or non-interlaced manner.

The **VSread** and **VSwrite** routines set the file interlace mode. The **VSwrite** routine will be discussed in Section 4.5.2.2 on page 115 and the **VSread** routine will be discussed in Section 4.6.3 on page 123.

VSsetinterlace sets the file interlacing mode for a vdata. Setting the parameter `interlace` to `FULL_INTERLACE` fills the vdata by record, whereas specifying `NO_INTERLACE` fills the vdata by field. (See Figure 4d.) For multi-component fields, all components are treated as a single field. As with file interlacing, the default vdata interlace mode is `FULL_INTERLACE` because it is more efficient to write complete records than it is to write fields if the file and memory interlace modes are the same, although both require the same amount of disk space.

FIGURE 4d

Interlaced and Non-Interlaced Vdata Contents

Vdata			
Mixed_Data_Type			
Temp	**Height**	**Speed**	**Ident**
1.11	**1**	**11.11**	**A**
2.22	**2**	**22.22**	**B**
3.33	**3**	**33.33**	**C**

File Interlacing: FULL_INTERLACE

Vdata			
Mixed_Data_Type			
Temp	**1.11**	**2.22**	**3.33**
Height	**1**	**2**	**3**
Speed	**11.11**	**22.22**	**33.33**
Ident	**A**	**B**	**C**

File Interlacing: NO_INTERLACE

VSsetinterlace can only be used for operations on new vdatas as the interlacing cannot be changed once the data has been written to a vdata object. Also, although records in a fully interlaced vdata can be written with one or more write operations, all records in a non-interlaced vdata must be written at the same time.

The parameters of **VSsetinterlace** are further described in the following table.

TABLE 4E

VSsetname, VSsetclass, VSfdefine, VSsetfields and VSsetinterlace Parameter List

Routine Name (Fortran-77)	Parameter	Data Type		Description
		C	Fortran-77	

VSsetname (vsfsnam)	vdata_id	int32	integer	Vdata identifier.
	vdata_name	char *	character* (*)	Vdata name.
VSsetclass (vsfscls)	vdata_id	int32	integer	Vdata identifier.
	vdata_class	char *	character* (*)	Vdata name.
VSfdefine (vsffdef)	vdata_id	int32	integer	Vdata identifier.
	fieldname	char *	character* (*)	Name of the field to be defined.
	data_type	int32	integer	Type of the field data.
	order	int32	integer	Order of the new field.
VSsetfields (vsfsfld)	vdata_id	int32	integer	Vdata identifier.
	fields	char *	character* (*)	Names of the vdata fields to be accessed.
VSsetinterlace (vsfsint)	vdata_id	int32	integer	Vdata identifier.
	interlace	int32	integer	Interlace mode.

4.5.2 Writing Data to Vdatas

Writing to a vdata requires the following steps:

1. Open a file.
2. Initialize the VS interface.
3. Access the vdata.
4. Seek to the target record.
5. Write the data.
6. Dispose of the vdata identifier.
7. Terminate access to the VS interface.
8. Close the file.

To write data to a vdata, the calling program must contain the following sequence of calls:

C:

```
file_id = Hopen(filename, file_access_mode, n_dds);
status = Vstart(file_id);
vdata_id = VSattach(file_id, vdata_ref, vdata_access_mode);
status = VSsetfields(vdata_id, *fields);
record_pos = VSseek(vdata_id, record_index);
num_of_recs = VSwrite(vdata_id, *databuf, n_records, interlace);
status = VSdetach(vdata_id);
status = Vend(file_id);
status = Hclose(file_id);
```

FORTRAN:

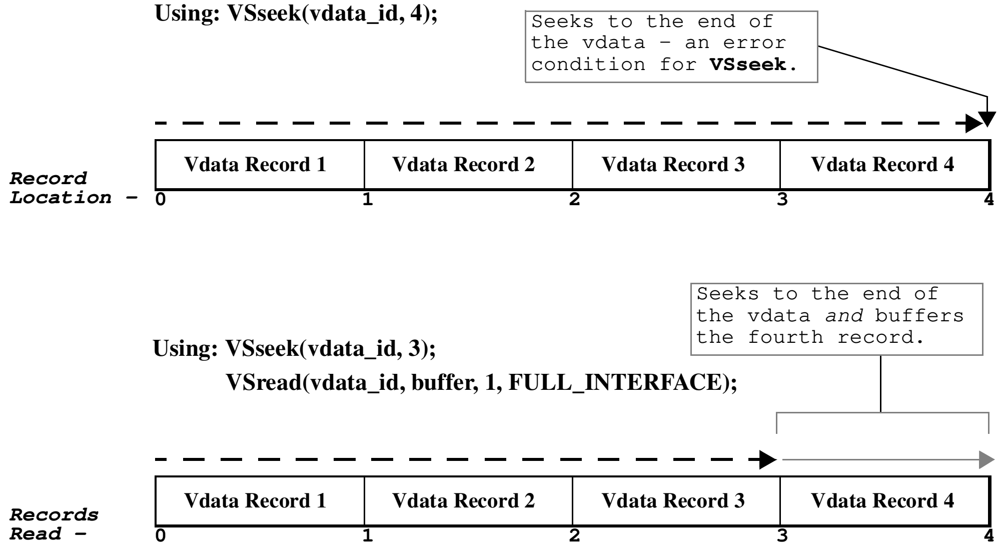
```
file_id = hopen(filename, file_access_mode, n_dds)
status = vfstart(file_id)
vdata_id = vsfatch(file_id, vdata_ref, vdata_access_mode)
status = vsfsfld(vdata_id, *fields);
record_pos = vsfseek(vdata_id, record_index);
num_of_recs = vsfwrit(vdata_id, *databuf, n_records, interlace)
status = vsfdtch(vdata_id)
status = vfend(file_id)
status = hclose(file_id)
```

4.5.2.1 Resetting the Current Position Within Vdatas: VSseek

VSseek provides a mechanism for random-access write operations within fully-interlaced vdatas. Random-access writes are not available for non-interlaced vdatas. The parameter `record_index` is the position of the record to be written. The position of the first record in a vdata is specified by `record_index=0`: any vdata operation will be performed on this vdata before **VSseek** is called.

One thing that should be kept in mind while using **VSseek** is that it has been designed for the purpose of *overwriting* data, not *appending* data. Figure 4e illustrates a situation where this aspect of **VSseek** can be misused with unexpected results and how **VSread** is called to correctly place the record pointer at the end of the vdata object for appending.

FIGURE 4e **Setting the Record Pointer at the End of a Vdata Object**



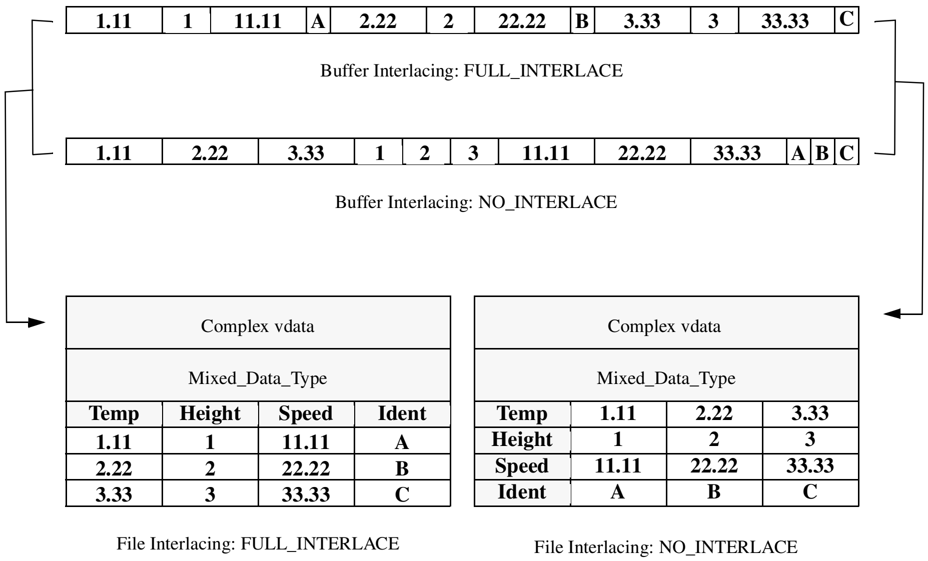
In this figure we have a vdata that consists of only four records. Using **VSseek** to seek to the end of the fourth record by setting the `record` parameter to `4` results in an error condition. To avoid this situation, we set the `record` parameter to `3` in order to set the current record pointer to the end of the third record. We then use **VSread** to read the contents of the fourth record in a buffer and move the current record pointer to the end of the fourth record. The contents of the buffer can then be discarded. This method can be generalized to a vdata of any number of records; seek to the last record with **VSseek**, then read the last record using **VSread**.

4.5.2.2 Writing to a Vdata: VSwrite

VSwrite writes buffered data to the vdata in the specified HDF file. The parameter `databuf` is a buffer containing records to store in the vdata. The `n_records` parameter is the number of records to be written.

The array `databuf` is assumed to be organized in memory as specified by `interlace`. Selecting `FULL_INTERLACE` indicates that the array in memory is organized by record, whereas `NO_INTERLACE` indicates that the array is organized by field. (See Figure 4f.) Notice that *file interlacing* is set by **VSsetinterlace** when the vdata is created, whereas the *buffer interlacing* is set by **VSwrite** when data is written to file. **VSwrite** will write interlaced or non-interlaced data to a file as an interlaced or non-interlaced vdata. However, multiple write operations can only be used on fully-interlaced vdatas.

FIGURE 4f **Writing Interlaced or Non-Interlaced Buffers into Interlaced or Non-interlaced Vdatas**



The data in `databuf` is assumed to contain the exact amount of data in the order needed to fill the fields defined in the last call to **VSsetfields**. Because **VSwrite** writes the contents of `databuf` contiguously to the vdata, any “padding” due to record alignment must be removed before attempting to write from `databuf` to the vdata. For more information on removing alignment padding see Section 4.5.2.3 on page 118.

It should be remembered that **VSwrite** writes whole records, not individual fields. If a modification to one field within a previously-written record is needed, the contents of the record must first be preserved by reading it using **VSread**, which will be described in Section 4.6.3 on page 123, updating the field then writing it using **VSwrite**.

To create an empty vdata either **VSwrite**, **VSsetname** and **VSsetclass** must be called before **VSdetach**. If **VSwrite** is not called the vdata created will be an empty vdata. If **VSsetfields** is called and **VSwrite**, **VSsetname** or **VSsetclass** is not called, the vdata will not be written.

TABLE 4F **VSseek and VSwrite Parameter List**

Routine Name (Fortran-77)	Parameter	Data Type		Description
		C	Fortran-77	
VSseek (**vsfseek**)	vdata_id	int32	integer	Vdata identifier.
	record	int32	integer	Record number to seek to.
VSwrite (**vsfwrit**)	vdata_id	int32	integer	Vdata identifier.
	databuf	char *	character* (*)	Buffer for records to be stored.
	n_records	int32	integer	Number of records to be stored.
	interlace	int32	integer	Interlace mode of the buffered data.

EXAMPLE 3. **Writing Data to a New Vdata**

In these examples, the VS interface is used to create a vdata containing ten rows, each comprised of one 16-bit integer field of order 3. The vdata is named "Example Vdata", the vdata class is "Example Vdata" and the field is named "Field Entries".

C:

```
#include "hdf.h"

#define FIELD_NAME "Field Entries"
#define NUMBER_OF_ROWS 10
#define ORDER 3

main( )
{

int32 file_id, vdata_id, status, num_of_elements;
int16 vdata_buf[NUMBER_OF_ROWS * ORDER], i;

/* Open the HDF file. */
file_id = Hopen("Example3.hdf", DFACC_CREATE, 0);

/* Initialize HDF for subsequent the vgroup/vdata access. */
status = Vstart(file_id);

/* Create a new vdata. */
vdata_id = VSattach(file_id, -1, "w");

/* Define the field data name, type and order. */
status = VSfdefine(vdata_id, FIELD_NAME, DFNT_INT16, ORDER);

/* Specify the field(s) that will be written to. */
status = VSsetfields(vdata_id, FIELD_NAME);

/* Generate the Vset data. */
for (i = 0; i < NUMBER_OF_ROWS * ORDER; i+=ORDER) {
   vdata_buf[i] = i;
   vdata_buf[i + 1] = i + 1;
   vdata_buf[i + 2] = i + 2;
}

/* Write the data to the Vset. */
num_of_elements = VSwrite(vdata_id, (char *)vdata_buf, NUMBER_OF_ROWS,
                              FULL_INTERLACE);

/* Set the name and class. */
status = VSsetname(vdata_id, "Example Vdata");
status = VSsetclass(vdata_id, "Example Vdata");

/* Terminate access to the vdata. */
status = VSdetach(vdata_id);

/* Terminate access to the Vset interface and close the file. */
status = Vend(file_id);
status = Hclose(file_id);

}
```

FORTRAN:

```
      PROGRAM WRITE VDATA

       integer file_id, vdata_id
       integer status
```

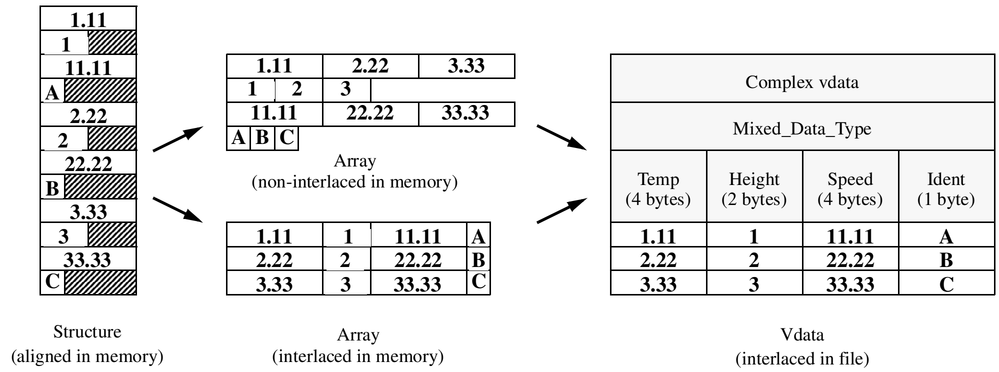
```
      integer vdata_buf(30), i, number_of_rows, num_of_elements
      integer hopen, vsfatch, vsffdef, vsfsfld
      integer vsfwrit, vsfsnam, vsfscls, vsfdtch
      integer hclose, vfstart, vfend

C     DFACC_CREATE, DFNT_INT16 and FULL_INTERLACE are defined
C     in hdf.inc.
      integer DFACC_CREATE, DFNT_INT16, FULL_INTERLACE
      parameter (DFACC_CREATE = 4,
     +           DFNT_INT16 = 22,
     +           FULL_INTERLACE = 0)

      character field_name *13
      parameter (field_name = 'Field Entries')
      parameter (number_of_rows = 10)

C     Create an HDF file with full access.
      file_id = hopen('Example3.hdf', DFACC_CREATE, 0)

C     Initialize HDF for subsequent vgroup/vdata access.
      status = vfstart(file_id)

C     Create a new vdata.
      vdata_id = vsfatch(file_id, -1, 'w')

C     Define the field data name, type and order.
      status = vsffdef(vdata_id, field_name, DFNT_INT16, 3)

C     Specify the field that will be written to.
      status = vsfsfld(vdata_id, field_name)

C     Generate the Vset data.
      do 10 i = 1, number_of_rows * 3, 3
        vdata_buf(i) = i+1
        vdata_buf(i + 1) = i+2
        vdata_buf(i + 2) = i+3
10    continue

C     Set the name and class.
      status = vsfsnam(vdata_id, 'Example Vdata')
      status = vsfscls(vdata_id, 'Example Vdata')

C     Write the data to the Vset.
      num_of_elements = vsfwrit(vdata_id, vdata_buf, number_of_rows *
3,
     +                FULL_INTERLACE)

C     Terminate access to the vdata.
      status = vsfdtch(vdata_id)

C     Terminate access to the Vset interface and close the file.
      status = vfend(file_id)
      status = hclose(file_id)

      end
```

4.5.2.3 Filling Records of Mixed Field Data Types

Storing fields of mixed data types is an efficient use of disk space and is useful in applications that use structures. While data structures in memory containing fields of variable lengths often contain padding or alignment bytes, data stored in a vdata does not contain padding. This is true for both fully-interlaced and non-interlaced data. Because of this, when variable-length field types are used it is not generally possible to write data directly from a structure in memory into a vdata in a file with a **VSwrite** call. Instead, the data from the structure must first be written to a temporary buffer array, removing all alignment bytes. This packed array is then written to a vdata using **VSwrite**. (See Figure 4b.)

FIGURE 4g

Removing Alignment Bytes When Writing Data From a C Structure to a Vdata



EXAMPLE 4.

Writing Packed Vdata Data to an HDF File

The following examples write packed data into a vdata. Note that while the **memcpy** routine is called for each field, the **bcopy** routine may be used instead.

Fortran-77 does not support record structures so there is no standard way to do packing as in C. The Fortran-77 example will work in many platforms but it is not guaranteed to be 100% portable.

C:

```
#include <stdio.h>
#include "hdf.h"

#define NRECORDS 20

#define FIELD_1 "Temp"
#define FIELD_2 "Height"
#define FIELD_3 "Speed"
#define FIELD_4 "Ident"
#define FIELD_NAMES  "Temp,Height,Speed,Ident"

main( )
{

struct {
   float32  temp;
   int16    height;
   float32  speed;
   char     ident;
} source[NRECORDS];

int32file_id, vdata_id;
uint8 *pntr;
uchar8 *databuf;
int i;
int32 status;
```

```
int32 msize = 0;

/* Create the HDF file. */
file_id = Hopen("Example4.hdf", DFACC_CREATE, 0);

/* Initialize the Vset interface. */
status = Vstart(file_id);

/* Create a new vdata. */
vdata_id = VSattach(file_id, -1, "w");

/* Define the field to write. */
status = VSfdefine(vdata_id, FIELD_1, DFNT_FLOAT32, 1);
status = VSfdefine(vdata_id, FIELD_2, DFNT_INT16, 1);
status = VSfdefine(vdata_id, FIELD_3, DFNT_FLOAT32, 1);
status = VSfdefine(vdata_id, FIELD_4, DFNT_CHAR8, 1);

/* Set the vdata name. */
status = VSsetname(vdata_id, "Example Vset Name");

/* Set the vdata class. */
status = VSsetclass(vdata_id, "Example Vset Class");

/* Set the field names. */
status = VSsetfields(vdata_id, FIELD_NAMES);

databuf = (uint8 *) malloc(((2 * sizeof(float32))
           + sizeof(int16) + sizeof(char)) * NRECORDS);

pntr = databuf;

for (i = 0; i < NRECORDS; i++) {
   source[i].temp = 1.11 * (i+1);
   memcpy(pntr, &source[i].temp, sizeof(float32));
   pntr += sizeof(float32);

   source[i].height = i+1;
   memcpy(pntr, &source[i].height, sizeof(int16));
   pntr += sizeof(int16);

   source[i].speed = 1.11 * (i+1);
   memcpy(pntr, &source[i].speed, sizeof(float32));
   pntr += sizeof(float32);

   source[i].ident = 'A'+i;
   memcpy(pntr, &source[i].ident, sizeof(char));
   pntr += sizeof(char);
}

/* Write the data to the Vset object. */
status = VSwrite(vdata_id, databuf, NRECORDS, FULL_INTERLACE);

/*
* Terminate access to the vdata, the VS interface
* and the HDF file.
*/
status = VSdetach(vdata_id);
status = Vend(file_id);
status = Hclose(file_id);

}
```

FORTRAN:

```
      PROGRAM DATA PACK

       integer hopen, vsfatch, vsffdef, vsfsnam
       integer vsfscls, vsfsfld, vsfwrit, vsfdtch
       integer hclose, vfstart

C      Parameter definitions
       integer*4 DFACC_CREATE, DFNT_FLOAT32, DFNT_INT16, DFNT_CHAR8
       integer*4 FULL_INTERLACE
       parameter (DFACC_CREATE = 4,
      +           DFNT_FLOAT32 = 5,
      +           DFNT_INT16 = 22,
      +           DFNT_CHAR8 = 4,
      +           FULL_INTERLACE = 0)

C      Example parameters.
       integer NRECORDS
       parameter (NRECORDS = 20)

C      Vdata fields.
       real temp
       integer*2 height
       real speed

C      Vdata field equivalence names for packing.
       character ctemp*4
       character cheight*2
       character cspeed*4
       equivalence (temp, ctemp), (height, cheight), (speed, cspeed)

C      Packing buffer with size (4+2+4+1)*NRECORDS = 220.
       character buffer*220

C      More local variables.
       integer i, bufptr, status
       integer*4 file_id, vdata_id
       character*20 idents /"ABCDEFGHIJKLMNOPQRST"/

C      Create the file.
       file_id = hopen('Example4.hdf', DFACC_CREATE, 0)

C      Initialize the interface.
       status = vfstart(file_id)

C      Create a new vdata.
       vdata_id = vsfatch(file_id, -1, 'w')

C      Define the fields to write.
       status = vsffdef(vdata_id, 'Temp', DFNT_FLOAT32, 1)
       status = vsffdef(vdata_id, 'Height', DFNT_INT16, 1)
       status = vsffdef(vdata_id, 'Speed', DFNT_FLOAT32, 1)
       status = vsffdef(vdata_id, 'Ident', DFNT_CHAR8, 1)

C      Set the vdata name.
       status = vsfsnam(vdata_id, 'Example Vset Name')

C      Set the vdata class.
       status = vsfscls(vdata_id, 'Example Vset Class')

C      Set the field names.
       status = vsfsfld(vdata_id, 'Temp,Height,Speed,Ident')
```

```
C     Pack NRECORDS of data into buffer.
      bufptr = 1

      do 10 i = 1, NRECORDS
C       Pack temp data.
        temp = 1.1 * i
        buffer(bufptr:bufptr+3) = ctemp
        bufptr = bufptr + 4

C       Pack height data.
        height = i
        buffer(bufptr:bufptr+1) = cheight
        bufptr = bufptr + 2

C       Pack speed data.
        speed = 11.1 * i
        buffer(bufptr:bufptr+3) = cspeed
        bufptr = bufptr + 4

C       Pack ident data.
        buffer(bufptr:bufptr) = idents(i:i)
        bufptr = bufptr + 1
10    continue

      status = vsfwrit(vdata_id, buffer, NRECORDS, FULL_INTERLACE)

C     Terminate access to the vdata object, the interface
C     and the file.
      status = vsfdtch(vdata_id)
      status = vfend(file_id)
      status = hclose(file_id)

      end
```

4.6 Reading from Vdatas

Reading from vdatas is more complicated than writing to vdatas, as it usually involves searching for a particular vdata, as well as searching within the vdata, before actually reading data. The process of reading from vdatas can be summarized in three steps:

1. Identify the appropriate vdata in the file.
2. Obtain information about the vdata.
3. Read in the desired data.

The last step will be covered first as the vdata of interest is usually known, therefore the first two steps are not needed.

These three basic steps can be expanded into the following:

1. Open a file.
2. Initialize the VS interface.
3. Access the vdata.
4. Optionally seek to the records to access.
5. Initialize the fields to access.
6. Read the data.
7. Detach from the vdata.

8. Terminate access to the VS interface.
9. Close the file.

The calling program must contain the following sequence of calls:

C:

```
file_id = Hopen(filename, file_access_mode, n_dds);
status = Vstart(file_id);
vdata_id = VSattach(file_id, vdata_ref, vdata_access_mode);
record_pos = VSseek(vdata_id, record_index);
status = VSsetfields(vdata_id, fieldname_list);
status = VSread(vdata_id, *databuf, n_records, interlace);
status = VSdetach(vdata_id);
status = Vend(file_id);
status = Hclose(file_id);
```

FORTRAN:

```
file_id = hopen(filename, file_access_mode, n_dds)
status = vfstart(file_id)
vdata_id = vsfatch(file_id, vdata_ref, vdata_access_mode)
record_pos = vsfseek(vdata_id, record_index)
status = vsfsfld(vdata_id, fieldname_list)
status = vsfread(vdata_id, *databuf, n_records, interlace)
status = vsfdtch(vdata_id)
status = vfend(file_id)
status = hclose(file_id)
```

4.6.1 Selecting the Record to Begin Reading from: VSseek

VSseek is described in Section 4.5.2.1 on page 114.

4.6.2 Selecting the Fields to be Read: VSsetfields

VSsetfields establishes access to the fields targeted for the next read operation. The argument `fieldname_list` is a comma-separated string of the target field names with no white space

The order the field names occur in `fieldname_list` is the order in which the fields will be read. For example, assume that a vdata contains fields named "A, B, C, D, E, F" and they are stored in alphabetical order. The following declarations demonstrate how to use `fieldname_list` to read a single field, a collection of random fields and a series of fields in reverse order;

- Single field: `fieldname_list = "B".`
- Collection of fields: `fieldname_list = "A,E".`
- Reverse order: `fieldname_list = "F,E,D,C,B,A".`

4.6.3 Reading from the Current Vdata: VSread

VSread sequentially retrieves data from the records in a vdata. The parameter `databuf` is the buffer to store the retrieved data, `n_records` is the number of records to retrieve and `interlace` specifies the interlace mode (`FULL_INTERLACE` or `NO_INTERLACE`) to be used in the contents of `databuf`. **VSread** returns the total number of records read if successful and `FAIL` (or `-1`) otherwise.

Prior to the first **VSread** call, **VSsetfields** must be called.

If a **VSread** call is successful, the data returned in `databuf` is formatted according to the interlace mode specified by `interlace` and the data fields appear in the order specified in the last call to **VSsetfields** for that vdata. To retrieve an arbitrary record from a vdata, use **VSseek** to specify the record position before calling **VSread**.

TABLE 4G

VSseek, VSsetfields and VSread Parameter List

Routine Name (Fortran-77)	Parameter	Data Type		Description
		C	Fortran-77	
VSseek (vsfseek)	vdata_id	int32	integer	Vdata identifier.
	record	int32	integer	Record number to seek to.
VSsetfields (vsfsfld)	vdata_id	int32	integer	Vdata identifier.
	fields	char *	character* (*)	Names of vdata fields to be accessed.
VSread (vsfread)	vdata_id	int32	integer	Vdata identifier.
	databuf	char *	character* (*)	Buffer for the retrieved data.
	n_records	int32	integer	Number of records to be retrieved.
	interlace	int32	integer	Interlace mode of the buffered data.

VSsetfields and **VSread** may be called several times to read from the same vdata. However, note that **VSread** operations are sequential. Thus, in the following code segment, the first call to **VSread** returns ten “A” data values from the first ten elements in the vdata, while the second call to **VSread** returns ten “B” data values from the second ten elements (elements 10 to 19) in the vdata.

```
VSsetfields(vs, "A");
VSread(vs, bufA, 10, interlace);

VSsetfields(vs, "B");
VSread(vs, bufB, 10, interlace);
```

To read the first ten “B” data values, the access routine **VSseek** must be called to explicitly position the read pointer back to the position of the first record. The following code segment reads the first ten “A” and “B” values into two separate float arrays `bufA` and `bufB`.

```
VSsetfields(vs, "A");
VSread(vs, bufA, 10, interlace);

VSseek(vs, 0); /* seeks to first element */
VSsetfields(vs, "B");
VSread(vs, bufB, 10, interlace);
```

EXAMPLE 5.

Reading Data from a Vdata

In the following examples, the VS interface is used to read the vdata created in the Example 4. **VSgetid** is used to demonstrate a method of obtaining the reference id number of the first vdata in the HDF file.

C:

```
#include “hdf.h”
#define NRECORDS 20

main( )
{

char vdata_name[MAX_NC_NAME], vdata_class[MAX_NC_NAME], fields[60];
int32 file_id, vdata_id, status, num_of_elements;
int32 n_records, interlace, vdata_size, vdata_ref;
uint8 databuf[((2 * sizeof(float32)) + sizeof(char) \
```

```
            + sizeof(int16)) * NRECORDS];

/* Open the HDF file. */
file_id = Hopen("Example4.hdf", DFACC_RDONLY, 0);

/* Initialize the Vset interface. */
status = Vstart(file_id);

/*
* Get the reference number for the first vdata in
* the file.
*/
vdata_ref = -1;
vdata_ref = VSgetid(file_id, vdata_ref);

/* Attach to the first vdata in read mode. */
vdata_id = VSattach(file_id, vdata_ref, "r");

/* Get the list of field names. */
status = VSinquire(vdata_id, &n_records, &interlace, fields,
                        &vdata_size, vdata_name);

/* Get the class. */
status = VSgetclass(vdata_id, vdata_class);

/* Determine the fields that will be read. */
status = VSsetfields(vdata_id, fields);

/* Print the vdata information. */
printf("Current vdata name: %s \nCurrent vdata class: %s\n", \
                     vdata_name, vdata_class);

/* Read the data. */
num_of_elements = VSread(vdata_id, databuf, n_records,
                FULL_INTERLACE);

/* Detach from the vdata, close the vset interface and the file. */
status = VSdetach(vdata_id);
status = Vend(file_id);
status = Hclose(file_id);

}
```

FORTRAN:

```
     PROGRAM DATA READ

      character vdata_name*30, vdata_class*30, fields*60
      character databuf*40
      integer file_id, vdata_ref, vdata_id
      integer*4 n_records, interlace, vdata_size
      integer status
      integer hopen, vsfgid, vsfatch, vsfinq, vsfgcls
      integer vsfsfld, vsfread, vsfdtch, hclose, vfstart, vfend

C     DFACC_RDONLY and FULL_INTERLACE are defined in hdf.inc.
      integer*4 DFACC_RDONLY, FULL_INTERLACE
      parameter (DFACC_RDONLY = 1,
     +           FULL_INTERLACE = 0)

C     Open an HDF file with read-only access.
      file_id = hopen('Example4.hdf', DFACC_RDONLY, 0)
```

```
C     Initialize the Vset interface.
      status = vfstart(file_id)

C     Get the reference number for the first Vdata in
C     the file.
      vdata_ref = -1
      vdata_ref = vsfgid(file_id, vdata_ref)

C     Attach to the first Vdata in read mode.
      vdata_id = vsfatch(file_id, vdata_ref, 'r')

C     Get the list of field names.
      status = vsfinq(vdata_id, n_records, interlace, fields,
     +               vdata_size, vdata_name)

C     Get the class.
      status = vsfgcls(vdata_id, vdata_class)

C     Determine the fields that will be read.
      status = vsfsfld(vdata_id, fields)

C     Print the Vdata information.
      print *, 'Current Vdata name: ', vdata_name
      print *, 'Current Vdata class: ', vdata_class

C     Read the data.
      status = vsfread(vdata_id, databuf, n_records, FULL_INTERLACE)

C     Detach from the Vdata, close the Vset interface and the file.
      status = vsfdtch(vdata_id)
      status = vfend(file_id)
      status = hclose(file_id)

      end
```

4.7 Searching for Vdatas in a File

There are several HDF routines that perform searches for a specific vdata in a file. In this section we introduce these four routines and in the following section methods for obtaining information about the members of a given vdata is described.

4.7.1 Finding All Vdatas that are Not Members of Vgroups: VSlone

A ***lone vdata*** is one that is not a member of a vgroup. The function **VSlone** returns the total number of lone vdatas. The parameter `ref_array` is an array allocated to hold the reference numbers of all lone vdatas in a file and the `maxsize` argument specifies the maximum size of `ref_array`. At most `maxsize` reference numbers will be returned in `ref_array`.

The space that should be allocated for `ref_array` is dependent upon on how many lone vdatas you expect to find. A size of 32,767 integers is adequate to handle any case. To use dynamic memory instead of allocating such a large array, first call **VSlone** with `maxsize` set to a small value like `0` or `1`, then use the returned value to allocate memory for `ref_array` to be passed to a subsequent call to **VSlone**.

The syntax of the **VSlone** function is as follows:

```
num_of_lone_vdatas = VSlone(file_id, ref_array, maxsize);
```

4.7.2 Searching for the Reference Number of a Vdata: VSgetid

VSgetid sequentially searches for vdata reference numbers in an HDF file. It returns the reference number for the vdata immediately following the vdata with the reference number `vdata_ref`. To initiate a search, **VSgetid** may be called using `vdata_ref = -1`. Doing so returns the reference number of the first vdata in the file. Searching past the last vdata in a file will return FAIL (or `-1`).

The syntax of the **VSgetid** function is as follows:

```
ref_num = VSgetid(file_id, vdata_ref);
```

4.7.3 Determining a Reference Number from a Vdata Name: VSfind

VSfind checks an HDF file for a vdata with the specified name and returns its reference number if it is found or `FAIL` (or `-1`) if not. The parameter `vdata_name` is the search key. Although there may be several identically named vdatas in a file, **VSfind** will only return the reference number of the first vdata in the file with the specified name.

The syntax of the **VSfind** function is as follows:

```
ref_num = VSfind(file_id, vdata_name);
```

4.7.4 Searching for a Vdata by Field Name: VSfexist

VSfexist queries a vdata for a set of specified field names and is often useful for locating vdata objects containing particular field names. Its return value is `1` if successful. The `fields` parameter is a string of comma-separated field names containing no white space - for example, "PX,PY,PZ".

The syntax of the **VSfexist** function is as follows:

```
status = VSfexist(vdata_id, fields);
```

To review, **VSfind** searches for a vdata with a specified name and returns its reference number if found. Given the reference number of the target vdata, **VSfexist** determines whether certain field names exist in it.

TABLE 4H **VSlone, VSgetid, VSfind and VSfexist Parameter List**

Routine Name (Fortran-77)	Parameter	Data Type		Description
		C	Fortran-77	
VSlone (vsflone)	file_id	int32	integer	File identifier.
	ref_array	int32 []	integer (*)	Buffer for a list of vdata reference numbers.
	maxsize	int32	integer	Maximum number of reference numbers to be buff- ered.
VSgetid (vsfgid)	file_id	int32	integer	File identifier.
	vdata_ref	int32	integer	Reference number of the vdata preceding the target vdata.
VSfind (vsffnd)	file_id	int32	integer	File identifier.
	vdata_name	char *	character* (*)	Name of the vdata to find.
VSfexist (vsfex)	vdata_id	int32	integer	Vdata identifier.
	fields	char *	character* (*)	Names of the fields to be queried.

EXAMPLE 6. **Locating a Vdata from a Specified Field Name**

The following examples search all vdatas in an HDF file for the first vdata containing the field names "Temp" and "Ident".

C:

```
#include "hdf.h"

#define FILE_NAME "Example4.hdf"

main( )
{

int32 file_id, vdata_ref, vdata_id, status;
int8 found_fields;

/* Open the HDF file. */
file_id = Hopen(FILE_NAME, DFACC_RDONLY, 0);

/* Initialize the vset interface. */
status = Vstart(file_id);

vdata_ref = -1;
found_fields = 0;

/* Attach to each vdata and search for the fields. */
while ((vdata_ref = VSgetid(file_id, vdata_ref)) != -1) {
   vdata_id = VSattach(file_id, vdata_ref, "r");
   if ((status = VSfexist(vdata_id, "Temp,Speed,Height,Ident"))
            != FAIL) {
      found_fields = 1;
      break;
   } else status = VSdetach(vdata_id);
}

if (!found_fields)
    printf("Fields TEMP and IDENT were not found.\n");
else printf("Fields TEMP and IDENT found in vdata id %d\n",
        vdata_ref);

/* Detach from the vdata, close the Vset interface and the file. */
status = VSdetach(vdata_id);
status = Vend(file_id);
status = Hclose(file_id);

}
```

FORTRAN:

```
     PROGRAM SCAN VDATAS

      integer file_id, vdata_ref
      logical found_fields
      integer vdata_id, status
      integer hopen, vsfgid, vsfatch, vsfex, vsfdtch, hclose
      integer vfstart, vfend

C     DFACC_RDONLY is defined in hdf.inc.
      integer DFACC_RDONLY
      parameter (DFACC_RDONLY = 1)

C     Open an HDF file with full access.
      file_id = hopen('Example4.hdf', DFACC_RDONLY, 0)
```

```
C     Initialize the Vset interface.
      status = vfstart(file_id)

      vdata_ref = -1
      found_fields = .FALSE.

C     Attach to each vdata and search for the fields.
30    vdata_ref = vsfgid(file_id, vdata_ref)
      if (vdata_ref .ne. -1) then
        vdata_id = vsfatch(file_id, vdata_ref, 'r')
        status = vsfex(vdata_id, 'Temp,Ident')
        if (status .eq. 1) then
          found_fields = .TRUE.
          go to 20
        else
          status = vsfdtch(vdata_id)
          go to 30
        end if
      end if

20    if (found_fields .eq. .FALSE.) then
        print *, 'Fields TEMP and IDENT were not found.'
      else
        print *, 'Fields TEMP and IDENT found in vdata id ',
     +           vdata_id
      end if

C     Detach from the vdata, close the Vset interface and the file.
      status = vsfdtch(vdata_id)
      status = vfend(file_id)
      status = hclose(file_id)

      end
```

4.8 Vdata Attributes

Version 4.1r1 of HDF includes the capability of assigning attributes to a vdata and/or a vdata field. The concept of attributes is fully explained in Chapter 3, titled *Scientific Data Sets (SD API)*. To review briefly: an attribute has a name, a data type, a number of attribute values and the attribute values themselves. All attribute values must be of the same data type - for example, an integer cannot be added to an attribute value consisting of ten characters, or a character value cannot be included in an attribute value consisting of 2 32-bit integers.

Any number of attributes can be assigned to either a vdata or any single field in a vdata. However, each attribute name should be unique within its scope. In other words, a field attribute name must be unique among all attributes of that field, and a vdata attribute name must be unique among all attributes of the vdata it's contained in.

4.8.1 Querying the Index of a Vdata Field Given the Field Name: VSfindex

VSfindex returns the index of specified field given the field name. It returns `SUCCEED` if successful, and `FAIL` otherwise.

TABLE 4I

VSfindex Parameter List

Routine Name (Fortran-77)	Parameter	Data Type		Description
		C	Fortran-77	
VSfindex (vsffidx)	vdata_id	int32	integer	Vdata identifier.
	field_name	char *	character* (*)	Name of the target vdata field.
	field_index	int32 *	integer	Index of the target vdata field.

4.8.2 Setting the Attribute of a Vdata Field or Vdata: VSsetattr

VSsetattr attaches an attribute to a vdata field or a vdata. If the attribute already exists, the new values will replace the current ones, provided the data type and order have not been changed. If either the data type or the order have been changed, **VSsetattr** will exit on an error condition.

The *field_index* parameter is zero-based, and represents the ordinal location of the field within the vdata - for example, a *field_index* value of `4` would refer to the fifth field in the vdata. If *field_index* is set to `_HDF_VDATA`, the attribute of the specified vdata will be set to the value pointed to by *values*. When using either Fortran-77 version of this routine, specify a *field_index* value of `-1` to attach an attribute to the specified vdata.

VSsetattr returns `SUCCEED` if successful, `FAIL` otherwise.

TABLE 4J

VSsetattr Parameter List

Routine Name (Fortran-77)	Parameter	Data Type		Description
		C	Fortran-77	
VSsetattr (vsfsnat/vsfscat)	vdata_id	int32	integer	Vdata identifier.
	field_index	int32	integer	Vdata field index.
	attr_name	char *	character* (*)	Name of the attribute.
	data_type	int32	integer	Data type of the attribute.
	count	int32	integer	Number of values the attribute contains.
	values	VOIDP	integer (*)/char-acter* (*)	Buffer containing the attribute values.

4.8.3 Querying the Total Number of Vdata Attributes: VSnattrs

VSnattrs returns the number of attributes of the specified vdata *and* the vdata fields contained in it. This is different from the **VSfnattrs** routine, which returns the number of attributes of a field contained in the specified vdata *or* the specified vdata.

VSnattrs returns total number of attributes assigned to this vdata and its fields when successful, and `FAIL` otherwise

TABLE 4K

VSnattrs Parameter List

Routine Name (Fortran-77)	Parameter	Data Type		Description
		C	Fortran-77	
VSnattrs (vsfnats)	vdata_id	int32	integer	Vdata identifier.

4.8.4 Querying the Number of Attributes of a Vdata or a Vdata Field: VSfnattrs

VSfnattrs returns the number of attributes attached to the specified vdata field *or* the number of attributes attached to the vdata identified by *vdata_id*. This is different from the **VSnattrs** routine, which returns the number of attributes of the specified vdata *and* the fields contained in it.

If *field_index* is set to `_HDF_VDATA`, the number of attributes attached to the vdata referred to by the value of *vdata_id* is returned. When using the Fortran-77 version of this routine, specify a *field_index* value of `-1` to return the number of vdata attributes.

As with **VSsetattr**, if *field_index* is set to a zero-based integer value, it will be used as the index of the target vdata field, and the attributes attached to that field will be returned.

VSfnattrs returns number of attributes assigned to this vdata or its fields when successful, and `FAIL` otherwise.

TABLE 4L **VSfnattrs Parameter List**

Routine Name (Fortran-77)	Parameter	Data Type		Description
		C	Fortran-77	
VSfnattrs **(vsffnas)**	vdata_id	int32	integer	Vdata identifier.
	field_index	int32	integer	Index of the target field.

4.8.5 Retrieving the Index of a Vdata Attribute Given the Attribute Name: VSfindattr

VSfindattr returns the index of an attribute with the specified name. The attribute must to attached to the vdata identified by the the specified vdata identifier.

If *field_index* is set to `_HDF_VDATA`, the index of the attribute attached to the vdata referred to by the value of *vdata_id* is returned. When using the Fortran-77 version of this routine, specify a *field_index* value of `-1` to return the vdata attribute indices.

As with **VSsetattr**, if *field_index* is set to a zero-based integer value, the value will be used as the index of the target vdata field, and the index of the attribute with the name specifiedby the *attr_name* parameter will be returned.

VSfindattr returns the index of the target attribute if successful, and `FAIL` otherwise.

TABLE 4M **VSfindattr Parameter List**

Routine Name (Fortran-77)	Parameter	Data Type		Description
		C	Fortran-77	
VSfindattr **(vsffdat)**	vdata_id	int32	integer	Vdata identifier.
	field_index	int32	integer	Index of the target field.
	attr_name	char *	character* (*)	Name of the target attribute.

4.8.6 Querying Information on a Given Vdata Attribute: VSattrinfo

VSattrinfo returns the name, data type, number of values, and the size of the values of the specified attribute of the specified vdata field or vdata.

The values of the *attr_name*, *data_type*, *count* and *size* parameters can be set to `NULL`, if the information returned by these parameters are not needed.

The *field_index* in **VSattrinfo** is the same as with the *field_index* parameter in **VSsetattr**. It can be set to either a integer field index, or `_HDF_VDATA` to specify the vdata referred to by *vdata_id*.

VSattrinfo returns `SUCCEED` if successful, and `FAIL` otherwise.

TABLE 4N **VSattrinfo Parameter List**

Routine Name (Fortran-77)	Parameter	Data Type		Description
		C	Fortran-77	
VSattrinfo (vsfainf)	vdata_id	int32	integer	Vdata identifier.
	field_index	int32	integer	Index of the target field
	attr_index	intn	integer	Index of the target attribute.
	attr_name	char *	character* (*)	Returned name of the target attribute.
	data_type	int32 *	integer	Returned data type of the target attribute.
	count	int32 *	integer	Number of values of the target attribute.
	size	int32 *	integer	Size, in bytes, of the values of the target attribute.

4.8.7 Querying the Values of a Given Vdata Attribute: VSgetattr

VSgetattr returns all of the values of the specified attribute of the specified vdata field or vdata.

The *attr_index* parameter is the ordinal, zero-based (as with *field_index* in **VSsetattr** and this routine) index of the target attribute.

If *field_index* is set to `_HDF_VDATA`, the value of the attribute attached to the vdata referred to by the value of *vdata_id* is returned. If *field_index* is set to a zero-based integer value, the value will be used as the index of the target vdata field, and the attribute located at the ordinal position specified by *attr_index* will be returned.

VSgetattr returns `SUCCEED` if successful, `FAIL` otherwise.

TABLE 4O **VSgetattr Parameter List**

Routine Name (Fortran-77)	Parameter	Data Type		Description
		C	Fortran-77	
VSgetattr (vsfgnat/vsfgcat)	vdata_id	int32	integer	Vdata identifier.
	field_index	int32	integer	Index of the target field.
	attr_index	intn	integer	Index of the target attribute.
	values	VOIDP	<valid numeric data type>	Buffer containing attribute values.

4.8.8 Determining if the Given Vdata is an Attribute: VSisattr

As attributes are stored by the HDF library as vdatas, a means of testing whether or not a particular vdata is attribute data is provided by the **VSisattr** routine.

VSisattr returns `TRUE` if the vdata is an attribute, and `FALSE` otherwise.

TABLE 4P

VSisattr Parameter List

Routine Name (Fortran-77)	Parameter	Data Type		Description
		C	Fortran-77	
VSisattr (vsfisat)	vdata_id	int32	integer	Vdata identifier.

EXAMPLE 7.

Attaching and Querying Vdata Attributes

These examples attach an attribute to the vdata created in Example 3 (in the file named "Example3.hdf") and to the first field of that vdata. Then it checks the number of attributes in the file and reads the attribute data.

C:

```
#include "hdf.h"
#include "vg.h"

#define FILE_NAME "Example3.hdf"
#define VATTR_NAME "Vdata Attribute 1"
#define FATTR_NAME "Field Attribute 1"

main( )
{

int32 file_id, vdata_ref, vdata_id, status;
int32 n_vdattrs, n_fldattrs;
int32 v_type, v_count, v_size, f_type, f_count, f_size;
char vd_attr[6] = {'m','N','p', 'S', 't', '\0'};
char vattr_buf[6];
char vattrname[30], fattrname[30];
float32 fld_attr[2] = {30.452, 5.467};
float32 fattr_buf[2];

/* Open the HDF file. */
file_id = Hopen(FILE_NAME, DFACC_RDWR, 0);

/* Initialize the vdata interface. */
status = Vstart(file_id);

/* Get the reference number of the target vdata. */
vdata_ref = VSfind(file_id, "Example Vdata");

/* Attach to the target vdata. */
vdata_id = VSattach(file_id, vdata_ref, "w");

/* Attach an attribute to the vdata. */
status = VSsetattr(vdata_id, _HDF_VDATA, VATTR_NAME, DFNT_CHAR,\
                6, vd_attr);

/* Attach an attribute to the first field in the vdata. */
```

```
status = VSsetattr(vdata_id, 0, FATTR_NAME, DFNT_FLOAT32, 2,
            fld_attr);

/* Get the number of attributes attached to the first field -
should be 1. */
n_fldattrs = VSfnattrs(vdata_id, 0);

/* Get the total number of field and vdata attributes.
This should be 2. */
n_vdattrs = VSnattrs(vdata_id);

/* Get information about the vdata attribute. */
status = VSattrinfo(vdata_id, _HDF_VDATA, 0, vattrname, \
        &v_type, &v_count, &v_size);

/* Get information about the field attribute. */
status = VSattrinfo(vdata_id, 0, 0, fattrname, &f_type,
            &f_count, &f_size);

/* Get the vdata attribute. */
status = VSgetattr(vdata_id, _HDF_VDATA, 0, vattr_buf);

/* Get the field attribute. */
status = VSgetattr(vdata_id, 0, 0, fattr_buf);

/* Detach from the vdata, close the VS interface and the file. */
status = VSdetach(vdata_id);
status = Vend(file_id);
status = Hclose(file_id);

}
```

FORTRAN:

```
     PROGRAM CREATE QUERY ATTRS

      integer file_id, vdata_ref, vdata_id, status
      integer n_fldattrs, n_vdattrs
      integer hopen, vfstart, vsffnd, vsfatch, vsfdtch, hclose
      integer vfend, vsfscat, vsfsnat, vsffnas, vsfnats
      integer vsfainf, vsfgnat, vsfgcat
      integer v_type, v_count, v_size, f_type, f_count, f_size
      double precision fld_attr(2), fattr_buf(2)
      character*20 vattrname, vattr_buf
      character*50 filename, vd_attr, fattr_name
      parameter (filename = 'Example3.hdf',
     +      vattr_name = 'Vdata Attribute 1',
     +      vd_attr = 'mnpst',
     +      fattr_name = 'Field Attribute 1'
     +     )

C     The following parameters are defined in hdf.inc.
      integer DFACC_RDWR, DFNT_CHAR, DFNT_FLOAT32, _HDF_VDATA
      parameter (DFACC_RDWR = 3, DFNT_CHAR = 4, DFNT_FLOAT32 = 5,
     +           _HDF_VDATA = -1)

      fld_attr(1) = 30.452
      fld_attr(2) = 5.467

C     Open an HDF file with full access.
      file_id = hopen(filename, DFACC_RDWR, 0)

C     Initialize the VS interface.
```

```
      status = vfstart(file_id)

C     Get the reference number of the target vdata.
      vdata_ref = vsffnd(file_id, 'Example Vdata')

C     Attach to the target vdata.
      vdata_id = vsfatch(file_id, vdata_ref, 'w')

C     Attach an attribute to the vdata.
      status = vsfscat(vdata_id, _HDF_VDATA, vattr_name,
     +                 DFNT_CHAR, 5, vd_attr)

C     Attach an attribute to the first field of the vdata.
      status = vsfsnat(vdata_id, _HDF_VDATA, fattr_name,
     +                 DFNT_FLOAT32, 2, fld_attr)

C     Get the number of attributes attached to the first field.
      n_fldattrs = vsffnas(vdata_id, 0)

C     Get the total number of field and vdata attributes.
      n_vdattrs = vsfnats(vdata_id)

C     Get information about the vdata attribute.
      status = vsfainf(vdata_id, _HDF_VDATA, 0, vattrname,
     +               v_type, v_count, v_size)

C     Get information about the field attribute.
      status = vsfainf(vdata_id, 0, 0, vattrname, f_type,
     +               f_count, f_size)

C     Get the vdata attribute.
      status = vsfgnat(vdata_id, _HDF_VDATA, 0, vattr_buf)

C     Get the field attribute.
      status = vsfgcat(vdata_id, 0, 0, fattr_buf)

C     Detach from the vdata, close the VS interface and the file.
      status = vsfdtch(vdata_id)
      status = vfend(file_id)
      status = hclose(file_id)

      end
```

4.9 Obtaining Information About a Specific Vdata

Once a vdata has been located, its contents must be determined. In this section three categories of routines that obtain vdata information are described:

- A general inquiry routine named **VSinquire**.
- As a supplement to **VSinquire**, a set of ***vdata query*** routines with names prefaced by "VSQuery".
- A set of routines prefaced by "VS". Some of the functionality of these routines are duplicated in the functionality of **VSinquire**.
- A set of ***field query*** routines with names prefaced by "VF".

4.9.1 Obtaining Vdata Information: VSinquire

VSinquire is a general inquiry routine used to request information about the contents of a vdata. It has the following syntax:

```
status = VSinquire(vdata_id, n_records, interlace, fields, vdata_size,
                   vdata_name);
```

The `n_records` parameter contains the returned number of records in the target vdata, the `interlace` parameter contains the returned interlace mode of the vdata contents, the `fields` parameter is a comma-separated list of the returned names of all fields in the vdata, the `vdata_size` parameter is the returned size, in bytes, of the vdata and the `vdata_name` parameter contains the returned name of the vdata.

If any of the parameters are set to `NULL`, the corresponding data is not returned.

The parameters of **VSinquire** are further defined in Table 4Q below.

TABLE 4Q **VSinquire Parameter List**

Routine Name (Fortran-77)	Parameter	Data Type		Description
		C	Fortran-77	
VSinquire (vsfinq)	vdata_id	int32	integer	Vdata identifier.
	n_records	int32 *	integer	Number of records in the vdata.
	interlace	int32 *	integer	Interlace mode.
	fieldname_list	char *	character* (*)	Buffer for the returned list of field names.
	vdata_size	int32 *	integer	Size in bytes of a vdata record.
	vdata_name	char *	character* (*)	Name of the vdata.

EXAMPLE 8. **Obtaining Vdata Information**

The following examples search for the vdata named "Example Vdata".

C:

```
#include "hdf.h"

#define FILE_NAME "Example3.hdf"

main( )
{

int32 file_id, vdata_ref, vdata_id, status;
int32 n_records, interlace, vdata_size;
char fields[30], vdata_name[VSNAMELENMAX];
int8 found_fields;

/* Open the HDF file. */
file_id = Hopen(FILE_NAME, DFACC_RDONLY, 0);

/* Initialize the vdata interface. */
status = Vstart(file_id);

vdata_ref = -1;
found_fields = 0;

/* Attach to each vdata and search for the fields. */
```

```
while ((vdata_ref = VSgetid(file_id, vdata_ref)) != -1) {
   vdata_id = VSattach(file_id, vdata_ref, "r");
    status = VSinquire(vdata_id, &n_records, &interlace, fields,
                           &vdata_size, vdata_name);
    status = VSdetach(vdata_id);
   if (!strncmp(vdata_name, "Example Vdata", 20)) {
      found_fields = 1;
      break;
   } else VSdetach(vdata_id);
}

if (!found_fields)
   printf("Vdata name - Example Vdata - was not found.\n");
else
   printf("Vdata name - Example Vdata - found, vdata id %d.\n",
                         vdata_ref);

/* Detach from the vdata, close the VS interface and the file. */
status = VSdetach(vdata_id);
status = Vend(file_id);
status = Hclose(file_id);

}
```

FORTRAN:

```
     PROGRAM SEARCH VDATANAME

      integer*4 file_id, vdata_ref
      integer*4 n_records, interlace, vdata_size, vdata_id
      character fields *30, vdata_name *20
      logical found_fields
      integer hopen, vsfgid, vsfatch, vsfinq, vsfdtch, hclose

C     DFACC_RDONLY is defined in hdf.inc.
      integer DFACC_RDONLY
      parameter (DFACC_RDONLY = 1)

      character target_vdata_name *20
      parameter (target_vdata_name = 'Example Vdata')

C     Open an HDF file with full access.
      file_id = hopen('Example3.hdf', DFACC_RDONLY, 0)

C     Initialize the VS interface.
      call vfstart(file_id)

      vdata_ref = -1
      found_fields = .FALSE.

C     Attach to each vdata and search for the fields.
10    vdata_ref = vsfgid(file_id, vdata_ref)
      if (vdata_ref .ne. -1) then
        vdata_id = vsfatch(file_id, vdata_ref, 'r')
        status = vsfinq(vdata_id, n_records, interlace, fields,
     +                 vdata_size, vdata_name)
        status = vsfdtch(vdata_id)

        if (vdata_name .eq. target_vdata_name) then
           found_fields = .TRUE.
           go to 20
        else
            go to 10
```

```
            end if
          end if

20        if (found_fields .eq. .FALSE.) then
            print *, 'Vdata name - Example Vdata - was not found'
          else
            print *, 'Vdata name - Example Vdata - found, vdata id ',
     +                 vdata_ref
          end if

C         Detach from the vdata, close the Vset interface and the file.
          status = vfend(file_id)
          status = hclose(file_id)

          end
```

4.9.2 VSQuery Vdata Information Retrieval Routines

The syntax of the VSQuery routines are as follows:

```
status = VSQueryname(vdata_id, *vdata_name);
status = VSQueryclass(vdata_id, *vdata_class);
status = VSQueryfields(vdata_id, fields);
status = VSQueryinterlace(vdata_id, *interlace);
status = VSQuerycount(vdata_id, *n_records);
vdata_tag = VSQuerytag(vdata_id);
vdata_ref = VSQueryref(vdata_id);
status = VSQueryvsize(vdata_id, *vdata_vsize);
```

All VSQuery routines except **VSQuerytag** and **VSQueryref** have two arguments. The second argument is the type of vdata information being requested. **VSQuerytag** and **VSQueryref** are functions that retrieve the vdata information through their return value.

- **VSQueryname** retrieves the name of the specified vdata.
- **VSQueryfields** retrieves the name of the fields in the specified vdata.
- **VSQueryinterlace** retrieves the interlace mode of the specified vdata.
- **VSQuerytag** returns the tag of the specified vdata.
- **VSQuerycount** returns the number of records in the specified vdata.
- **VSQueryref** returns the reference number of the specified vdata
- **VSQueryvsize** retrieves the size, in bytes, of a record in the specified vdata.

The parameters of these routines are listed in the following table.

TABLE 4R

VSQuery Routine Parameter List

Routine Name (Fortran-77)	Parameter	Data Type		Description
		C	Fortran-77	
VSQueryname (vsqfname)	vdata_id	int32	integer	Vdata identifier.
	vdata_name	char *	character* (*)	Name of the vdata.
VSQueryfields (vsqfflds)	vdata_id	int32	integer	Vdata identifier.
	fields	char *	character* (*)	Comma-separated list of the field names in the vdata.
VSQueryinterlace (vsqfintr)	vdata_id	int32	integer	Vdata identifier.
	interlace	int32 *	integer	Interlace mode.

VSQuerycount (vsqnfelt)	vdata_id	int32	integer	Vdata identifier.
	n_records	int32 *	integer	Number of records in the vdata.
VSQueryvsize (vsqfvsiz)	vdata_id	int32	integer	Vdata identifier.
	vdata_size	int32 *	integer	Size in bytes of the vdata record.
VSQuerytag (vsqtag)	vdata_id	int32	integer	Vdata identifier.
VSQueryref (vsqref)	vdata_id	int32	integer	Vdata identifier.

4.9.3 VS Vdata Information Retrieval Routines

Some routines with names prefaced by "VS" are used to obtain specific types of vdata information. Here is the syntax of these routines:

```
num_of_elements = VSelts(vdata_id);
status = VSgetclass(vdata_id, vdata_class);
num_of_fields = VSgetfields(vdata_id, field_names);
interlace_mode = VSgetinterlace(vdata_id);
status = VSgetname(vdata_id, vdata_name);
size_of_fields = VSsizeof(vdata_id, field_name_list);
```

With the exception of **VSgetclass**, the information obtainable through these routines can also be obtained through **VSinquire**. **VSinquire** provides a way to query commonly used vdata information with one standardized routine call. However, as there are situations where the HDF programmer may not want all of the information **VSinquire** provides, the VS routines have also been provided.

- **VSelts** returns the number of elements in the specified vdata.
- **VSgetclass** retrieves the class of the specified vdata.
- **VSgetfields** returns the number of and retrieves the names of all fields in the specified vdata.
- **VSgetinterlace** retrieves the interlace mode of the specified vdata.
- **VSgetname** retrieves the name of the specified vdata.
- **VSsizeof** returns the size, in bytes, of the specified field or fields.

The parameters of these routines are described in the following table.

TABLE 4S **VSelts, VSgetclass, VSgetfields, VSgetinterlace, VSgetname and VSsizeof Parameter List**

Routine Name (Fortran-77)	Parameter	Data Type		Description
		C	Fortran -77	
VSelts (vsfelts)	vdata_id	int32	integer	Vdata identifier.
VSgetclass (vsfcls)	vdata_id	int32	integer	Vdata identifier.
	vdata_class	char *	character* (*)	Pointer to the class name.
VSgetfields (vsfgfld)	vdata_id	int32	integer	Vdata identifier.
	fields	char *	character* (*)	Pointer to the field names.
VSgetinterlace (vsfgint)	vdata_id	int32	integer	Vdata identifier.
VSgetname (vsfgnam)	vdata_id	int32	integer	Vdata identifier
	vdata_name	char *	character* (*)	Pointer to the vdata name.
VSsizeof (vsfsiz)	vdata_id	int32	integer	Vdata identifier.
	fields	char *	character* (*)	List of field names to be queried.

4.9.4 VF Field Information Retrieval Routines

Routines whose names of prefaced by "VF" are used for obtaining information about specific fields in a vdata. Here is the syntax of these routines:

```
field_name = VFfieldname(vdata_id, field_index);
field_file_size = VFfieldesize(vdata_id, field_index);
field_mem_size = VFfieldisize(vdata_id, field_index);
num_of_fields = VFnfields(vdata_id);
field_order = VFfieldorder(vdata_id, field_index);
field_type = VFfieldtype(vdata_id, field_index);
```

The second, `field_index` is the index number that uniquely identifies the location of the field within the vdata. Field index numbers are assigned in increasing order and are zero-based.

The values returned by each of the VF routines are as follows:

- **VFfieldname** returns the name of the specified field.
- **VFfieldesize** returns the size of the specified field as stored in the HDF file. This is the size of the field as tracked by the HDF library.
- **VFfieldisize** returns the size of the specified field as stored in memory. This is the native machine size of the field.
- **VFnfields** returns the number of fields in the specified vdata.
- **VFfieldorder** returns the order of the specified field.
- **VFfieldtype** returns the data type of the specified field.

If unsuccessful, these routines return `FAIL` (or `-1`). The parameters of these routines are described in the following table.

TABLE 4T **VFfieldname, VFfieldesize, VFfieldisize, VFnfields, VFfieldorder and VFfieldtype Parameter List**

Routine Name (Fortran-77)	Parameter	Data Type		Description
		C	Fortran-77	
VFfieldname (vffname)	vdata_id	int32	integer*4	Vdata identifier.
	field_index	int32	integer	Field index.
VFfieldesize (vffesiz)	vdata_id	int32	integer	Vdata identifier.
	field_index	int32	integer	Field index.
VFfieldisize (vffisiz)	vdata_id	int32	integer	Vdata identifier.
	field_index	int32	integer	Field index.
VFnfields (vfnflds)	vdata_id	int32	integer	Vdata identifier.
VFfieldorder (vffordr)	vdata_id	int32	integer	Vdata identifier.
	field_index	int32	integer	Field index.
VFfieldtype (vfftype)	vdata_id	int32	integer	Vdata identifier.
	field_index	int32	integer	Field index.