Chapter 2

# HDF Fundamentals

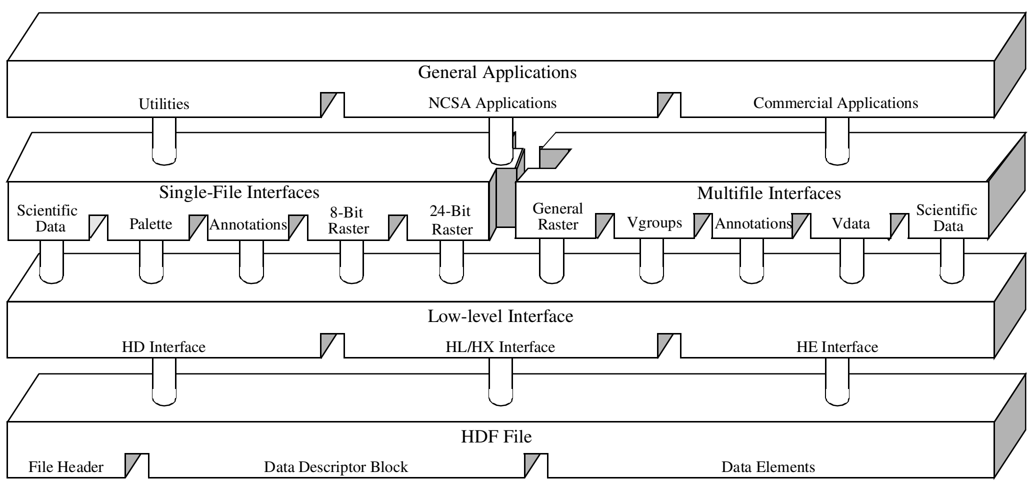
## 2.1 Chapter Overview

In this chapter, our description of HDF in Chapter 1, titled *Introduction to HDF*, is expanded to include a description of the hierarchical structure of HDF interaction with HDF file objects, the physical format of HDF files, the low-level HDF interfaces, and programming language issues pertaining to the use of Fortran-77, ANSI C and K&R C in HDF programming.

## 2.2 The Hierarchy of HDF Interaction

To review the description of HDF provided in Chapter 1, HDF is a physical file format at its lowest level and a collection of utilities and applications at its highest. Between these two levels, HDF is a library that itself provides two levels of programming interfaces. HDF can be thought of conceptually as three interface layers built upon a physical file format.

FIGURE 2a **The Three Levels of Interaction with the HDF File Format**




Refer to Figure 2a. Of the three types of top-level general applications, only the command-line utilities will be extensively covered in this manual. See Chapter 13, titled *HDF Command-Line Utilities*, for descriptions of this aspect of HDF. These general applications directly call the single-file and multifile interfaces.

The two interactive levels immediately below this level, the ***low-level interface*** and the HDF data file itself, are only briefly described as the single-file and multifile interfaces provide a safer and more standardized means of accessing these levels. The single-file and multifile interfaces - the second highest level of interaction within HDF - are routinely updated as aspects of these lower-level interfaces are changed, in a manner as transparent to the HDF user as possible. With the exception of the few instances where lower-level interface functionality has not yet been incorporated into the higher-level interface functions, the HDF user need not directly concern themselves with these levels.

## 2.3 Data Objects

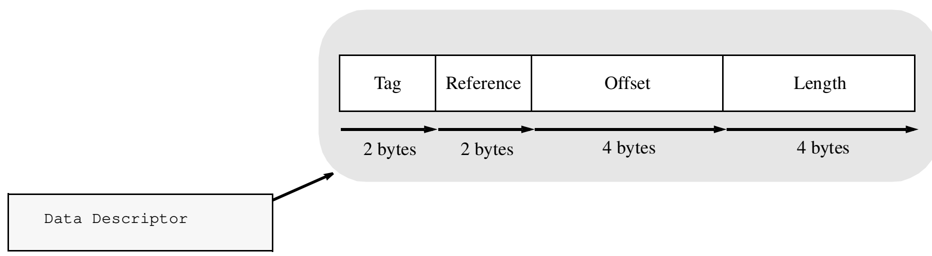
The term ***data object*** is used to describe the fundamental conglomerate structure use to encapsulate data.

Data objects contain a ***data descriptor*** and a ***data element***. Data descriptors consist of information about the type, location, and size of a data element. Data elements contain the primary data itself.

### 2.3.1 Data Descriptors

All data descriptors are twelve bytes long and contain four fields. (See Figure 2b.)

FIGURE 2b **The Contents of a Data Descriptor**




#### Tags

Tags identify the type of data stored in its data element. For example, a raster image descriptor is identified by a `DFTAG_RI` tag and a palette descriptor a `DFTAG_LUT` tag. There are currently over 200 tags defined for general use. A complete list of tags and their descriptions can be found in Appendix A of this manual

Tag values ranging between 1 and 32,767 are reserved for commonly-used data types. These tags are assigned by the HDF development group. Tag values ranging between 32,768 and 64,999 are not regulated by NCSA and are available for private application. These tags are not documented by NCSA and may therefore conflict with tags assigned by someone else. Therefore, it is best to limit applications to the use of the commonly-used tags.

#### Reference Numbers

Reference numbers distinguish between different data elements with the same tag. For example, all raster image descriptors will have the same raster data tag. The pairing of a tag and a reference number provides a unique identifier for any HDF object within a file.

Although HDF assigns reference numbers in increasing order, it is the write operations that are counted, not the number or type of data objects added to the file. For example, writing a raster image set followed by a scientific data set uses a minimum of six data objects but only two write operations. Consequently, HDF will assign 1 as the reference number for the first raster image set and all its members and 2 as the reference number for the scientific data set and all its members.

While application programmers may find it convenient to impart some additional meaning to reference numbers, it should be noted that HDF will not internally recognize any such meaning.

**Offsets and Lengths**

The offset field points to the location of the data element in the file by storing the number of bytes from the beginning of the file to the beginning of the data element. The length field contains the total size of the data element in bytes.

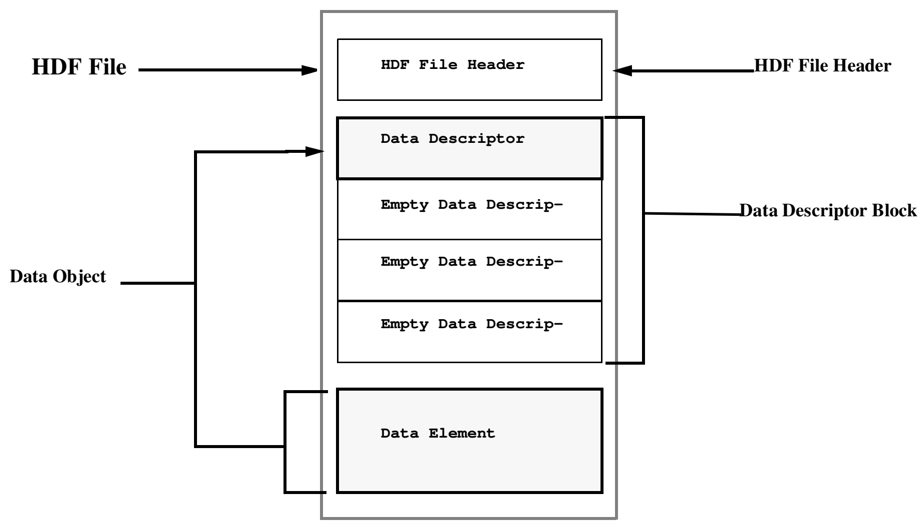
**Data Elements**

The type of the data stored in a data element is identified by its tag, however, other interpretive information may be required before it can be processed properly.

## 2.4 File Format

HDF files contain a ***file header*** and at least one ***data descriptor block***, as depicted in Figure 2c. The HDF file header occupies the first four bytes of every HDF file with a signature field containing the 32-bit hexadecimal value `0e031301`. This number is considered a "magic cookie" as it identifies the file as an HDF file. Initially the data descriptor block consists of a group of empty data descriptors. As data objects are written to the file, the HDF library fills the data descriptor block by pairing one data descriptor for each data element. Refer to the following figure.

FIGURE 2c **The Physical Layout of an HDF File Containing One Data Object**




### 2.4.1 Grouping Data Objects in an HDF File

HDF files that contain more than one data element are generally easier to work with when the data objects containing related data are grouped together. These groups of data objects are called ***data sets***. The HDF user uses the application interface to define, manipulate and dispose of data sets in a file.

As an example, an 8-bit raster image data set requires three objects: a group object identifying the members of the set, an image object containing the image data and a dimension object indicating the size of the image. It is sometimes possible to add additional data objects to the minimum set - for example, an 8-bit raster image set may include a palette.

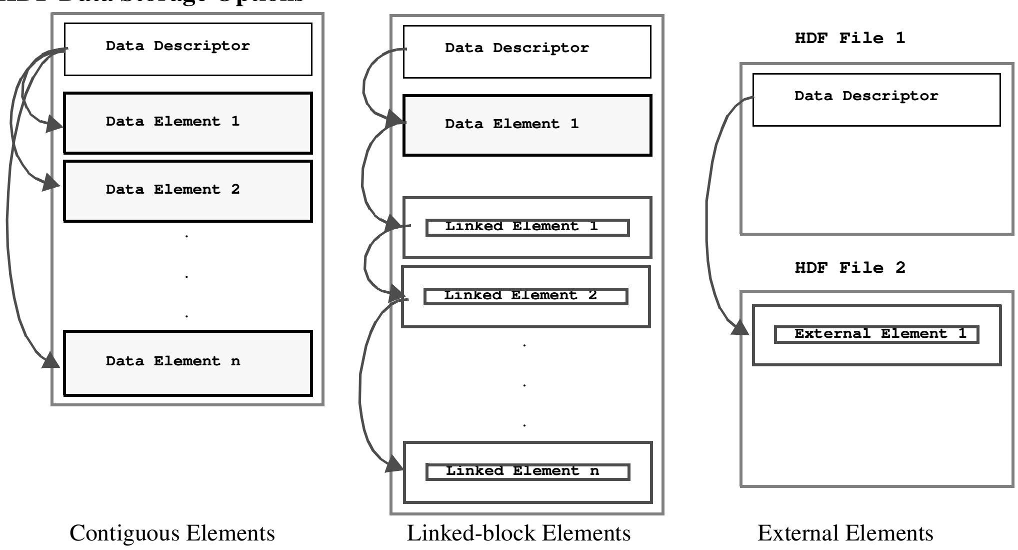
Data objects are individually accessible even if they are included in a set, therefore data objects can belong to more than one set and sets can be included in larger groups. For example, a palette object included in one raster image set may also be a part of another raster image set if its tag and reference number are included in a data descriptor within that second set.

Several data sets may be further grouped into ***group objects***. The contents of the group object depends on the HDF data set it supports.

### 2.4.2 Storing Data Objects

Data objects can be stored in HDF data files as a ***contiguous element***, as ***linked-block elements*** or as ***external elements***. These data storage options are illustrated below in Figure 2d.

FIGURE 2d **HDF Data Storage Options**




#### 2.4.2.1 Contiguous Data Elements

This is the default method of data storage in HDF. In this case, all data objects exist in one HDF file and are stored in the manner illustrated in Figure 2c. Each data element in the file is one complete unit of raw data, and as only one unique data descriptor points to it there are no references or relationships to the other data elements in the file. As additional data elements are created and written they are appended to the end of the HDF file.

#### 2.4.2.2 Linked-block Data Elements

Linked-block elements are a series of data elements existing in one HDF file which serve as a means of adding data to a pre-existing data element. These elements are linked to each other and to the original data element by a linked-list structure similar to the data descriptor list. They are of a uniform size with the exception of the first block after the original data element, which is the only block that can be resized.

#### 2.4.2.3 External Data Elements

External data elements are those that exist in a file apart from the one the data descriptor list resides in. These additional HDF data files are referred to as ***external data files***. The HDF library keeps track of the filesystem location of the external data file as well as the standard tag/reference number pair, offset and length information of the external element.

## 2.5 Header File Information

The "hdf.h" header file must be included in every HDF application program as it contains necessary declarations and definitions as well as prototypes for the HDF API routines. To use these routines, the HDF user must be familiar with the most-commonly used definitions stored in the "hdf.h" file.

### 2.5.1 File Access Code Definitions

These definitions are passed into API file access routines as parameters. The `DFACC_RDONLY` and `DFACC_CLOBBER` definitions exist in the "hdf.h" file to accommodate backward compatibility with applications designed to work with older versions of the HDF API library.

TABLE 2A **File Access Code Definitions**

| Definition Name | Definition Value | Description |
|---|---|---|
| **DFACC_READ** | 1 | Read access. |
| **DFACC_WRITE** | 2 | Write access. |
| **DFACC_RDWR** | 3 | Read and write access. |
| **DFACC_CREATE** | 4 | File creation access. |
| **DFACC_CLOBBER** | 4 | Same as DFACC_CREATE. |
| **DFACC_RDONLY** | 1 | Same as DFACC_READ. |

### 2.5.2 Data Type Definitions

These definitions are used in comparison expressions to determine the type of an input variable or a value returned by an API function. `DFNT_FLOAT`, `DFNT_DOUBLE`, `DFNT_UCHAR`, and `DFNT_CHAR` are included in the "hdf.h" file for backward compatibility. These definitions are located in the "hntdefs.h" header file.

TABLE 2B **Data Type Definitions**

| Definition Name | Definition Value | Description |
|---|---|---|
| **DFNT_CHAR8** | 4 | 8-bit character type. |
| **DFNT_CHAR** | 4 | Same as DFNT_CHAR8. |
| **DFNT_UCHAR8** | 3 | 8-bit unsigned character type. |
| **DFNT_UCHAR** | 3 | Same as DFNT_UCHAR8. |
| **DFNT_INT8** | 20 | 8-bit integer type. |
| **DFNT_UINT8** | 21 | 8-bit unsigned integer type. |
| **DFNT_INT16** | 22 | 16-bit integer type. |
| **DFNT_UINT16** | 23 | 16-bit unsigned integer type. |
| **DFNT_INT32** | 24 | 32-bit integer type. |
| **DFNT_UINT32** | 25 | 32-bit unsigned integer type. |
| **DFNT_INT64** | 26 | 64-bit integer type. |
| **DFNT_UINT64** | 27 | 64-bit unsigned integer type. |

| | | |
|---|---|---|
| **DFNT_FLOAT32** | 5 | 32-bit floating-point type. |
| **DFNT_FLOAT64** | 6 | 64-bit floating-point type. |

### 2.5.3 Tag Definitions

These definitions identify the object tags defined and used by the HDF API library. The concept of object tags is introduced in Section 2.3.1 on page 8. Note that, in the tag list that follows in Table 2C, tags can also identify properties of data objects such as raster image compression and pixel interlacing type (these concepts are described in Chapter 6, titled *8-bit Raster Images. (DFR8 API)* 8- and 24-raster images and scientific data sets are respectively described in Chapter 6, titled *8-bit Raster Images (DFR8 API)*, Chapter 7, titled *24-bit Raster Images (DF24 API)* and Chapter 3, titled *Scientific Data Sets (SD API)*.

TABLE 2C **Tag Definitions**

| Definition Name | Definition Value | Description |
|---|---|---|
| **DFTAG_FID** | ((uint16) 100) | File identifier. |
| **DFTAG_FD** | ((uint16) 101) | File description. |
| **DFTAG_TID** | ((uint16) 102) | Tag identifier. |
| **DFTAG_TD** | ((uint16) 103) | Tag descriptor. |
| **DFTAG_DIL** | ((uint16) 104) | Data identifier label. |
| **DFTAG_DIA** | ((uint16) 105) | Data identifier annotation. |
| **DFTAG_NT** | ((uint16) 106) | Number type. |
| **DFTAG_ID8** | ((uint16) 200) | 8-bit raster image dimension record. |
| **DFTAG_IP8** | ((uint16) 201) | 8-bit raster image palette. |
| **DFTAG_RI8** | ((uint16) 202) | 8-bit raster image data. |
| **DFTAG_CI8** | ((uint16) 203) | 8-bit raster image data - RLE compressed. |
| **DFTAG_II8** | ((uint16) 204) | 8-bit raster image data - IMCOMP compressed. |
| **DFTAG_ID** | ((uint16) 300) | 24-bit raster image dimension record. |
| **DFTAG_LUT** | ((uint16) 301) | 24-bit raster image palette. |
| **DFTAG_RI** | ((uint16) 302) | 24-bit raster image data. |
| **DFTAG_CI** | ((uint16) 303) | 24-bit raster image - compressed. |
| **DFTAG_RIG** | ((uint16) 306) | Raster image group. |
| **DFTAG_LD** | ((uint16) 307) | Palette dimension record. |
| **DFTAG_SDG** | ((uint16) 700) | Scientific data group. |
| **DFTAG_SDD** | ((uint16) 701) | Scientific data dimension record. |
| **DFTAG_SD** | ((uint16) 702) | Scientific data group data. |
| **DFTAG_SDS** | ((uint16) 703) | Scientific data scale. |
| **DFTAG_SDL** | ((uint16) 704) | Scientific data label. |
| **DFTAG_SDU** | ((uint16) 705) | Scientific data unit. |
| **DFTAG_NDG** | ((uint16) 720) | Numeric data group. |
| **DFTAG_CAL** | ((uint16) 731) | Calibration information. |
| **DFTAG_VG** | ((uint16) 1965) | Vgroup. |
| **DFTAG_VH** | ((uint16) 1962) | Vdata header. |
| **DFTAG_VS** | ((uint16) 1963) | Vdata storage. |
| **DFTAG_RLE** | ((uint16) 11) | Run-length encoding compression algorithm. |
| **DFTAG_IMCOMP** | ((uint16) 12) | IMCOMP compression algorithm. |
| **DFTAG_JPEG** | ((uint16) 13) | JPEG compression algorithm - 24-bit data. |
| **DFTAG_GREYJPEG** | ((uint16) 14) | JPEG compression algorithm - 8-bit data. |

| | | |
|---|---|---|
| **DFIL_PIXEL** | ((uint16) 0) | Pixel interlacing. |
| **DFIL_LINE** | ((uint16) 1) | Scan-line interlacing. |
| **DFIL_PLANE** | ((uint16) 2) | Scan-plane interlacing. |

### 2.5.4 Limit Definitions

These definitions declare the maximum size of specific data object parameters, such as the maximum length of a vdata field or the maximum number of objects in a vgroup. Vdata objects are discussed in Chapter 4, titled *Vdatas (VS API)*, and vgroup objects are described in Chapter 5, titled *Vgroups (V API)*. Except for `FIELDNAMELENMAX`, these can be safely altered by the HDF user.

TABLE 2D **Limit Definitions**

| Definition Name | Definition Value | Description |
|---|---|---|
| **FIELDNAMELENMAX** | 128 | Maximum length of a vdata field in bits - 16 characters. |
| **VSNAMELENMAX** | 64 | Maximum length of a vdata name in bytes - 64 characters. |
| **VGNAMELENMAX** | 64 | Maximum length of a vgroup name in bytes - 64 characters. |
| **MAX_VFILE** | 16 | Maximum number of open data files. |
| **MAXNVELT** | 64 | Maximum number of objects in a vgroup. |
| **MAX_ORDER** | 32000 | Maximum order of a vdata field. |
| **MAX_FIELD_SIZE** | 32000 | Maximum length of a field. |

## 2.6 Basic Operations on HDF Files

The HDF programming model specifies that a data file be first explicitly opened by an application, then manipulated, then explicitly closed by the application code at the end of its execution. To use the routines designed to open and close data files, the user must first know about the ***file identifiers*** used by the HDF API routines.

### 2.6.1 File Identifiers

HDF data files are uniquely identified by either a filename or a file identification number, or ***file id***. The filename is the name of the file as represented in the native filesystem, and is created along with the file itself through the HDF file creation routine. The file id is the numeric identifier given to the file by the HDF library, also at the time of creation, and is generally only used by the HDF library routines; the HDF user need not keep track of them.

As every file is assigned its own identifier, the order in which files are accessed is very flexible. For example, it is perfectly valid to open a file and obtain an identifier for it, then open a second file without closing the first file or disposing of the first file identifier. The only requirement made by HDF is that all file ids be individually discarded before the termination of the calling program.

File identifiers created by any HDF API routine cannot be used by the routines of any other interface - they are not interchangeable.

### 2.6.2 Opening HDF Files: Hopen

The **Hopen** routine opens or creates an HDF data file, depending on the access mode specified, and returns the file id the HDF library has assigned it. The parameter names and data types are listed in Table 2E below. Refer also to the *HDF Reference Manual* for additional information on **Hopen** and to Section 2.5 on page 11 for information regarding file access codes.

TABLE 2E

**Hopen Parameter List**

| Routine Name (Fortran-77) | Parameter | Data Type | | Description |
|---|---|---|---|---|
| | | C | Fortran-77 | |
| **Hopen (hopen)** | filename | `char *` | `character8 (*)` | File identifier. |
| | access | `intn` | `integer` | Access mode definition. |
| | n_dds | `int16` | `integer*2` | Number of data descriptors in a block if a new file is to be created. |

## 2.6.3 Closing HDF Files: Hclose

The **Hclose** routine closes the file designated by the file id passed in as the `file_id` parameter. The parameter names and data types are listed in Table 2F below. Refer also to the *HDF Reference Manual* for additional information regarding **Hclose**.

TABLE 2F

**Hclose Parameter List**

| Routine Name (Fortran-77) | Parameter | Data Type | | Description |
|---|---|---|---|---|
| | | C | Fortran-77 | |
| **Hclose (hclose)** | file_id | `int32` | `integer*4` | File identifier. |

## 2.6.4 Determining the Number of Objects with a Specified Tag: Hnumber

**Hnumber** determines how many objects with the specified tag exist within a file. To count the total number of objects in a file, set the `tag` argument to `DFTAG_WILDCARD`. Note that a return value of 0 is not a error condition. The parameter names and data types are listed in Table 2G below. Refer also to the *HDF Reference Manual* for additional information regarding **Hnumber**.

TABLE 2G

**Hnumber Parameter List**

| Routine Name (Fortran-77) | Parameter | Data Type | | Description |
|---|---|---|---|---|
| | | C | Fortran-77 | |
| **Hnumber (hnumber)** | file_id | `int32` | `integer*4` | File identifier. |
| | tag | `int32` | `uint16` | Tag to be counted. |

## 2.6.5 Getting the HDF Library Version Used to Create a File: Hgetlibversion

**Hgetlibversion** returns the version of the HDF library currently being used, as well as additional textual information regarding the library. The parameter names and data types are listed in Table 2H below. Refer also to the *HDF Reference Manual* for additional information regarding **Hgetlibversion**.

TABLE 2H

**Hgetlibversion Parameter List**

| Function Name (Fortran-77) | Parameter | Data Type C | Description |
|---|---|---|---|
| **Hgetfileversion** | major_v | `uint32 *` | Major version number. |
| | minor_v | `uint32 *` | Minor version number. |
| | release | `uint32 *` | Complete library version number. |
| | string | `char [80]` | Additional information about the library version. |

### 2.6.6 Locate an Object by its Tag/Reference Number Pair: Hfind

**Hfind**, like all H functions, must be preceded by a call to **Hopen** and followed at some point by a call to **Hclose**.

Although **Hfind** is capable of executing many kinds of search operations, it is particularly useful for determining the valid reference numbers for any specified tag. When supplied with a recognized HDF tag, a wildcarded reference number and a search direction, **Hfind** will search sequentially through the objects stored in an HDF file until the first data object with the specified tag is encountered. Once the data object is discovered, **Hfind** will return its reference number. By using **Hfind** in a conditional loop the reference number for any or all data objects in an HDF file can be retrieved. For more about **Hfind** consult the *HDF Reference Guide*.

TABLE 2I

**Hfind Parameter List**

| Routine Name (Fortran-77) | Parameter | Data Type C | Description |
|---|---|---|---|
| **Hfind** | major_v | `uint32 *` | Major version number. |
| | minor_v | `uint32 *` | Minor version number. |
| | release | `uint32 *` | Complete library version number. |
| | string | `char [80]` | Additional information about the library version. |

## 2.7 Application Programming Interfaces

HDF provides Fortran-77 and C APIs for storing and retrieving 8- and 24-bit raster images, palettes, scientific data, and annotations. These interfaces are described in detail in Chapters 3 through 12 of this manual.

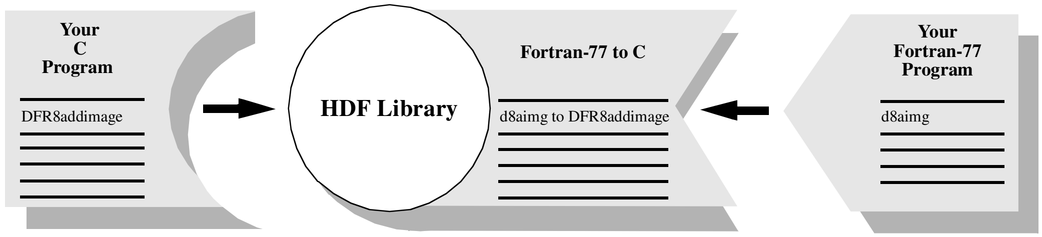
## 2.8 Fortran-77 and C Language Issues

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.

### 2.8.1 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 2e **Use of a Function Call Converter to Route Fortran-77 HDF Calls to the C Library**




## 2.8.2 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.

## 2.8.3 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.

## 2.8.4 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.

## 2.8.5 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 2J.)

TABLE 2J **Data Type Definitions**

| Data Type | C | Fortran-77 |
|---|---|---|
| **8-bit signed integer** | `int8` | `integer*1` |
| **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. |

| Data Type | C | Fortran-77 |
|---|---|---|
| **32-bit floating point number** | `float32` | `real*4` |
| **64-bit floating point number** | `float64` | `real*8` |
| **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.

### 2.8.6 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.

### 2.8.7 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.

## 2.9 Low-Level Interfaces

The low-level HDF interface consists of the ***H interface***, the ***HL/HX interface*** and the ***HE interface***.

### 2.9.1 The H Interface

The low-level H interface builds and manipulates data objects in an HDF file. A thorough explanation of each H interface routine can be found in the *HDF Specification and Developer's Guide*. Table 2K lists and briefly describes the H interface routines that are most commonly used in HDF applications.

TABLE 2K **Some H Interface Routines**

| Category | Routine Name | Description |
|---|---|---|
| **Input/Output** | `Hopen` | Provides an access path to a file. |
| | `Hclose` | Closes an access path to a file. |
| | `Hgetlibversion` | Returns version information for the current HDF library. |

### 2.9.2 The HX Interface

The HX interface routines create and maintain linked and external data elements. A thorough explanation of each HX interface routine can be found in the *HDF Specification and Developer's Guide*.

### 2.9.3 The HE Interface

The HE interface routines provide error handling functionality. The HE interface routines are described in the *HDF Specification and Developer's Guide* and Chapter 12, titled *Error Reporting*.