Chapter 5

Vgroups (V API)

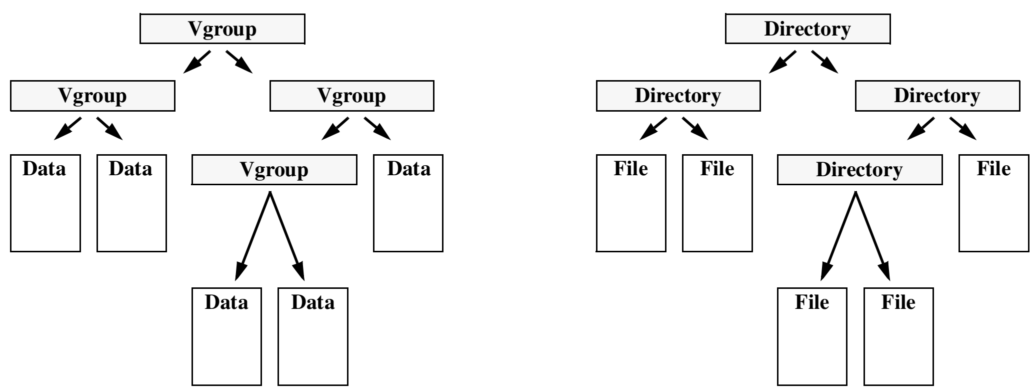
5.1 Chapter Overview

This chapter describes the vgroup data model and the vgroup API. The first section describes the vgroup data model. The second section presents the vgroup API, followed by a presentation of a programming model for vgroups. The next three sections describe the use of the API in accessing vgroups, creating vgroups and reading from vgroups. The final two sections cover vgroup utilities and obsolete vgroup routines.

5.2 The Vgroup Data Model

A ***vgroup*** is a structure designed to associate related objects. Data organization within a vgroup resembles the UNIX file system. (See Figure 5a.) The general structure of vgroups are similar to UNIX directories or subdirectories in that a vgroup may contain references to other vgroups or data objects. In previous versions of HDF, the data objects in a vgroup were limited to vdatas. Any HDF data object can now be included within a vgroup.

FIGURE 5a **Similarity of the HDF Vgroup Structure and the UNIX File System**



5.2.1 Vgroup Names and Classes

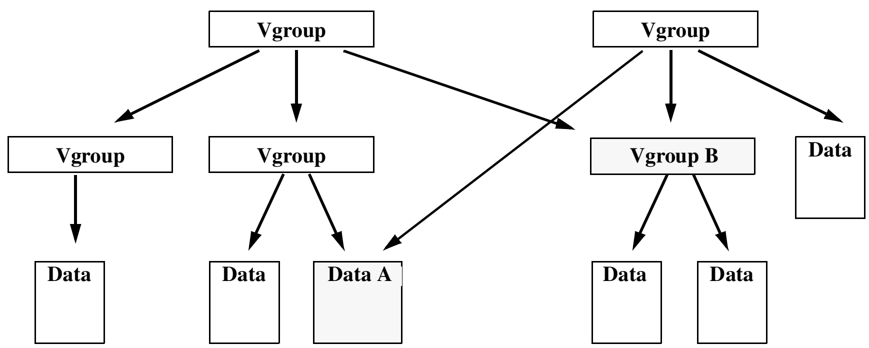
Vgroups have attributes that can be used for the purpose of searching for and classifying them. Two such attributes are the vgroup name and the vgroup class. A ***vgroup name*** is a character string that can be used to describe the contents of a specific vgroup. A vgroup name can be used as a primary key in searches.

A ***vgroup class*** is also a character string and can be also used in searches for a specific vgroup. The purpose of a vgroup class is to allow an application to determine the intended use of its members. For example, a vdata object named "Storm Tracking Data - 5/11/94" and another vdata object named "Storm Tracking Data - 6/23/94" can be grouped together under a vgroup named "Storm Tracking Data - 1994". If the data was collected in Anchorage, Alaska the class name might be "Anchorage Data", particularly if other vgroups contain storm track data collected in different locations. The specific use of the vgroup name and class name is solely determined by the HDF users.

5.2.2 Vgroup Organization

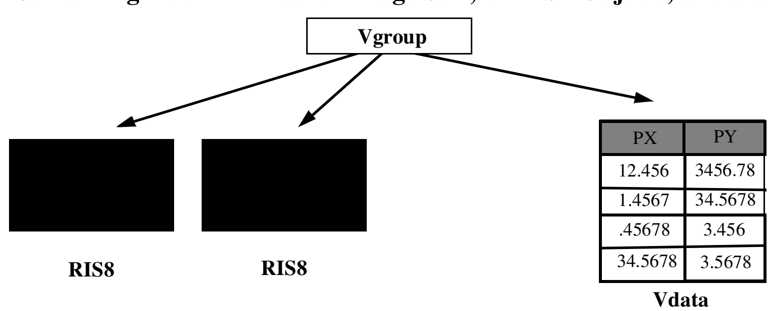
There are many ways to organize vgroups through the use of the Vgroup API. Vgroups may contain any number of groups and data objects, including data objects and vgroups that are members of other vgroups. Therefore, an object may have more than one parent vgroup. For example, Data object A and Vgroup B, shown in Figure 5b, are members of multiple vgroups with different organizational structures.

FIGURE 5b **Sharing Vgroups and Vdatas Among Vgroups**



A vgroup can contain any combination of HDF data objects. Figure 5c illustrates a vgroup that contains two raster image sets and a vdata.

FIGURE 5c **A Vgroup Containing Two 8-Bit Raster Image Sets, or RIS8 Objects, and a Vdata**



PX	PY
12.456	3456.78
1.4567	34.5678
.45678	3.456
34.5678	3.5678

5.2.3 Vgroup Conventions: An Example Using Vsets

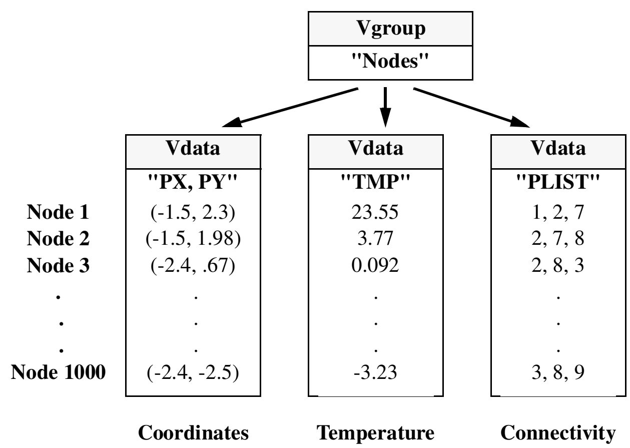
Although vgroups can contain any combination of objects, it is often useful to establish conventions on the content and structure of vgroups. For example, a type of vgroup called a ***vertex set*** or ***vset***, is often used by scientific and graphics programmers to describe the surfaces of an object as well as its properties. This structure is used by NCSA Polyview for viewing meshes and other objects that can be described by polygonal data.

A vset consists of one list of coordinate data, one list of connectivity data and one list of node property data. These three lists are stored in separate vdata objects within the vset.

Each n-dimensional coordinate in the coordinate list defines the relative location of a vertex, or ***node***. Each item of connectivity data represents an edge between two vertices and a list of this edge data is referred to as a ***connectivity list*** , which describes one polygon. Connectivity lists are a table of comma-separated numbers, with each number representing the position of one particular node in the table of nodes. For example, the number "2" as an item in a connectivity list would represent the second entry in the node table. ***Node properties*** are user-defined values attached to each node within the polygon and can be numbers or characters.

For example, consider a heated mesh of 400 triangles formed by connecting 1000 nodes. A vset describing this mesh might contain the coordinates of the vertices, the temperature value of the vertices and a connectivity list describing the edges of the triangles. This vset configuration is the one used most often by graphics applications like NCSA Polyview and is illustrated in Figure 5d below.

FIGURE 5d **Vset Structure Describing a Heated Mesh**



1.3 The Vgroup API

The Vgroup interface consists of routines for accessing, creating and getting information about vgroups.

1.3.1 Vgroup Library Routines

Vgroup API routine names are prefaced by "V". These routines are categorized as follows:

- ***Access routines*** open files, initialize the Vgroup interface and access individual vgroups. They also terminate access to vgroups and the Vgroup interface and close HDF files.
- ***Create routines*** organize, label and add data objects to vgroups.
- ***File inquiry routines*** return information about how vgroups are stored in a file. They are useful for locating vgroups in a file.
- ***Vgroup inquiry routines*** provide specific information about a specific vgroup. This information includes the class, name, member count and additional member information.

The Vgroup API routines, along with the two H interface routines used to open and close HDF files for vgroup access, are listed in Table 5A below.

TABLE 5A **Vgroup Interface Routines**

Category	Routine Name		Description
	C	Fortran-77	
Access	Vstart	vfstart	Initializes the V interface.
	Vattach	vfatch	Establishes access to a vgroup.
	Vdetach	vsfdtch	Terminates access to a vgroup.
	Vend	vfend	Terminates access to the V interface.
Create	Vsetclass	vfscls	Assigns a class to a vgroup.
	Vsetname	vfsnam	Assigns a name to a vgroup.
	Vinsert	vfinsrt	Adds a vgroup or vdata to an existing vgroup.
	Vaddtagref	vfadtr	Adds any HDF data object to an existing vgroup.
	Vsetattr	vfsnatt/vfscatt	Sets the attribute of a vgroup.
File Inquiry	Vlone	vflone	Returns the reference numbers of vgroups not included in other vgroups.
	Vgetid	vfgid	Returns the reference number for the next vgroup in the HDF file.
Vgroup Inquiry	Vinquire	vfinq	Returns general information about a vgroup.
	Vgetclass	vfgcls	Returns the class of the specified vgroup.
	Vgetname	vfgnam	Returns the name of the specified vgroup.
	Visvg	vfisvg	Checks if a vgroup identifier belongs to a vgroup within a vgroup.
	Visvs	vfisvs	Checks if a vdata identifier belongs to a vdata within a vgroup.
	Vgettagref	vfgttr	Retrieves a tag/ reference number pair for a data object in the specified vgroup.
	Vntagrefs	vfntr	Returns the number of tag/reference number pairs contained in the specified vgroup.
	Vgetnext	vfgnxt	Returns the identifier of the next vgroup or vdata in a vgroup.
	Vgettagrefs	vfgttrs	Retrieves the tag/reference pairs of all of the data objects with a vgroup.
	Vinqtagref	vfinqtr	Checks if an object belongs to a vgroup.
	Vgetversion	None	Queries the vgroup version of a given vgroup.
	Vnaattrs	vfnatts	Queries the total number of vgroup attributes.
	Vfindattr	vffdatt	Retrieves the index of an attribute given the attribute name.
	Vattrinfo	vfainfo	Queries information on a given vgroup attribute.
	Vgetattr	vfgnatt/vfgcatt	Queries the values of a given attribute.

1.3.2 Vgroup Identifiers in the Vgroup Interface

The Vgroup interface identifies vgroups in several ways. Before a vgroup is attached, it is uniquely identified by its ***name***, ***class*** and ***reference number***. When a vgroup is attached, it is assigned a vgroup identifier or ***vgroup id***. After a vgroup has been attached, its vgroup id is used by the Vgroup interface routines to access vgroups.

The name, class, and reference number are values assigned to the vgroup when it is created. The name and class are normally assigned by the calling program and may be changed throughout the scope of the vgroup object. However, the reference number is assigned by the HDF library and not changed.

1.4 Programming Model for the Vgroup Interface

The programming model for accessing vgroups is as follows:

1. Open a file.
2. Initialize the V interface
3. Open a vgroup.
4. Perform the desired operations on the vgroup.
5. Dispose of the vgroup id.
6. Terminate access to the V interface.
7. Close the file.

To access a single vgroup in an HDF file, the calling program must contain the following calls:

C:

```
file_id = Hopen(filename, file_access_mode, n_dds);
status = Vstart(file_id);
vgroup_id = Vattach(file_id, vgroup_ref, vgroup_access_mode);
<Optional operations>
status = Vdetach(vgroup_id);
status = Vend(file_id);
status = Hclose(file_id);
```

FORTRAN:

```
file_id = hopen(filename, file_access_mode, n_dds)
status = vfstart(file_id)
vgroup_id = vsfatch(file_id, vdata_ref, vgroup_access_mode)
<Optional operations>
status = vfdtch(vgroup_id)
status = vfend(file_id)
status = hclose(file_id)
```

The calling program must obtain a separate vgroup id for each vgroup to be accessed.

1.4.1 Accessing Files and Vgroups: Hopen, Vstart and Vattach

An HDF file must be opened by **Hopen** before it can be accessed using the V interface. **Hopen** is described in Chapter 2, titled *HDF Fundamentals*.

The Vgroup interface routines are used in a similar manner to the V interface routines. Before performing operations on a vgroup, a calling program must call **Vstart** for every file it intends to access and **Vattach** for every vgroup it intends to access. **Vstart** initializes the internal vgroup structures in a file. Consequently, any HDF file involved in a vgroup operation must be initialized by **Vstart** before the operation begins. **Vstart** takes one argument, the file id returned by **Hopen**.

Vattach provides access to an individual vgroup for all read and write operations. It takes three arguments: `file_id`, `vgroup_ref` and the `access_mode`. The argument `file_id` is a file identifier returned by **Hopen** and `vgroup_ref` is the reference number that identifies the vgroup to be accessed. Specifying a nonexistent reference number will return an error, specifying a value of -1 for the reference number will create a new vgroup and specifying an existing reference number will provide access to the corresponding vgroup. To find the reference number of a vgroup, consider using either **Vgetnext** or **Vgetid**.

The `access_mode` parameter in **Vattach** specifies the type of access ("r" or "w") required for operations on the selected vgroup. This argument is not currently used; all vgroups are automatically opened with read and write access. Multiple attaches may be made to the same vgroup, which will result in several vgroup identifiers being assigned to the same vgroup.

The parameters of **Vstart** and **Vattach** are defined in Table 5B below.

1.4.2 Terminating Access to Vgroups and Files: Vdetach, Vend and Hclose

Successfully terminating access to a vgroup requires one **Vdetach** call for every **Vattach** call made. The calling program must also call **Vend** once for every **Vstart** call made. All **Vdetach** calls must occur prior to the first **Vend** call for the same file. **Vdetach** terminates access by updating internal library structures and frees all memory associated with the vgroup. Once a vgroup is detached, its vgroup id is invalid and any attempts to access it will result in an error condition.

Vend releases all internal data structures allocated by calling **Vstart**. Its one argument is `file_id`, the file identifier returned by **Hopen**. **Vend** must be called once per file and only after all vgroups are detached. Attempts to use the Vgroup interface after calling **Vend** will produce errors.

Hclose terminates access to a file only if there are no vgroup ids attached to the file. To successfully close a file, individually release all vgroup idsby making one **Vdetach** call for each **Vattach** call.

The parameters of **Vdetach** and **Vend** are also defined in the following table.

TABLE 5B **Vstart, Vattach, Vdetach and Vend Parameter List**

Routine Name (Fortran-77)	Parameter	Data Type		Description
		C	Fortran-77	
Vstart (vstart)	file_id	int32	integer	File identifier.
Vattach (vfatch)	file_id	int32	integer	File identifier.
	vgroup_ref	int32	integer	Reference number for existing vgroup or -1 for new.
	access_mode	char *	character* (*)	Access mode of the vgroup operation.
Vdetach (vfdtch)	vgroup_id	int32	integer	Vgroup identifier.
Vend (vend)	file_id	int32	integer	IFile identifier.

1.5 Creating and Writing to a Vgroup

There are two steps involved in the creation of vgroups. First, the vgroup must be created, then the data objects must be inserted into it. Any HDF data object can be inserted into a vgroup. Creation and insertion operations are usually performed at the same time - although, as with the VS interface routines it isn't required to do so.

1.5.1 Creating a Vgroup:Vattach

Creating a vgroup involves the following steps:

1. Open the file.
2. Initialize the V interface.
3. Create the new vgroup.
4. Optionally assign a vgroup name.
5. Optionally assign a vgroup class.
6. Optionally insert the data objects.

7. Terminate access to the vgroup.
8. Terminate access to the V interface.
9. Close the file.

These steps correspond to the following calling sequence:

C:
```
file_id = Hopen(filename, file_access_mode, n_dds);
status = Vstart(file_id);
vgroup_id = Vattach(file_id, vgroup_ref, vgroup_access_mode);
status = Vsetname(vgroup_id, vgroup_name);
status = Vsetclass(vgroup_id, vgroup_class);
num_of_tag_refs = Vaddtagref(vgroup_id, tag, ref); or
obj_pos = Vinsert(vgroup_id, v_id);
status = Vdetach(vgroup_id);
status = Vend(file_id);
status = Hclose(file_id);
```

FORTRAN:
```
file_id = hopen(filename, file_access_mode, n_dds)
status = vfstart(file_id)
vgroup_id = vfatch(file_id, vgroup_ref, vgroup_access_mode)
status = vfsnam(vgroup_id, vdata_name)
status = vfscls(vgroup_id, vdata_class)
num_of_tag_refs = vfadtr(vgroup_id, tag, ref) or
obj_pos = vfinsrt(vgroup_id, v_id)
status = vfdtch(vgroup_id)
status = vfend(file_id)
status = hclose(file_id)
```

In the routines that follow, `vgroup_id` is the vgroup identifier returned by **Vattach.**

When a new vgroup is created, the value of `vgroup_ref` must be set to `-1` and the value of `access_mode` must be `"w"`.

1.5.2 Assigning a Vgroup Name and Class: Vsetname and Vsetclass

Vsetname assigns a name to a vgroup. The parameter `vgroup_name` is a character string with the name to be assigned to the vgroup. If **Vsetname** is not called, a vgroup name is set to a zero-length string. A name may be assigned and reset any time after the vgroup is created.

Vsetclass assigns a class to a vgroup. The parameter `vgroup_class` is a character string with the class name to be assigned to the vgroup. If **Vsetclass** is not called, a vgroup class is set to a zero-length string. As with the vgroup names, the class may be set and reset at any time after the vgroup is created.

Vsetname and **Vsetclass** are further described below. (See Table 5C on page 148.)

1.5.3 Inserting Any HDF Data Object Into a Vgroup: Vaddtagref

The routine most commonly used for inserting objects into a vgroup is **Vaddtagref**. Data objects may be added to a vgroup when the vgroup is created or at any point thereafter.

The `tag` and `ref` parameters in **Vaddtagref** are the tag and reference number of the data object to be inserted into the vgroup respectively. **Vaddtagref** returns the total number of tag and reference number pairs in the vgroup if the operation is successful and `FAIL` (or `-1`) otherwise.

Vaddtagref is firther described below. (See Table 5C on page 148.)

1.5.4 Inserting a Vdata or Vgroup Into a Vgroup: Vinsert

Vinsert is a routine designed specifically for inserting vdatas or vgroups into a parent vgroup. To use **Vinsert**, you must provide the vdata id of the parent vgroup, as well as the vdata id or the vgroup id of the vdata or vgroup to be inserted. These ids are obtained by inserting the vgroups and vdatas in the manner described in the preceeding subsection and in Chapter 4, titled *Vdatas (VS API)*.

The `v_id` parameter in **Vinsert** is either a vdata id or a vgroup id, depending on whether a vdata or vgroup is to be inserted. **Vinsert** returns the index of the inserted vdata or vgroup if the operation is successful and `FAIL` (or `-1`) otherwise.

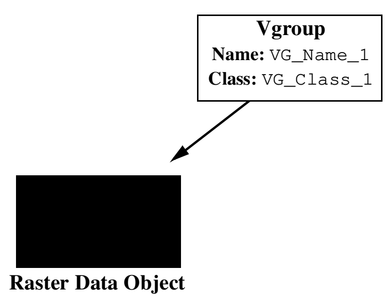
Vinsert is further defined in the following table.

TABLE 5C **Vsetname, Vsetclass, Vaddtagref and Vinsert Parameter List**

Routine Name (Fortran-77)	Parameter	Data Type		Description
		C	Fortran-77	
Vsetname (vfsnam)	vgroup_id	int32	integer	Vgroup identifier.
	vgroup_name	char *	character* (*)	Vgroup name.
Vsetclass (vfscls)	vgroup_id	int32	integer	Vgroup identifier.
	vgroup_class	char *	character* (*)	Vgroup class.
Vaddtagref (vfadtr)	vgroup_id	int32	integer	Vgroup identifier.
	tag	int32	integer	Tag of the object to be inserted.
	ref	int32	integer	Reference number of the object to be inserted.
Vinsert (vfinsrt)	vgroup_id	int32	integer	Vgroup identifier.
	v_id	int32	integer	Vgroup or vdata identifier of the object to be inserted.

EXAMPLE 1. **Creating a Vgroup Containing a Raster Image**

The following examples use **Vaddtagref** to insert an 8-bit raster image into a vgroup that is created with the name "VG_Name_1" and the class "VG_Class_1". The following figure illustrates what these examples do. The C and Fortran-77 code uses two different means of initializing the example raster image set, but in most real-world applications this set would be read from an external data file using the RIS8 routines described in Chapter 6, titled *8-Bit Raster Images (DFR8 API)*.



C:

```
#include "hdf.h"

#define HEIGHT 6
```

```
#define WIDTH 5

main( )
{

    int32 file_id, vgroup_id, vdata_id, status;
    uint16 tag, ref;

    /* Construct the image to be written to the vgroup. */
    static uint8 raster_data[HEIGHT][WIDTH] =
                            { 1, 2, 3, 4,  5,
                6,  7,  8,  9, 10,
                11, 12, 13, 14, 15,
                16, 17, 18, 19, 20,
                21, 22, 23, 24, 25,
                26, 27, 28, 29, 30 };

    /* Open an HDF file with full access. */
    file_id = Hopen("Example1.hdf", DFACC_CREATE, 0);

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

    /* Create a vgroup. */
    vgroup_id = Vattach(file_id, -1, "w");

    /* Set the name and class for this vgroup. */
    status = Vsetname(vgroup_id, "VG_Name_1");
    status = Vsetclass(vgroup_id, "VG_Class_1");

    /* Write the data to file and determine its tag and ref number. */
    status = DFR8addimage("Example1.hdf", (VOIDP) raster_data, WIDTH,
                          HEIGHT, 0);
    ref = DFR8lastref( );

    /* This tag definition is from hdf.h. */
    tag = DFTAG_RI8;

    /* Insert the data image into the vgroup. */
    status = Vaddtagref(vgroup_id, tag, ref);

    /* Terminate access to the vgroup interface. */
    status = Vdetach(vgroup_id);
    status = Vend(file_id);

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

}
```

FORTRAN:

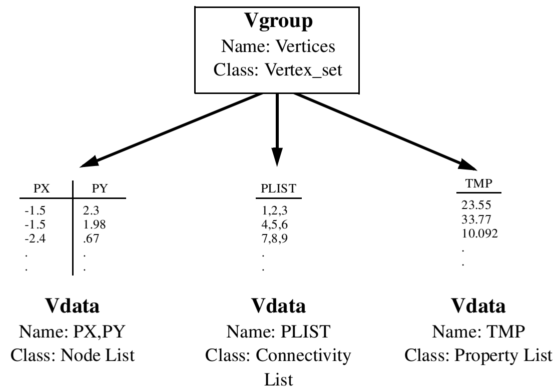
```
     PROGRAM VGROUP INSERT

      integer file_id, vgroup_id, status, tag, ref
      integer*4 raster_data(6, 5)
      integer i, j, k
      integer hopen, vfatch, vfsnam, vfscls
      integer d8aimg, d8lref, vfadtr, vfdtch, vfatch
      integer hclose, vfstart, vfend
      integer DFACC_CREAT
      parameter (DFACC_CREAT = 4)
```

```
C     Construct the data image to be written to the vgroup.
      do 20 j = 1, 5
        do 10 i = 1, 6
            raster_data(i, j) = k
            k = k + 1
10      continue
20    continue

C     Open an HDF file with full access.
C     DFACC_CREAT is defined in hdf.inc.
      file_id = hopen('Example1.hdf', DFACC_CREAT, 0)

C     Open a vgroup with write access.
      status = vfstart(file_id)
      vgroup_id = vfatch(file_id, -1, 'w')

C     Set the name and class for this vgroup.
      status = vfsnam(vgroup_id, 'VG_Name_1')
      status = vfscls(vgroup_id, 'VG_Class_1')

C     Write the data to file and determine its tag and reference
C     number.
      status = d8aimg('Example1.hdf', raster_data, 6, 5, 0)
      ref = d8lref( )

C     The number '202' corresponds to the value of DFTAG_RI8 in
C     hdf.inc.
      tag = 202

C     Insert the data image into the vgroup.
      status = vfadtr(vgroup_id, tag, ref)

C     Terminate access to the V interface.
      status = vfdtch(vgroup_id)
      status = vfend(file_id)

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

      end
```

EXAMPLE 2.

Inserting Three Vdatas Into a Vgroup to Create a Vset

Consider a heated mesh of twenty triangles formed by connecting thirty nodes. The following examples demonstrate how to create a vset containing three vdatas where the vdatas are the coordinate position of each node, the connectivity list describing each relation of three vertices (or triangles) in the vset and a temperature value for each node. The following figure illustrates what is done in the examples. In real-world applications, the contents of the vdatas would be read in from external files, not initialized as in these examples.



C:

```
#include "hdf.h"

main( )
{

int32 file_id, vgroup_id, vdata_id, status;
int32 num_of_elements, vg_position;
float32 pxy[30][2] = {-1.5, 2.3, -1.5, 1.98, -2.4, .67,
                -3.4, 1.46, -.65, 3.1, -.62, 1.23,
                -.4, 3.8, -3.55, 2.3, -1.43, 2.44,
                .23, 1.13, -1.4, 5.43, -1.4, 5.8,
                -3.4, 3.85, -.55, .3, -.21, 1.22,
                -1.44, 1.9, -1.4, 2.8, .94, 1.78,
                -.4, 2.32, -.87, 1.99, -.54, 4.11,
                -1.5, 1.35, -1.4, 2.21, -.22, 1.8,
                -1.1, 4.55, -.44, .54, -1.11, 3.93,
                -.76, 1.9, -2.34, 1.7, -2.2, 1.21};

float32 temp[30];
uint8 mesh[20][3];
uint8 i, j, k = 0;

/* Open an HDF file with full access. */
file_id = Hopen("Example2.hdf", DFACC_CREATE, 0);

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

/* Initialize the data buffer arrays. */
for (i = 0; i < 30; i++)
   temp[i] = i * 10.0;

for (i = 0; i < 20; i++) {
   for (j = 0; j < 3; j++) {
      mesh[i][j] = ++k;
   }
}

/* Create a vgroup with write access, then name it "Vertices". */
vgroup_id = Vattach(file_id, -1, "w");
status = Vsetname(vgroup_id, "Vertices");
```

```
/* Create a vdata to store the x,y values, set its name and class. */
vdata_id = VSattach(file_id, -1, "w");
status = VSsetname(vdata_id, "PX and PY");
status = VSsetclass(vdata_id, "Node List");

/* Specify the PX field information. */
status = VSfdefine(vdata_id, "PX", DFNT_FLOAT32, 2);

/* Specify the PY field information. */
status = VSfdefine(vdata_id, "PY", DFNT_FLOAT32, 2);

/* Set the field names. */
status = VSsetfields(vdata_id, "PX,PY");

/* Write the buffered data into the vdata object. */
num_of_elements = VSwrite(vdata_id, (VOIDP)pxy, 30, FULL_INTERLACE);

/* Insert the vdata into the vgroup. */
vg_position = Vinsert(vgroup_id, vdata_id);

/* Detach from the vdata. */
status = VSdetach(vdata_id);

/* Create a vdata to store the temperature property data. */
vdata_id = VSattach(file_id, -1, "w");
status = VSsetname(vdata_id, "PLIST");
status = VSsetclass(vdata_id, "Connectivity List");
status = VSfdefine(vdata_id, "PLIST", DFNT_FLOAT32, 1);
status = VSsetfields(vdata_id, "PLIST");
num_of_elements = VSwrite(vdata_id, (VOIDP)temp, 30, FULL_INTERLACE);
vg_position = Vinsert(vgroup_id, vdata_id);
status = VSdetach(vdata_id);

/* Create a vdata to store the mesh. */
vdata_id = VSattach(file_id, -1, "w");
status = VSsetname(vdata_id, "TMP");
status = VSsetclass(vdata_id, "Property List");
status = VSfdefine(vdata_id, "TMP", DFNT_INT8, 3);
status = VSsetfields(vdata_id, "TMP");
num_of_elements = VSwrite(vdata_id, (VOIDP)mesh, 20, FULL_INTERLACE);
vg_position = Vinsert(vgroup_id, vdata_id);
status = VSdetach(vdata_id);

/* Terminate access to the "Vertices" vgroup. */
status = Vdetach(vgroup_id);

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

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

}
```

FORTRAN:

```
     PROGRAM VDATA INSERT

      integer*4 file_id, vgroup_id, vdata_id, status, num_of_elements
      integer*4 vg_position
      integer*4 mesh(20, 3)
```

```
      double precision pxy(30, 2), temp(30)
      integer i, j
      integer hopen, vfatch, vfsnam, vsfatch
      integer vsfsfld, vsfwrit, vsfdtch, vsffdef, vfdtch
      integer hclose, vfinsrt, vfstart, vfend, vsfscls, vsfsnam
      data    pxy / -1.5, 2.3, -1.5, 1.98, -2.4, .67,
     +             -3.4, 1.46, -.65, 3.1, -.62, 1.23,
     +            -.4, 3.8, -3.55, 2.3, -1.43, 2.44,
     +            .23, 1.13, -1.4, 5.43, -1.4, 5.8,
     +            -3.4, 3.85, -.55, .3, -.21, 1.22,
     +            -1.44, 1.9, -1.4, 2.8, .94, 1.78,
     +            -.4, 2.32, -.87, 1.99, -.54, 4.11,
     +            -1.5, 1.35, -1.4, 2.21, -.22, 1.8,
     +            -1.1, 4.55, -.44, .54, -1.11, 3.93,
     +            -.76, 1.9, -2.34, 1.7, -2.2, 1.21 /

      integer*4 DFACC_CREAT
      parameter (DFACC_CREAT = 4)

C     Open an HDF file with full access.
C      DFACC_CREAT is defined in hdf.inc.
      file_id = hopen('Example2.hdf', DFACC_CREAT, 0)

C     Create a vgroup with write access, then name it 'Vertices'.
      status = vfstart(file_id)

C     Initialize the data buffer arrays.
      do 10 i = 1, 30
        temp(i) = i * 10.0
10    continue

      do 20 i = 1, 20
        do 30 j = 1, 3
          mesh(i, j) = k
             k = k + 1
30      continue
20    continue
      vgroup_id = vfatch(file_id, -1, 'w')
      status = vfsnam(vgroup_id, 'Vertices')

C     Create a vdata to store the x,y values.  The number
C     '5' in the third argument of vsffdef corresponds to
C     DFNT_FLOAT32 in hdf.inc.
      vdata_id = vsfatch(file_id, -1, 'w')
      status = vsfsnam(vdata_id, 'PX,PY')
      status =  vsfscls(vdata_id, 'Node List')
      status = vsffdef(vdata_id, 'PX', 5, 1)
      status = vsffdef(vdata_id, 'PY', 5, 1)
      status = vsfsfld(vdata_id, 'PX,PY')
      num_of_elements = vsfwrit(vdata_id, pxy, 30, FULL_INTERLACE)
      vg_position = vfinsrt(vgroup_id, vdata_id)
      status = vsfdtch(vdata_id)

C     Create a vdata to store the temperature data.  The number
C     '5' in the third argument corresponds to DFNT_FLOAT32
C     in hdf.inc.
      vdata_id = vsfatch(file_id, -1, 'w')
      status = vsfsnam(vdata_id, 'TMP')
      status =  vsfscls(vdata_id, 'Property List')
      status = vsffdef(vdata_id, 'TMP', 5, 1)
      status = vsfsfld(vdata_id, 'TMP')
      num_of_elements = vsfwrit(vdata_id, temp, 30, FULL_INTERLACE)
      vg_position = vfinsrt(vgroup_id, vdata_id)
```

```
      status = vsfdtch(vdata_id)

C     Create a vdata to store the mesh. The number '24' in the
C     third argument of vsffdef corresponds to DFNT_INT32 in
C     hdf.inc.
      vdata_id = vsfatch(file_id, -1, 'w')
      status = vsfsnam(vdata_id, 'PLIST')
      status = vsfscls(vdata_id, 'Connectivity List')
      status = vsffdef(vdata_id, 'PLIST', 24, 3)
      status = vsfsfld(vdata_id, 'PLIST')
      num_of_elements = vsfwrit(vdata_id, mesh, 20, FULL_INTERLACE)
      vg_position = vfinsrt(vgroup_id, vdata_id)
      status = vsfdtch(vdata_id)

C     Terminate access to the vgroup and the file.
      status = vfdtch(vgroup_id)
      status =  vfend(file_id)

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

      end
```

EXAMPLE 3.

Adding a Scientific Data Set to a Vgroup

The following examples create an HDF file, an SDS and a vgroup. Then they insert the SDS into the vgroup.

C:

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

main()
{
   int ret;
   int32 fid, sds_id, sref;
   int32 dims[1];
   int32 vfid, vg_id, status;

   dims[0] = 10;

   vfid = Hopen("Example3.hdf", DFACC_CREATE, 0);
   fid = SDstart("Example3.hdf", DFACC_RDWR);
   sds_id = SDcreate(fid, "Test_SD1", DFNT_INT8, 1, dims);

   status = Vstart(vfid);
   vg_id = Vattach(vfid, -1, "w");

   sref = SDidtoref(sds_id);
   status = Vaddtagref(vg_id, DFTAG_NDG, sref);

   status = SDendaccess(sds_id);
   status = SDend(fid);

   status = Vdetach(vg_id);
   status = Vend(vfid);
```

```
    status = Hclose(vfid);

}
```

FORTRAN:

```
      PROGRAM ADD SDS TO VGROUP

       integer status
       integer*4 fid, sds_id, sref, vfid, vg_id
       integer dims(1)
       integer*4 DFACC_CREATE, DFACC_RDWR, DFNT_INT8, DFTAG_NDG
       parameter (DFACC_CREATE = 4, DFACC_RDWR = 3, DFNT_INT8 = 20,
      +           DFTAG_NDG = 720)

       integer hopen, sfstart, sfcreate, vfstart, vfatch
       integer sfid2ref, vfadtr, sfendacc, vfdtch, vfend, hclose

       dims(1) = 10
       vfid = hopen('Example3.hdf', DFACC_CREATE, 0)
       fid = sfstart('Example3.hdf', DFACC_RDWR)
       sds_id = sfcreate(fid, 'Test_SD1', DFNT_INT8, 1, dims)
       status = vfstart(vfid)
       vg_id = vfatch(vfid, -1, 'w')

       sref = sfid2ref(sds_id)
       status = vfadtr(vg_id, DFTAG_NDG, sref)

       status = sfendacc(sds_id)

       status = vfdtch(vg_id)
       status = vfend(vfid)
       status = hclose(vfid)

       end
```

1.6 Reading from Vgroups

Reading from vgroups is more complicated than writing to vgroups. The process of reading from vgroups involves three steps:

1. Locate the appropriate vgroup.
2. Identify the members of the vgroup of interest.
3. Perform the read operation on the vgroup members.

1.6.1 Locating Vgroups in Files

There are several routines provided for the purpose of searching for a particular vgroup. The syntax for these routines are as follows.

C:

```
num_of_lones = Vlone(file_id, ref_array, maxsize);
ref_num = Vgetid(file_id, vgroup_ref);
status = Vgetname(vgroup_id, vgroup_name);
status = Vgetclass(vgroup_id, vgroup_class);
```

FORTRAN:

```
num_of_lones = vflone(file_id, ref_array, maxsize)
ref_num = vfgetid(file_id, vgroup_ref)
status = vfgfnam(vgroup_id, vgroup_name)
status = vfgcls(vgroup_id, vgroup_class)
```

1.6.1.1 Locating a Lone Vgroup: Vlone

A ***lone vgroup*** is one that is not a member of another vgroup. **Vlone** returns an array of reference numbers for all lone vgroups in a file. **Vlone** is useful for locating unattached vgroups in a file or the vgroups at the top of a grouping hierarchy. The parameter `ref_array` is an array allocated to hold the reference numbers. The `maxsize` argument specifies the maximum size of `ref_array`. At most `maxsize` reference numbers will be returned in `ref_array`. The value returned from **Vlone** is the total number of vgroups that are not linked to any other vgroup in the file.

The space allocated for `ref_array` depends on how many lone vgroups are expected to be found. 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 **Vlone** with `maxsize` set to a small value, for example, `0` or `1`, then use the returned value to allocate memory for `ref_array`. This array is then to be passed to an argument of **Vlone**.

1.6.1.2 Determining the Next Vgroup Identifier: Vgetid

Vgetid sequentially searches through an HDF file to check vgroup reference numbers. It returns the reference number for the vgroup immediately following the vgroup with the reference number in the `vgroup_ref` parameter. To initiate a search, **Vgetid** may be called with `vgroup_ref` set to `-1`. Doing so returns the reference number of the first vgroup in the file. Any attempt to search past the last vgroup in a file will cause **Vgetid** to return a value of `-1`.

1.6.1.3 Determining a Vgroup's Name and Class: Vgetname and Vgetclass

Vgetname returns the name of the vgroup through the parameter `vgroup_name`. The maximum length of the name is defined by the macro `VGNAMELENMAX`.

Vgetclass returns the class of the vgroup through the parameter `vgroup_class`. The maximum length of the class name is defined by the macro `VGNAMELENMAX`.

TABLE 5D **Vlone, Vgetid, Vgetname and Vgetclass Parameter List**

Routine Name (Fortran-77)	Parameter	Data Type		Description
		C	Fortran-77	
Vlone **(vflone)**	file_id	int32	integer	File identifier.
	ref_array	int32	integer	Buffer for the reference numbers of lone vgroups.
	maxsize	int32	integer	Maximum number of vgroups to store in `ref_array`.
Vgetid **(vfgid)**	file_id	int32	integer	File identifier.
	vgroup_ref	int32	integer	Reference number of the previous vgroup.
Vgetname **(vfgnam)**	vgroup_id	int32	integer	Vgroup identifier.
	vgroup_name	char *	character* (*)	Buffer for the name of the vgroup.
Vgetclass **(vfgcls)**	vgroup_id	int32	integer	Vgroup identifier.
	vgroup_class	char *	character* (*)	Buffer for the vgroup class. (if any)

EXAMPLE 4. **Printing Vgroup Reference Numbers**

These examples obtain and print the reference numbers for all of the lone vgroups and the vdata objects in an HDF file.

C:

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

```
main( )
{

int32 file_id, status;
int32 vgroup_ref, vgroup_id;
int32 maxsize, i, num_of_lones;
int32 n_entries, *ref_array;
char vgroup_name[VGNAMELENMAX];

/* Open the "Example2.hdf" file. */
file_id = Hopen("Example2.hdf", DFACC_READ, 0);

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

/* Get and print the reference numbers of all the lone
vgroups.  First, call Vlone with maxsize set to 0 to
get the length of the storage array, then call Vlone
again to put the reference id numbers into the array. */
maxsize = Vlone(file_id, ref_array, 0);
ref_array = (int32 *) HDmalloc(sizeof(int32) * maxsize);
num_of_lones = Vlone(file_id, ref_array, maxsize);

for (i = 0; i < maxsize; i++)
   printf("Lone vgroup reference id  %d\n", ref_array[i]);

printf("*******\n");
HDfree(ref_array);

/* Set the reference number variable to start the search
at the first vgroup in the file. */
vgroup_ref = -1;

/* Print every reference id in the file. */
while (TRUE) {
   vgroup_ref = Vgetid(file_id, vgroup_ref);
   if (vgroup_ref == -1) break;
   vgroup_id = Vattach(file_id, vgroup_ref, "r");
   status = Vinquire(vgroup_id, &n_entries, vgroup_name);
       printf("Found vgroup with ref %d, number of entries %d, name
%s\n",
              vgroup_ref, n_entries, vgroup_name);
       Vdetach(vgroup_id);
}

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

}
```

FORTRAN:

```
     PROGRAM PRINT REFS

      integer*4 file_id, ref_array(31)
      integer vgroup_ref, vgroup_id, status
      integer n_entries, i
      character vgroup_name(64)
      logical loop_flag
      integer hopen, vflone, vfgid, vfatch, vfdtch
      integer vfinq, hclose, vfstart, vfend
```

```
      integer*4 DFACC_READ
      parameter (DFACC_READ = 1)

C     Open an HDF file with read access.
C     DFACC_READ is defined in hdf.inc.
      file_id = hopen('Example2.hdf', DFACC_READ, 0)

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

C     Get and print the reference numbers of all the lone
C     vgroups.
      status = vflone(file_id, ref_array, 30)
      do 10 i = 1, status
         print *, 'Lone vgroup reference id ', ref_array(i)
10    continue

      print *, '*******'

C     Set the reference number variable to start the search
C     at the first vgroup in the file.
      vgroup_ref = -1

C     Print every reference id in the file.
      loop_flag = .TRUE.
      if (loop_flag) then
20       vgroup_ref = vfgid(file_id, vgroup_ref)
        if (vgroup_ref .eq. -1) then
          go to 30
        end if
        vgroup_id = vfatch(file_id, vgroup_ref, 'r')
        status = vfinq(vgroup_id, n_entries, vgroup_name)
        print *, 'Found vgroup with ref ', vgroup_ref,
     +           ' number of entries ', n_entries,
     +           ' name ', vgroup_name
        status = vfdtch(vgroup_id)
        goto 20
      end if

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

      end
```

1.7 Vgroup Attributes

Version 4.1r1 of HDF includes the capability of assigning attributes to a vgroup. 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 two 32-bit integers.

Any number of attributes can be assigned to a vgroup - however, each attribute name must be unique among all attributes in the vgroup.

1.7.1 Querying Information on a Given Vgroup Attribute: Vattrinfo

Vattrinfo returns the name, data type, number of values, and the size of the values of the specified attribute of the specified vgroup.

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 value of *attr_index* parameter is used as the index of the target vgroup attribute, and is zero-based. For example, a *attr_index* value of `4` would refer to the fifth attribute of the vgroup.

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

TABLE 5E **Vattrinfo Parameter List**

Routine Name (Fortran-77)	Parameter	Data Type		Description
		C	Fortran-77	
Vattrinfo (vfainfo)	vgroup_id	int32	integer	Vgroup identifier.
	attr_index	intn	integer	Index of the target attribute..
	name	char *	character* (*)	Returned name of the target attribute.
	data_type	int32 *	integer	Returned data type of the target attribute.
	count	int32 *	integer	Returned number of values of the target attribute.
	size	int32 *	integer	Returned size, in bytes, of the values of the target attribute.

1.7.2 Querying the Vgroup Version of a Given Vgroup: Vgetversion

Vgetversion returns the vgroup version number of the specified vgroup. The structure of the vgroup has gone through several changes since HDF was first written. Determining the version of any particular vgroup is necessary as some of the older versions of vgroups don't support attributes.

The newest version is version 4, which is the only version that supports vgroup attributes. **Vgetversion** returns a value of `VSET_NEW_VERSION` when encountering a vgroup of this vgroup version. Version 3 is the vgroup version corresponding to all versions of the HDF library between 3.2 and 4.0 release 2. **Vgetversion** returns a value of `VSET_VERSION` when encountering a vgroup of this vgroup version. The third version is version 2. This vgroup version corresponds to all HDF library versions before version 3.2, and **Vgetversion** returns a value of `VSET_OLD_VERSION` when encountering a vgroup of this vgroup version.

Vgetversion returns the vset version number if successful, and `FAIL` otherwise.

TABLE 5F **Vgetversion Parameter List**

Routine Name (Fortran-77)	Parameter	Data Type		Description
		C	Fortran-77	
Vgetversion (vfgver)	vgroup_id	int32	integer	Vgroup identifier.

1.7.3 Querying the Total Number of Vgroup Attributes: Vnattrs

Vnattrs returns number of attributes assigned to this vgroup and its fields when successful, and `FAIL` otherwise.

TABLE 5G **Vnattrs Parameter List**

Routine Name (Fortran-77)	Parameter	Data Type		Description
		C	Fortran-77	
Vnattrs (vfnatts)	vgroup_id	int32	integer	Vgroup identifier.

1.7.4 Querying the Values of a Given Vgroup Attribute: Vgetattr

Vgetattr returns all of the values of the specified attribute of the specified vgroup.

The *attr_index* parameter is the ordinal, zero-based index of the target attribute.

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

TABLE 5H **Vgetattr Parameter List**

Routine Name (Fortran-77)	Parameter	Data Type		Description
		C	Fortran-77	
Vgetattr (vsgnatt/vsgcatt)	vgroup_id	int32	integer	Vgroup identifier.
	attr_index	intn	integer	Index of the target attribute.
	values	VOIDP	<valid numeric data type>	Buffer containing attribute values.

1.7.5 Setting the Attribute of a Vgroup: Vsetattr

Vsetattr attaches an attribute to a vgroup. 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, **Vsetattr** will exit on an error condition.

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

TABLE 5I **Vsetattr Parameter List**

Routine Name (Fortran-77)	Parameter	Data Type		Description
		C	Fortran-77	
Vsetattr (vfsatt/vfscatt)	vgroup_id	int32	integer	Vgroup identifier.
	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	<valid numeric data type>	Buffer containing the attribute values.

1.7.6 Retrieving the Index of a Vgroup Attribute Given the Attribute Name:

Vfindattr

Vfindattr returns the index of an attribute with the given name, specified by the value of the *attr_name* parameter, of a vgroup.

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

TABLE 5J

Vfindattr Parameter List

Routine Name (Fortran-77)	Parameter	Data Type		Description
		C	Fortran-77	
Vfindattr (**vffdatt**)	vgroup_id	int32	integer	Vgroup identifier.
	attr_name	char *	character* (*)	Name of the target attribute.

EXAMPLE 5.

Attaching and Querying Vgroup Attributes

These examples attach an attribute to the vgroup created in Example 1 (in the file named "Example1.hdf"). Then it checks the number of attributes in the file, the version of the vgroup and then reads the attribute data.

C:

```
#include "hdf.h"

#define FILE_NAME "Example1.hdf"
#define VGATTR_NAME "Vgroup Attribute 1"

main( )
{

int32 file_id, vgroup_ref, vgroup_id, status;
int32 vg_version;
int32 v_type, v_count, v_size;
char vg_attr[6] = {'m','N','p', 'S', 't', '\0'};
char vgattr_buf[6], vattrname[30];

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

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

/* Get the reference number of the target vgroup. */
vgroup_ref = Vfind(file_id, "VG_Name_1");

/* Attach to the target vgroup. */
vgroup_id = Vattach(file_id, vgroup_ref, "w");

/* Get the version of the vgroup just created. */
vg_version = Vgetversion(vgroup_id);

/* Attach an attribute to the vgroup. */
status = Vsetattr(vgroup_id, VGATTR_NAME, DFNT_CHAR,
            6, vg_attr);

/* Get information about the vgroup attribute. */
status = Vattrinfo(vgroup_id, 0, vattrname, &v_type,
            &v_count, &v_size);
```

```
/* Get the vgroup attribute. */
status = Vgetattr(vgroup_id, 0, vgattr_buf);

/* Detach from the vgroup, close the V interface and the file. */
status = Vdetach(vgroup_id);
status = Vend(file_id);
status = Hclose(file_id);

}
```

FORTRAN:

```
         PROGRAM CREATE QUERY ATTRS

      integer file_id, vgroup_ref, vgroup_id, status
      integer vg_version
      integer hopen, vfstart, vfind, vfatch, vfdtch, hclose
      integer vfend, vsgver, vfgcatt, vfainfo, vfscatt
      integer v_type, v_count, v_size
      character*20 vgattrname, vgattr_buf
      character* (*) filename, vgattr_name, vg_attr
      parameter (filename = 'Example1.hdf',
     +   vgattr_name = 'Vgroup Attribute 1',
     +   vg_attr = 'mnpst'
     +    )

C     The following parameters are defined in hdf.inc.
      integer DFACC_RDWR, DFNT_CHAR
      parameter (DFACC_RDWR = 3, DFNT_CHAR = 4)

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

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

C     Get the reference number of the target vgroup.
      vgroup_ref = vfind(file_id, 'VG_Name_1')

C     Attach to the target vdata.
      vgroup_id = vfatch(file_id, vgroup_ref, 'w')

C     Get the version of the vgroup just created.
      vg_version = vsgver(vgroup_id)

C     Attach an attribute to the vgroup.
      status = vfscatt(vgroup_id, vgattr_name, DFNT_CHAR, 5, vg_attr)

C     Get information about the vgroup attribute.
      status = vfainfo(vgroup_id, 0, vgattrname, v_type, v_count,
     +                 v_size)

C     Get the vgroup attribute.
      status = vfgcatt(vgroup_id, 0, vgattr_buf)

C     Detach from the vgroup, close the V interface and the file.
      status = vfdtch(vgroup_id)
      status = vfend(file_id)
      status = hclose(file_id)

      end
```

1.8 Obtaining Information About the Contents of a Vgroup

When the target vgroup has been located, a way to determine its internal structure is needed. The HDF routines necessary to perform this operation are listed here, along with their syntax:

C:
```
num_of_elems = Vntagrefs(vgroup_id);
status = Vgettagref(vgroup_id, index, tag, ref);
num_of_pairs = Vgettagrefs(vgroup_id, tag_array, ref_array, max-
                                size);
true_false = Vinqtagref(vgroup_id, tag, ref);
true_false = Visvg(vgroup_id, vgroup_ref);
true_false = Visvs(vgroup_id, vdata_ref);
```

FORTRAN:
```
num_of_elems = vfntr(vgroup_id)
status = vfgttr(vgroup_id, index, tag, ref)
num_of_pairs = vfgttrs(vgroup_id, tag_array, ref_array, maxsize)
true_false = vfinqtr(vgroup_id, tag, ref)
true_false = vfisvg(vgroup_id, vgroup_ref)
true_false = vfisvs(vgroup_id, vdata_ref)
```

Each routine queries a specific item of vgroup information. The parameters of these routines are described below. (See Table 5K on page 164.)

1.8.1 Returning the Tag/Reference Number Pairs of Vgroup Members: Vntagrefs, Vgettagref and Vgettagrefs

Vntagrefs returns the number of members stored in the specified vgroup. This routine is used together with **Vgettagrefs** or in a loop with **Vgettagref** to identify the HDF objects linked to a given vgroup.

Vgettagref returns the tag/reference number pair of a specified HDF object stored within the vgroup in the `tag` and `ref` parameters respectively. The parameter `index` specifies the location of the member within the vgroup and is zero-based.

Vgettagrefs returns the tags/reference number pairs of the vgroup members in arrays. The tag/reference number pairs are returned in the parameters `tag_array` and `ref_array`. The parameter `maxsize` specifies the maximum number of tag/reference number pairs to return, therefore each array must be at least `maxsize` in size.

1.8.2 Returning Vgroup Member Information: Vinqtagref, Visvg and Visvs

Vinqtagref returns `TRUE` (or `1`) if the data object with tag/reference number pair specified by the `tag` and `ref` parameters are a member of the vgroup and `FALSE` (or `0`) otherwise.

Visvg returns `TRUE` (or `1`) if the vgroup reference number specified by the `vgroup_ref` parameter is that of a vgroup stored in the vgroup, and `FALSE` (or `0`) otherwise.

Visvs returns `TRUE` (or `1`) if the vdata reference number specified by the `vdata_ref` parameter is that of a vdata stored in the vgroup, and `FALSE`(or `0`) otherwise.

TABLE 5K

Vgetname, Vgetclass, Visvg and Visvs Parameter List

Routine Name (Fortran-77)	Parameter	Data Type		Description
		C	Fortran-77	
Vntagrefs (vfntr)	vgroup_id	int32	integer	Vgroup identifier.
Vgettagref (vfgttr)	vgroup_id	int32	integer	Vgroup identifier.
	index	int32	integer	Index of the tag/reference number pair tobe retrieved.
	tag	int32	integer	Pointer to the tag.
	ref	int32	integer	Pointer to the reference number.
Vgettagrefs (vfgttrs)	vgroup_id	int32	integer	Vgroup identifier.
	tag_array	int32 *	integer	Buffer for the returned tags.
	ref_array	int32 *	integer	Buffer for the returned reference numbers.
	maxsize	int32	integer	Maximum number of tag/reference numbrr pairs to be returned.
Vinqtagref (vfinqtr)	vgroup_id	int32	integer	Vgroup identifier.
	tag	int32	interger	Tag of item to be queried.
	ref	int32	integer	Reference number of items to be queried.
Visvg (vfisvg)	vgroup_id	int32	integer	Vgroup identifier.
	vgroup_ref	int32	integer	Vgroup reference number to be queried.
Visvs (vfisvs)	vgroup_id	int32	integer	Vgroup identifier.
	vdata_ref	int32	integer	Vdata reference number to be queried.

EXAMPLE 6.

Extracting the Tag/Reference Number Pairs of Vgroup Members

The following examples make use of **Vntagrefs** to determine the number of tag/reference number pairs assigned to the members of a vgroup. It then uses **Vgettagref** to extract each tag/reference number pair.

C:

```
#include "hdf.h"

main( )
{
int32 file_id;
int32 vgroup_id, vgroup_ref;
int32 vdata_tag, vdata_ref;
int32 status, i, npairs;

/* Open the "Example2.hdf" file. */
file_id = Hopen("Example2.hdf", DFACC_READ, 0);

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

/* Attach to every vgroup in the file. */
vgroup_ref = -1;

while (TRUE) {
   vgroup_ref = Vgetid(file_id, vgroup_ref);
   if (vgroup_ref == -1) break;
   vgroup_id = Vattach(file_id, vgroup_ref, "r");

   /* Get the total number of tag/reference id pairs. */
   npairs = Vntagrefs(vgroup_id);
```

```
    /* Print every tag and reference id with their
    corresponding file position. */
    for (i = 0; i < npairs; i++) {
       status = Vgettagref(vgroup_id, i, &vdata_tag, &vdata_ref);
       printf("Found tag = %d, ref = %d at position %d.\n", \
               vdata_tag, vdata_ref, i+1);
    }

    /* Terminate access to the vgroup. */
    status = Vdetach(vgroup_id);
 }

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

 }
```

FORTRAN:

```
      PROGRAM EXTRACT TAG

       integer*4 file_id, vgroup_id, vgroup_ref, vdata_tag
       integer*4 vdata_ref, npairs
       integer i, status
       integer hopen, vfatch, vfntr, vfgttr
       integer vfgid, vfdtch, hclose, vfstart, vfend
       integer*4 DFACC_READ
       parameter (DFACC_READ = 1)

C      Open an HDF file with read-only access.
C      DFACC_READ is defined in hdf.inc.
       file_id = hopen('Example2.hdf', DFACC_READ, 0)

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

C      Attach to every vgroup in the file.
       vgroup_ref = -1
30     vgroup_ref = vfgid(file_id, vgroup_ref)

       if (vgroup_ref .eq. -1) then
         go to 20
       else
         vgroup_id = vfatch(file_id, vgroup_ref, 'r')
C        Get the total number of tag/reference id pairs.
         npairs = vfntr(vgroup_id)
C        Print every tag and reference id with their
C        corresponding file position.
         do 10 i = 1, npairs
           status = vfgttr(vgroup_id, i-1, vdata_tag, vdata_ref)
           if (status .eq. -1) then
             go to 40
           end if
           print *, 'Found tag = ', vdata_tag, ' ref = ', vdata_ref,
      +              ' at position ', i

10       continue

C        Terminate access to the vgroup.
40       status = vfdtch(vgroup_id)
       end if
```

```
      go to 30

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

      end
```

1.9 The Vgroup Command-Line Utilities

The **vshow** and **vmake** utilities are Unix command line utilities for working with vgroups and vdatas. We give a very brief description of these here. For more information, refer to Chapter 13, titled *HDF Command-Line Utilities*.

1.9.1 Vshow

The **vshow** utility lists information about vgroups and vdatas in HDF files. It lists and displays information on all the vgroups in the file in the order stored, followed by the vdatas.

1.9.2 Vmake

The **vmake** utility is a general utility for generating vsets in an HDF file. **Vmake** is used to create new vgroups, store data into new vdatas and create links among vgroups and vdatas.

1.10 Obsolete Vgroup Interface Routines

The following routines have been replaced by newer routines with similar functionality. They are still supported by the HDF V interface, but their use is not recommended. HDF may not support these routines in a future version.

1.10.1 Determining the Next Vgroup or Vdata Identifier: Vgetnext

Vgetnext searches through a vgroup for vgroups or vdatas. The syntax for **Vgetnext** is:

C: `ref_num = Vgetnext(vgroup_id, vgroup_ref);`

FORTRAN: `ref_num = vfgnxt(vgroup_id, vgroup_ref)`

Vgetnext searches the vgroup with the identifier `vgroup_id` and returns the reference number of the vgroup or vdata following the vgroup with the reference number specified by the `vgroup_ref` parameter. To initiate the search, **Vgetnext** is called with `vgroup_ref` set to `-1`. This will return the reference number of the first object in the target vgroup. The value `-1` is returned when an error occurs or when there are no more entities in the host vgroup.

Vgetnext is now obsolete as the routine **Vgettagref** provides the same functionality and, in addition, is not restricted to searching for members that are vgroups or vdatas.

1.10.2 Determining the Number of Members and Vgroup Name: Vinquire

The syntax for **Vinquire** is:

C:
```
status = Vinquire(vgroup_id, n_members, vgroup_name);
```

FORTRAN:
```
status = vfinq(vgroup_id, n_members, vgroup_name)
```

Vinquire returns the number of data objects and the name of the vgroup specified by `vgroup_id` in the `n_members` and `vgroup_name` parameters respectively. If either `n_members` or `vgroup_name` are set to `NULL`, the corresponding data is not returned. The maximum length of the vgroup's name is defined by the macro `VGNAMELENMAX`.

Vinquire is now obsolete as the **Vntagrefs** routine can be used to get the number of objects in a vgroup and **Vgetname** can be used to retrieve the name of a vgroup.

TABLE 5L **Vgetnext and Vinquire Parameter List**

Routine Name (Fortran-77)	Parameter	Data Type		Description
		C	Fortran-77	
Vgetnext **(vfgnxt)**	vgroup_id	int32	integer	Vgroup identifier of the parent vgroup.
	vgroup_ref	int32	integer	Reference number for the target vgroup.
Vinquire **(vfinq)**	vgroup_id	int32	integer	Vgroup identifier.
	members	int32 *	integer	Pointer to the number of entries in the vgroup.
	vgroup_name	char *	character* (*)	Buffer for the name of the vgroup.

1.11 Vgroup Backward Compatibility Issues

1.11.1 Vset Implementation Integrated into the Vgroup Interface

In HDF versions before 2.0, vsets were created and manipulated through an interface separate from the other main HDF interfaces like the SDS interface, VS interface, etc.. A pointer to a specially-defined vset structure was returned by the vset interface and data type definitions specific to the vset interface was used. The names of these definitions were prefaced by "LOCAL_".

After HDF version 2.0 the `vs_id` identifier replaced the pointer to the vset structure, the "LOCAL_" data type definitions were made obsolete by newer definitions and all vset functionality was integrated in the V interface routines. HDF programs written for HDF libraries earlier than version 2.0 should be modified accordingly in order to function correctly with newer versions of HDF.