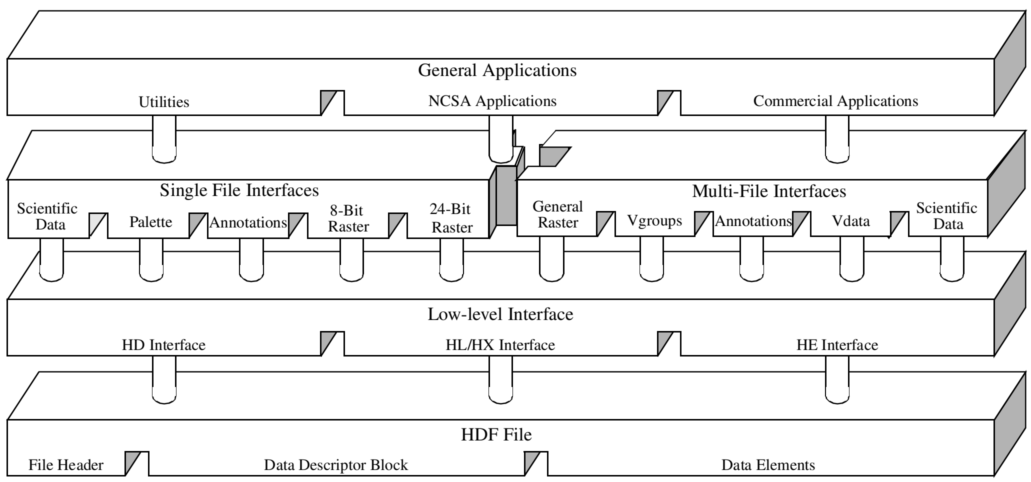
Section
1

Introduction to the HDF APIs

1.1 Overview of the HDF Interfaces

The HDF library structure consists of three interface layers built upon a physical file format. (See Figure 1a.) The first layer, or the *low-level interface*, is generally reserved for software developers because it provides support for low-level details such as file I/O, error handling, and memory management. The second layer, containing the single and multifile *application interfaces*, consists of a set of interfaces designed to simplify the process of storing and accessing data. The single file interfaces are operate on one file at a time, whereas the multifile interfaces can operate on several simultaneously. The highest HDF layer includes a collection of command-line *utilities* that operate on HDF files or the data objects they contain.

FIGURE 1a **The Three Levels of Interaction with the HDF File Format**



1.2 The Low-Level Interface

This is the layer of HDF reserved for software developers and provides routines for error handling, file I/O, memory management, and physical storage. For a more detailed discussion of the low-level interface, consult the *HDF Specification and Developer's Guide*.

1.2.1 The H Interface

The low-level H interface provides a collection of routines, whose names begin with the letter H, for managing HDF files.

Prior to HDF version 3.2, all low-level routines began with the prefix ’DF’. As of HDF version 3.3, the DF interface was no longer recommended for use. It is only supported to maintain backward compatibility with programs and files created under earlier versions of the HDF library.

1.2.2 The HDF Interface

The names of these routines are prefaced by ’HDF’. As HDF begins expanding its interfaces to include mult-file support for each data model, it is anticipated that the some H routines will evolve into an HDF interface. There are only two such routines, **HDFopen** and **HDFclose**. **HDFopen** and **HDFclose** currently operate as macros for **Hopen** and **Hclose** respectively.

1.2.3 The HE Interface

The HDF library incorporates an error stack via the HE interface. In addition to other functions, the HE routines add information onto the error stack, print information from the stack, and clear the stack.

1.3 Single-File Application Interfaces

The HDF single-file application interfaces include several independent modules each designed to simplify the process of storing and accessing a specific type of data. These interfaces support the 8-bit raster image(DFR8), 24-bit raster image (DF24), palette (DFP), scientific data (DFSD), and annotation (DFAN) models. All single- file interfaces are built upon the H routines - unless otherwise specified, all the low-level details can be ignored.

1.3.1 8-bit Raster Image Sets: The DFR8 Interface

The HDF 8-bit raster interface provides a collection of routines for managing 8-bit raster image sets. Any 8-bit raster image accompanied by its dimension record is recognized as an 8-bit raster image set. Raster image sets may also include a palette.

Every function in the 8-bit raster interface begins with the prefix ’DFR8’. The equivalent Fortran-77 functions use the prefix ’d8’.

1.3.2 Palettes: The DFP Interface

The HDF palette interface provides a collection of routines for managing palette data. This interface is most often used for working with multiple palettes stored in a single file or palettes not specifically assigned to a raster image.

The names of the routines in the palette interface are prefaced by ’DFP’. The equivalent Fortran-77 routine names are prefaced by ’dp’.

1.3.3 24-bit Raster Image Sets: The DF24 Interface

The HDF 24-bit raster interface provides a collection of routines for managing 24-bit raster image sets. Any 24-bit raster image array accompanied by its dimension record is recognized as a 24-bit raster image set.

The names of the routines in the 24-bit raster interface are prefaced by ’DF24’. The equivalent Fortran-77 routine names are prefaced by ’d2’.

1.3.4 Scientific Data Sets: The Single File DFSD Interface

There are two HDF interfaces that support multi-dimensional arrays: the single-file DFSD interface described here, which permits access to only one file at a time, and the newer multifile SD interface, which permits simultaneous access to more than one file.

The single-file scientific data set interface provides a collection of routines for reading and writing arrays of arbitrary rank and number type. Any array accompanied by a record of its rank and number type qualifies as a scientific data set. Scientific data sets may also include predefined attribute records.

The names of the routines in the single-file scientific data set interface are prefaced by ’DFSD’. The equivalent Fortran-77 routine names are prefaced by ’ds’.

1.3.5 Annotations: The DFAN Interface

The single-file annotation interface provides a collection of routines for reading and writing text strings assigned to HDF data objects or files. Annotations consist of labels and descriptions. A label is a null-terminated sequence of characters.

The names of the routines in the single-file annotation interface are prefaced by ’DFAN’. The equivalent Fortran-77 routine names are prefaced by ’da’.

1.4 Multi-File Application Interfaces

The HDF multifile interfaces are designed to allow operations on more than one file and more than one data object at the same time. The multifile interfaces provided are the AN, GR, SD, VS, VSQ, VF, V, and VH interfaces. The AN interface is the multifile version of the DFAN annotation interface. The GR interface is the multifile version of the 8- and 24-bit annotation interfaces. The SD interface is the multifile version of the scientific data set interface. The VS, VSQ, and VF interfaces support the vdata model and have always provided multiple file access. Similarly, the V and VH interfaces also provide multiple file access for the vgroup data model.

Like the single-file interfaces, the multifile interfaces are built upon the low-level H routines. Unlike single-file operations, operations performed via a multifile interface are not implicitly preceded by **Hopen** and followed by **Hclose**. Instead, each series of operations on a file must be preceded by an explicit call to open and close the file. Once the file is opened, it remains open until an explicit call is made to close it. This process allows for operations on more than one file at a time.

1.4.1 Scientific Data Sets: The SD Interface

The scientific data set interface provides a collection of routines for reading and writing arrays of arbitrary dimension and number type. Multidimensional arrays accompanied by a record of their dimension and number type are called scientific data sets. Under the multifile interface, scientific data sets may include predefined or user defined attribute records. Each attribute record is optional and describes a particular facet of the environment from which the scientific data was taken.

The names of the routines in the multifile scientific data set interface are prefaced by 'SD'. The equivalent Fortran-77 routine names are prefaced by 'sf'.

1.4.2 Multifile Annotations: The AN Interface

The purpose of the AN multifile annotation interface is to permit concurrent operations on a set of annotations that exist in more than one file. The design of the AN interface is similar to the multifile interfaces for raster image (GR) and scientific data set objects (SD).

The C routine names of the multifile annotation interface are prefaced by the string 'AN' and the Fortran-77 routine names are prefaced by 'af'.

1.4.3 General Raster Images: The GR Interface

The routines in the GR interface provide for multifile operations on general raster (GR) image data sets.

The C routine names in the general raster interface have the prefix 'GR' and the equivalent Fortran-77 routine names are prefaced by 'mg'.

1.4.4 Scientific Data Sets: The netCDF Interface

The SD interface is designed to be as interoperable as possible with netCDF, an interface developed by the Unidata Program Center. Consequently, the SD interface can read files written by the netCDF interface, and the netCDF interface (as implemented in HDF) can read both netCDF files and HDF files that contain scientific data sets.

Further information regarding the netCDF interface routines and their equivalents in the HDF interface can be found in the User's Guide. Additional information on the netCDF interface can be found in the netCDF User's Guide available by anonymous FTP from `unidata.ucar.edu`.

1.4.5 Vdata: The VS Interface

The VS interface provides a collection of routines for reading and writing customized tables. Each table is comprised of a series of vdata records whose values are stored in fixed length fields. In addition to its records, a vdata may contain three kinds of identifying information: a vdata name, vdata class, and several vdata field names.

Routines in the VS interface are prefaced by 'VS'. The equivalent Fortran-77 routine names are prefaced by 'vsf'.

1.4.6 Vdata Query: The VSQ Interface

The VSQ interface provides a collection of routines for inquiring about existing Vdata. These routines provide information such as the number of records in a Vdata, its field names, number types, and name. All routines in the VSQ interface are prefaced by 'VSQ'.

1.4.7 Vdata Fields: The VF Interface

The VF interface provides a collection of routines for inquiring about the fields in an existing Vdata. These routines provide information such as the field name, size, order, and number type.

All routines in the VF interface are prefaced by 'VF'. There are no equivalent Fortran-77 functions.

1.4.8 Vgroups: The V Interface

The vgroup interface provides a collection of routines for reading and writing customized data sets. Each vgroup may contain one or more vdatas, vgroups, or data objects stored via other HDF data models. In addition to its members, a vgroup may also be given a vgroup name and a vgroup class.

Every routine name in the vgroup interfaceare prefaced by 'V'. The equivalent Fortran-77 routine names are prefaced by 'vf'.

1.4.9 High-Level Vdata/Vgroups: The VH Interface

The high-level VH interface provides a collection of routines for creating simple vdatas and vgroups with a single function call. All routines in this interface are prefaced by 'VH'.

1.4.10 Vgroup Inquiry: The VQuery Interface

The high-level VQ interface provides one routine that returns tag information from a specified vgroup, and one routine that returns reference number information from a specified vgroup. All C routine names in this interface are prefaced by 'VQuery'.

1.5 Fortran-77 and C Language Issues

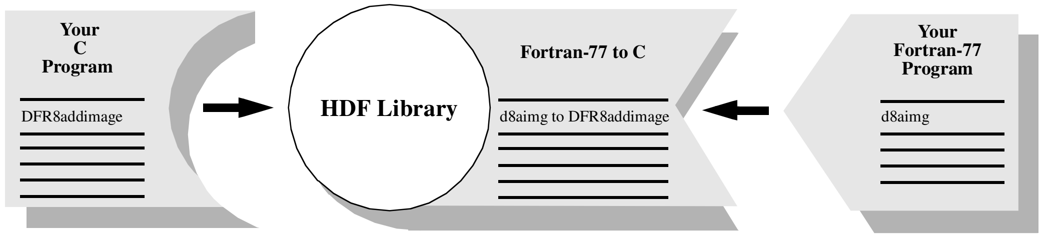
In order to make the Fortran-77 and C versions of each routine as similar as possible, some compromises have been made in the process of simplifying the interface for both programming languages.

1.5.1 Fortran-77-to-C Translation

Nearly all of the HDF library code is written in C. The Fortran-77 HDF API routines translate all parameter data types to C data types, then call the C routine that performs the main function. For example, **d8aimg** is the Fortran-77 equivalent for **DFR8addimage**. Calls to either routine execute the same C code that adds an 8-bit raster image to an HDF file - see the following figure.

FIGURE 2b

Use of a Function Call Converter to Route Fortran-77 HDF Calls to the C Library



1.5.2 Case Sensitivity

Fortran-77 identifiers generally are not case sensitive, whereas C identifiers are. Although all of the Fortran-77 routines shown in this manual are written in lower case, Fortran-77 programs can generally call them using either upper- or lower-case letters without loss of meaning.

1.5.3 Name Length

Because some Fortran-77 compilers only interpret identifier names with seven or fewer characters, the first seven characters of the Fortran-77 HDF routine names are unique.

1.5.4 Header Files

The inclusion of header files is not generally permitted by Fortran-77 compilers. However, it is sometimes available as an option. On UNIX systems, for example, the macro processors `m4` and `cpp` let your compiler include and preprocess header files. If this capability is not available, you may have to copy whatever declarations, definitions, or values you need from the “constants.f” file into your program code. If it is, include the header file named “hdf.inc” in your Fortran-77 code. The “constants.f” file is included in the “hdf.inc” header file.

1.5.5 Data Type Specifications

When mixing machines, compilers, and languages, it is difficult to maintain consistent data type definitions. For instance, on some machines an integer is a 32-bit quantity and on others, a 16-bit quantity. In addition, the differences between Fortran-77 and C lead to difficulties in describing the data types found in the argument lists of HDF routines. To maintain portability, the HDF library expects assigned names for all data types used in HDF routines. (See TABLE 2A)

TABLE 2A

Data Type Definitions

Data Type	C	Fortran-77
8-bit signed integer	`int8`	Not supported.
8-bit unsigned integer	`uint8`	`character*1`
16-bit signed integer	`int16`	`integer*2`
16-bit unsigned integer	`uint16`	Not supported.
32-bit signed integer	`int32`	`integer*4`
32-bit unsigned integer	`uint32`	Not supported.
32-bit floating point number	`float32`	`real*4`
64-bit floating point number	`float64`	`real*8`

Data Type	C	Fortran-77
Native signed integer	`intn`	`integer`
Native unsigned integer	`uintn`	Not supported.

When using a Fortran-77 data type that is not supported, the general practice is to use another data type of the same size. For example, an 8-bit signed integer can be used to store an 8-bit unsigned integer variable unless the code relies on a sign-specific operation.

1.5.6 Array Specifications

In the declarations contained in the headers of Fortran-77 functions, the following conventions are followed:

- `character* x(*)` means that `x` refers to an array that contains an indefinite number of characters. It is the responsibility of the calling program to allocate enough space to hold whatever data is stored in the array.
- `real* x(*)` means that `x` refers to an array of reals of indefinite size and of indefinite rank. It is the responsibility of the calling program to allocate an actual array with the correct number of dimensions and dimension sizes.

1.5.7 Fortran-77, ANSI C and K&R C

As much as possible, we have conformed the HDF API routines to those implementations of Fortran and C that are in most common use today, namely Fortran-77, ANSI C and K&R C. Due to the increasing availability of ANSI C, future versions of HDF will no longer support K&R C.

As Fortran-90 is a superset of Fortran-77, HDF programs should compile and run correctly when using a Fortran-90 compiler.

1.6 Error Codes

The error codes defined in the HDF library are defined in the following table.

TABLE 1B **HDF Error Codes**

Error Code	Code Definition
DFE_NONE	No error.
DFE_FNF	File not found.
DFE_DENIED	Access to file denied.
DFE_ALROPEN	File already open.
DFE_TOOMANY	Too many AID's or files open.
DFE_BADNAME	Bad file name on open.
DFE_BADACC	Bad file access mode.
DFE_BADOPEN	Miscellaneous open error.
DFE_NOTOPEN	File can't be closed because it hasn't been opened.
DFE_CANTCLOSE	`fclose` error
DFE_READERROR	Read error.
DFE_WRITEERROR	Write error.
DFE_SEEKERROR	Seek error.

Error Code	Code Definition
DFE_RDONLY	File is read only.
DFE_BADSEEK	Attempt to seek past end of element.
DFE_PUTELEM	`Hputelement` error.
DFE_GETELEM	`Hgetelement` error.
DFE_CANTLINK	Cannot initialize link information.
DFE_CANTSYNC	Cannot syncronize memory with file.
DFE_BADGROUP	Error from `DFdiread` in opening a group.
DFE_GROUPSETUP	Error from `DFdisetup` in opening a group.
DFE_PUTGROUP	Error on putting a tag/reference number pair into a group.
DFE_GROUPWRITE	Error when writing group contents.
DFE_DFNULL	Data file reference is a null pointer.
DFE_ILLTYPE	Data file contains an illegal type: internal error.
DFE_BADDDLIST	The DD list is non-existent: internal error.
DFE_NOTDFFILE	The current file is not an HDF file and it is not zero length.
DFE_SEEDTWICE	The DD list already seeded: internal error.
DFE_NOSUCHTAG	No such tag in the file: search failed.
DFE_NOFREEDD	There are no free DD's left: internal error.
DFE_BADTAG	Illegal WILDCARD tag.
DFE_BADREF	Illegal WILDCARD reference number.
DFE_NOMATCH	No DDs (or no more DDs) that match the specified tag/reference number pair.
DFE_NOTINSET	Warning: Set contained unknown tag. Ignored.
DFE_BADOFFSET	Illegal offset specified.
DFE_CORRUPT	File is corrupted.
DFE_NOREF	No more reference numbers are available.
DFE_DUPDD	The new tag/reference number pair has been allocated.
DFE_CANTMOD	Old element doesn't exisr. Cannot modify.
DFE_DIFFFILES	Attempt to merge objects in different files.
DFE_BADAID	An invalid AID was received.
DFE_OPENAID	Active AIDs still exist.
DFE_CANTFLUSH	Cannot flush DD back to file.
DFE_CANTUPDATE	Cannot update the DD block.
DFE_CANTHASH	Cannot add a DD to the hash table.
DFE_CANTDELDD	Cannot delete a DD in the file.
DFE_CANTDELHASH	Cannot delete a DD from the hash table.
DFE_CANTACCESS	Cannot access specified tag/reference number pair.
DFE_CANTENDACCESS	Cannot end access to data element.
DFE_TABLEFULL	Access table is full.
DFE_NOTINTABLE	Cannot find element in table.
DFE_UNSUPPORTED	Feature not currently supported.
DFE_NOSPACE	`malloc` failed.
DFE_BADCALL	Routine calls were in the wrong order.
DFE_BADPTR	NULL pointer argument was specified.
DFE_BADLEN	Invalid length was specified.
DFE_NOTENOUGH	Not enought space for the data.
DFE_NOVALS	Values were not available.
DFE_ARGS	Invalid arguments passed to the routine.
DFE_INTERNAL	Serious internal error.

Error Code	Code Definition
DFE_NORESET	Too late to modify this value.
DFE_GENAPP	Generic application level error.
DFE_UNINIT	Interface was not initialized correctly.
DFE_CANTINIT	Cannot initialize the interface the operation requires.
DFE_CANTSHUTDOWN	Cannot shut down the interface the operation requires.
DFE_BADDIM	Negative number of dimensions, or zero dimensions, was specified.
DFE_BADFP	File contained an illegal floating point number.
DFE_BADDATATYPE	Unknown or unavailable data type was specified.
DFE_BADMCTYPE	Unknown or unavailable machine type was specified.
DFE_BADNUMTYPE	Unknown or unavailable number type was specified.
DFE_BADORDER	Unknown or illegal array order was specified.
DFE_RANGE	Improper range for attempted access.
DFE_BADCONV	Invalid data type conversion was specified..
DFE_BADTYPE	Incompatible types were specified.
DFE_BADSCHEME	Unknown compression scheme was specified.
DFE_BADMODEL	Invalid compression model was specified.
DFE_BADCODER	Invalid compression encoder was specified.
DFE_MODEL	Error in the modeling layer of the compression operation.
DFE_CODER	Error in the encoding layer of the compression operation.
DFE_CINIT	Error in encoding initialization.
DFE_CDECODE	Error in decoding compressed data.
DFE_CENCODE	Error in encoding compressed data.
DFE_CTERM	Error in encoding termination.
DFE_CSEEK	Error seeking in an encoded dataset.
DFE_MINIT	Error in modeling initialization.
DFE_COMPINFO	Invalid compression header.
DFE_CANTCOMP	Cannot compress an object.
DFE_CANTDECOMP	Cannot decompress an object.
DFE_NODIM	A dimension record was not associated with the image.
DFE_BADRIG	Error processing a RIG.
DFE_RINOTFOUND	Cannot find raster image.
DFE_BADATTR	Invalide attribute.
DFE_BADTABLE	The nsdg table has incorrect information.
DFE_BADSDG	Error in processing an SDG.
DFE_BADNDG	Error in processing an NDG.
DFE_VGSIZE	Too many elements in the vgroup.
DFE_VTAB	Element not in `vtab[]`.
DFE_CANTADDELEM	Cannot add the tag/reference number pair to the vgroup.
DFE_BADVGNAME	Cannot set the vgroup name.
DFE_BADVGCLASS	Cannot set the vgroup class.
DFE_BADFIELDS	Invalid fields string passed to vset routine.
DFE_NOVS	Cannot find the vset in the file.
DFE_SYMSIZE	Too many symbols in the users table.
DFE_BADATTACH	Cannot write to a previously attached vdata.
DFE_BADVSNAME	Cannot set the vdata name.
DFE_BADVSCLASS	Cannot set the vdata class.
DFE_VSWRITE	Error writing to the vdata.

Error Code	Code Definition
DFE_VSREAD	Error reading from the vdata.
DFE_BADVH	Error in the vdata header.
DFE_VSCANTCREATE	Cannot create the vdata.
DFE_VGCANTCREATE	Cannot create the vgroup.
DFE_CANTATTACH	Cannot attach to a vdata or vset.
DFE_CANTDETACH	Cannot detach a vdata or vset with write access.
DFE_BITREAD	A bit read error occurred.
DFE_BITWRITE	A bit write error occurred.
DFE_BITSEEK	A bit seek error occurred.
DFE_TBBTINS	Failed to insert the element into tree.
DFE_BVNEW	Failed to create a bit vector.
DFE_BVSET	Failed when setting a bit in a bit vector.
DFE_BVGET	Failed when getting a bit in a bit vector.
DFE_BVFIND	Failed when finding a bit in a bit vector.