* Software Overview

# Chapter Overview

This chapter describes the HDF software organization and provides guidelines for writing HDF software.

HDF is an amalgam of code and functionality from many sources. For example, the netCDF code came from the Unidata Program Center, and data compression and conversion software has been acquired from a variety of third parties. The HDF development team wrote the code for the basic HDF functionality and perfomed all of the integration work.

This document contains specifications for the HDF code and functionality. It does not include specifications for code or functionality from non-NCSA sources, though it does sometimes refer to specifications provided by other sources. Only the HDF interface to such code is specified in this document.

## HDF Software Layers

There are three basic levels of HDF software:

* HDF low-level interface
* HDF application interfaces
* HDF applications and utilities

The lowest layer, the *low-level interface*, includes general purpose routines that form the basis of all higher-level HDF development. The low-level interface directly executes operations such as file I/O, error handling, memory management, and physical storage.

The *application interfaces* support higher level views of data and provide the interfaces for building user-level applications. Routines that handle raster images, palettes, annotations, scientific data sets, vdatas, vgroups, and netCDF appear at this level.

The *applications and utilities* are implemented at the highest level.

The utilities perform general functions, such as listing the contents of an HDF file, and more specialized functions, such as converting data from one HDF data type to another (e.g., raster images to scientific data sets). In general, the utilities have simple command line interfaces and perform data management tasks.

The applications usually perform data analysis tasks and have polished interactive user interfaces. They include the NCSA Visualization Tool Suite, commercial software packages that use HDF, and other packages created by various third party projects.

Figure 3a illustrates this layered implementation.

* HDF Software Layers

[[1]](#footnote-1)



The low-level interface is described in detail in this document. The application interfaces and command line utilities are described in the document *NCSA HDF Calling Interfaces and Utilities* for Versions 3.2 and earlier and in the *HDF User’s Guide* and *HDF Reference Manual* for Versions 3.3 and 4.x. Other HDF-based software tools should have their own manuals.

Since the original HDF user community wrote programs primarily in C and FORTRAN, all HDF application interfaces are callable from both C and FORTRAN programs. The functions of the low-level interface, however, are provided only as C-callable routines.

## Software Organization

### Versions and Release Numbers

Since HDF is under continual development, new releases are periodically made available. Releases are identified with a version number consisting of three elements:

< majorv > Major version number, integer

< minorv > Minor version number, integer

< rn > Release number, integer

The version number is presented in the following format:

< majorv >.< minorv >r< rn >(e.g., Version 3.2r1)

These elements are interpreted as follows:

Major version number

A new major version number is assigned when there is some fundamental difference between a new version of the library and the previous version. When a new major version is released, HDF users and developers are strongly encouraged to obtain the new source code and documentation. There will probably be added functionality in successive major versions of the library and some obsolete code may be deleted. Some user code may have to be modified to use the new library.

Minor version number

A new minor version number indicates an intermediate release between one major version and the next. Changes will probably be significant. When a new minor version is released, users and developers are strongly encouraged to obtain the new source code and documentation. There may be minor interface changes.

Release number

A new release number is assigned when bug fixes or other small modifications have been made. Using a new release of the same version of the library will not usually require modifying existing user code.

### ANSI C and Portability

To ensure that HDF can be easily ported to new platforms, all versions of the HDF source code from Version 3.2 on are written in ANSI standard C, with special provisions for non-ANSI compilers. For more information about porting HDF and writing portable HDF-based code, refer to Chapter 11, Portability Issues.

### Modules and Interfaces

The HDF distribution contains many source files or modules that can be grouped into families. For example, dfp.c, dfpf.c, and dfpff.f all share the root name dfp and, therefore, all belong to the dfp family. In general, each family of source modules represents one HDF applications interface; the dfp family represents the HDF Palette Interface (DFP).

For each interface, there is necessarily one file that contains the C code that provides the basic functionality of that interface. Some interfaces may have one or two additional code modules that provide FORTRAN callability for the interface, so a family may have one, two, or three files:

1 file Modules of this sort are generally not calling interfaces themselves; they provide useful support functions for actual calling interfaces. Since they are not meant to be called by any routine outside the HDF library, they do not need to be FORTRAN-callable. Example: hblocks.c is called only by internal HDF routines and has only the C-callable interface.

2 files Some interfaces need only one extra source module to provide FORTRAN compatibility. In such cases, there are only two source modules for the interface. Example: mfan.c and mfanf.c make up the Multifile Annotation Interface.

3 files Most current implementations of FORTRAN-callable HDF interfaces require that character string arguments be passed to some of their functions. Due to differences in the way C and FORTRAN represent strings, passing strings requires that there be a small amount of special purpose FORTRAN code written for each function that takes a string argument.

Therefore, most FORTRAN-callable HDF interfaces consist of three source modules:

* The primary C module
* A FORTRAN-callable C module
* A FORTRAN module

Example: dfsd.c, dfsdf.c, and dfsdff.f make up the Single-file Scientific Data Interface. dfsd.c contains the basic functionality of the interface. dfsdf.c provides the major part of FORTRAN callability. And dfsdff.f contains the special purpose FORTRAN code that enables passing character string arguments.

Table 3a, "HDF Version 4.x Source Code Modules," on page 20 lists the families of source code modules and header files of HDF Version 4.x. The first column of the table lists the name of the interface or the category of the modules, depending on their functionality. The modules are categorized as follows:

* Low-level interface, or H-level interface, includes modules that facilitate portability and provide physical storage management, error handling mechanisms, support for simultaneous access to multiple objects within a single file, support for simultaneous access to multiple files, and an interface for key lower-level modules. Low-level routines begin with an H (e.g., Hopen/Hclose or Hread/Hwrite).
* Multifile Scientific Data interface (SD API) includes modules that provide the mechanisms for managing scientific data sets in a multifile environment. These modules reside in the directory mfhdf/, which is separate from that of the other interfaces. Library routines in this interface begin with SD. This interface replaces the Single-file Scientific Data interface (DFSD API). (A subtantial number of local or internal routine names in this code are influenced by netCDF. )
* Vdata interface (VS API) includes modules that provide mechanisms for managing Vdatas. Library routines in this interface begin with VS.
* Vgroup interface (V API) includes modules that provide mechanisms for managing Vgroups. Library routines in this interface begin with a V. Note that in the Content Description column, the V and VS routines share some modules and header files.
* Multifile Annotation interface (AN API) includes modules that provide mechanisms for managing annotations in a multifile environment. Library routines in this interface begin with AN. This interface replaces the Single-file Annotation interface (DFAN API).
* General Raster Image interface (GR API) includes modules that provide mechanisms for managing general raster images in a multifile environment. Library routines in this interface begin with GR. This interface replaces the 8-bit Raster Image interface (DFR8 API) and the 24-bit Raster Image interface (DFR24 API), which operate in the single-file environment.
* Palette interface (DFP API) includes modules that provide mechanisms for managing the palettes that are used by the raster image interfaces. Library routines in this interface begin with DFP.
* Compression/Decompression includes modules that provide mechanisms for managing file and image compresion and decompression.
* Conversion includes modules that provide mechanisms to support conversion to and from the HDF format.
* Single-file Scientific Data interface (DFSD API) includes modules that provide mechanisms for managing scientific data sets in a single-file environment. Library routines in this interface begin with DFSD. This interface is replaced by the Multifile SD interface (SD API).
* Single-file General Raster Image interface (DFGR API) includes modules that provide mechanisms for managing general raster images in the single-file environment. This interface is an older version of the GR interface.
* 8-bit Raster Image interface (DFR8 API) includes modules that provide mechanisms for managing 8-bit raster images. This interface is replaced by the Multifile GR interface.
* 24-bit Raster Image interface (DFR24 API) includes modules that provide mechanisms for managing 24-bit raster images. This interface is replaced by the Multifile GR interface.
* Single-file Annotation interface (DFAN API) includes modules that provide mechanisms for managing annotations in the single-file environment. This interface is replaced by the Multifile AN interface.
* Developer-level interface includes modules that are at a lower level than the H-level modules, which heavily use the developer-level routines. These modules simplify the task of writing HDF applications by providing low-level routines for internal I/O handling, dynamic storage handling, memory management, and data descriptor handling.
* Mac Only interface includes modules that implement UNIX-like directory reading for the Macintosh.

The second column of Table 3a divides the modules in the interface into three groups: header files, C modules, and FORTRAN interface and support. The header files are discussed in the next section. The C modules group contains the primary C modules. The FORTRAN interface and support group contains either or both the FORTRAN-callable C module and the FORTRAN module of the interface.

### Header Files

In addition to the source code modules discussed above, some interfaces also have C header files associated with them that are meant to be included by C applications programmers with the #include preprocessor directive. They contain useful constants and data structures for interaction with the interface from C programs. The header files can be identified by the same name as the root name for the rest of the family with the .h extension. For example, dfsd.h is the header file for the Single-file Scientific Data Interface.

Of particular importance among the C header files are mfhdf.h, hdf.h and hdfi.h:

mfhdf.h Contains symbolic constants and public data structures for HDF’s SD interface. mfhdf.h must be included by any program that uses the SD API of the HDF library.

hdf.h Contains all the symbolic constants and public data structures required by HDF. hdf.h must be included by any program that uses the HDF library. (Note that this file is automatically included by the inclusion of mfhdf.h and need not be included separately.)

hdfi.h Contains specific portability information about each platform on which HDF is supported. hdfi.h is automatically included in a program when hdf.h is included, so programmers need not explicitly include it.

Refer to Chapter 11, Portability Issues, for more information on hdfi.h and other portability issues. Refer to Table 3a for the listing of the header files provided in the current version of the HDF library.

* HDF Version 4.x Source Code Modules

|  |  |  |  |
| --- | --- | --- | --- |
| Category | Module type | Module name | Content Description |
| H-level | Header files | hchunks.h  hdf.h  hdfi.h  herr.h  hfile.h  hkit.h  hlimits.h  hntdefs.h  hproto.h  htags.h  patchlevel.h | Definitions for chunked elements  HDF user-level definitions, for applications using HDF routines  Definitions for portability  Definitions for HDF error handling/reporting routines  Definitions for HDF low-level file I/O routines  Definitions for string mapping routines  Defined limits for the library, reserved Vdata/Vgroup names and classes, and pre-attribute names. Definitions for most of the constants in the library.  Number-type definitions for HDF  Useful macros, potential for future functions  HDF tag definitions  Definition of PATCHLEVEL |
| C modules | hblocks.c  hchunks.c  herr.c  hextelt.c  hfile.c  hkit.c | Routines to implement linked-block elements  Routines to implement chunked elements  Routines for error handling/reporting  Routines for external elements  Low-level file I/O routines  Various string mapping routines |
| FORTRAN interface and support | herrf.c | C stubs for FORTRAN error handling/reporting routines |
| Multifile  Scientific Data  (SD API) | Header files | alloc.h  error.h  hdf2netcdf.h  local\_nc.h  mfhdf.h  mfhdfi.h  win32cdf.h | Definitions for memory management  Prototypes for error handling routines  HDF names of netCDF API functions  Definitions of structures for CDF and its components  Definitions for applications using SD routines  Definitions that are used in both local\_nc.h and mfhdf.h  Definitions used for the Windows version of the library |
| C modules | array.c  attr.c  cdf.c  dim.c  error.c  file.c  globdef.c  hdfsds.c  iarray.c  mfsd.c  nssdc.c  putget.c  putgetg.c  sharray.c  string.c  var.c  xdrposix.c  xdrstdio.c | Routines that operate the structure NC\_array  Routines that operate the structure NC\_attr  Routines that operate the CDF structure NC its components  Routines that operate NC\_dim and locally related routines  Utility routines to implement consistent error logging mechanisms for netCDF  Low-level "nc" routines and other routines that operate the structures NC and XDR  Initialization of global variables that allow the creation of SunOS sharable libraries  Routines that read old SDS objects out of HDF files  Routines that operate NC\_iarray  SD and SDI library routines that are local to this module  Routines that read CDF V2.x files created with the CDF library  Routines that read/write SD objects at the Vgroup and Vdata level  Routines that perform I/O on a generalized hyperslab  Internal routines for short integers  Routines that operate NC\_string  Routines that operate NC\_var and locally related routines  Routines that implement XDR on a POSIX file descriptor  Routines that implement XDR on a stdio stream |
| FORTRAN interface and support | mfsdf.c  mfsdff.f | C stubs for SD library routines  FORTRAN stubs for SD library routines |
| Vdata (VS API) | Header files | vattr.h | definitions for vgroup/vdata attribute interface |
| C modules | vattr.c  vg.c  vhi.c  vio.c  vrw.c  vsfld.c | V and VS library routines that handle Vgroup/Vdata attributes  Mostly Vdata library routines, but also some Vgroups routines  VH library routines for vdata high-level access  VS library routines that handle vdatas and locally used routines  VS library routines that read and write vdatas  VF and VS library routines that handle vdata fields |
| FORTRAN interface and support | vattrf.c  vattrff.f  vgf.c  vgff.f | C stubs for handling vgroup/vdata attributes  FORTRAN stubs for handling vgroup/vdata attributes  C stubs for vgroups and vdatas library routines  FORTRAN stubs for vgroups and vdatas library routines |
| Vgroup (V API) | Header files | dfgroup.h  vg.h  vgint.h | Definitions for dfgroup.c  Defined symbols and structures used in all v\*.c files  Private defined symbols and structures used in all v\*.c files |
| C modules | vconv.c  vgp.c  vparse.c | Routines that handle Vgroup/Vdata compatibility and conversion  V library routines that handle Vgroups and locally used routines  Routines for parsing |
| FORTRAN interface and support |  | listed in Vdata API |
| Multifile  Annotation  (AN API) | Header files | mfan.h | Definitions for multifile annotations |
| C modules | mfan.c | AN library routines that read and write multifile annotations |
| FORTRAN interface and support | mfanf.c | C stubs for handling multifile annotations |
| Multifile  General Raster Image (GR API) | Header files | mfgr.h | Definitions for multifile general raster images |
| C modules | mfgr.c | GR library routines that access multifile general raster images |
| FORTRAN interface and support | mfgrf.c  mfgrff.f | C stubs for accessing multifile general raster images  FORTRAN stubs for accessing multifile general raster images |
| Palette  (DFP API) | Header files |  | This interface uses only the header file hdf.h |
| C modules | dfp.c | DFP routines that read and write palettes |
| FORTRAN interface and support | dfpf.c  dfpff.f | C stubs for palette routines  FORTRAN stubs for palette routines |
| Compression/  Decompression | Header files | cnbit.h  crle.h  hcomp.h  hcompi.h | Definitions for N-bit encoding  Definitions for run-length encoding  Definitions for compression information and structures  Internal library header file for compression information |
| C modules | crle.c  dfcomp.c  dfjpeg.c  dfrle.c  dfunjpeg.c  hcomp.c  hcompri.c | Internal I/O routines for HDF run-length encoding  Routines that perform file compression  Routines that perform JPEG image compression  Routines that perform RLE image compression  Routines that perform JPEG image decompression  I/O routines for compressed data  Routines for reading and writing old-style compressed raster images, such as JPEG, (raster specific) RLE, and IMCOMP |
| FORTRAN interface and support | none |  |
| Conversion | Header files | dfconvrt.h  dfufp2i.h  hconv.h | The macro DFconvert to speed up the conversion process  Definitions for dfufp2i.c  Definitions for data conversion |
| C modules | dfconv.c  dfkconv.c  dfkcray.c  dfkfuji.c  dfknat.c  dfkswap.c  dfkvms.c  dfufp2i.c | Routines that support conversion to and from HDF format  Routines to support Convex-native conversion to/from HDF format  Routines to support Cray conversion to/from HDF format  Routines to support Fujitsu-native (VP) conversion to/from HDF format  Routines to support native-mode conversion to/from HDF format  Routines to support little-endian conversion to/from HDF format  Routines to support Vax-native conversion to/from HDF format  Utility functions to convert floating point data to 8-bit raster image set (RIS8) format |
| FORTRAN interface and support | none |  |
| Single-file  Scientific Data  (DFSD API) | Header files | dfsd.h | Definitions for single-file scientific data |
| C modules | dfsd.c | DFSD routines that read and write Scientific Data Sets |
| FORTRAN interface and support | dfsdf.c  dfsdff.f | C stubs for single-file scientific data routines  FORTRAN stubs for single-file Scientific Data routines |
| Single-file  General Rasters  (DFGR API) | Header files | dfgr.h | Definitions for single-file general and 24-bit raster images |
| C modules | dfgr.c  dfimcomp.c | DFGR routines that read and write general raster images (old)  Routines that perform color image compression |
| FORTRAN interface and support | none |  |
| 8-bit Raster Images  (DFR8 API) | Header files | dfrig.h | Definitions for 8-bit raster image groups |
| C modules | dfr8.c | DFR8 routines that read and write 8-bit raster image groups |
| FORTRAN interface and support | dfr8f.c  dfr8ff.f | C stubs for 8-bit raster image group routines  FORTRAN stubs for 8-bit raster image group routines |
| 24-bit Raster Images  (DFR24 API) | Header files |  | This interface uses dfgr.h in the single-file General Raster interface |
| C modules | df24.c | Routines that read and write 24-bit raster images |
| FORTRAN interface and support | df24f.c  df24ff.f | C stubs for 24-bit raster image routines  FORTRAN stubs for 24-bit raster image routines |
| Single-file  Annotations  (DFAN API) | Header files | dfan.h | Definitions for single-file annotations |
| C modules | dfan.c | Routines that read and write single-file annotations |
| FORTRAN interface and support | dfanf.c  dfanff.f | C stubs for annotation routines  FORTRAN stubs for annotation routines |
| Developer-level | Header files | atom.h  bitvect.h  cdeflate.h  cnbit.h  cnone.h  cskphuff.h  cszip.h  df.h  dfi.h  dfivms.h  dfstubs.h  dfutil.h  dgroup.h  dynarray.h  glist.h  hbitio.h  hqueue.h  linklist.h  maldebug.h  mcache.h  mstdio.h  tbbt.h | Definitions for atom code  Definitions for bit vector code  Definitions for deflate encoding  Definitions for N-bit encoding  Definitions for none-encoding  Definitions for Skipping Huffman encoding  Definitions for szip encoding  Definitions for data descriptors  HDF internal header file  HDF internal header file for VMS  Definitions for dfstubs.c HDF 3.1 emulation using new routines from hfile.c  Definitions for low-level utility routines  Definitions for low-level implementation of groups  Definitions for dynamic storage handling  Definitions for general list  Data structures and macros for bitfile access to HDF data objects; mainly used for compression I/O and N-bit data objects  Modified version of Berkley code for manipulating memory pool  Definition for generic linked lists  Definitions for dynamic memory handling  Modified version of Berkley code for manipulating memory pool  Definitions for stdio-like routines  Definitions for using threaded, balanced, binary trees |
| C modules | atom.c  bitvect.c  cdeflate.c  cnbit.c  cnone.c  cskphuff.c  cszip.c  dfstubs.c  dfgroup.c  dfutil.c  dynarray.c  glist.c  hbitio.c  hbuffer.c  hdfalloc.c  hfiledd.c  linklist.c  maldebug.c  mcache.c  mstdio.c  tbbt.c | Internal storage routines for handling atoms  Routines that operate ordered sets of bits, or bit vectors  Internal I/O routines for HDF gzip deflate encoding  Internal I/O routines for HDF N-bit encoding  Internal I/O routines for HDF noencoding  Internal I/O routines for HDF Skipping Huffman encoding  Internal I/O routines for HDF szip encoding  V3.x stubs for V4.0 H-level I/O routines  Low-level routines (DF\*) for implementing groups  General-purpose utility routines  Internal routines that handle dynamic storage  Implementation of general list  HDF bit level I/O routines  Routines that manage buffered elements  HDF routines for memory management  Routines that manage DDs and DD blocks  Internal storage routines for handling generic linked lists  Utility routines for memory management  Modified version of Berkley code for manipulating memory pool  HDF stdio-like routines  Routines for using threaded, balanced, binary trees |
| FORTRAN interface and support | dff.c  dfff.f  dfutilf.c  dfutilff.f  hfilef.c  hfileff.f | C stubs for low-level I/O routines  FORTRAN stubs for low-level I/O routines  C stubs for general-purpose utility routines  FORTRAN stubs for general-purpose utility routines  C stubs for low-level routines  FORTRAN stubs for low-level routines |
| Mac only | Header files | dir\_mac.h  sys\_dir\_mac.h | Definitions for dir\_mac.c  Additional definitions to be included in dir\_mac.h |
| C modules | dir\_mac.c | Implementation of UNIX-like directory reading for the Macintosh |

### The HDF Test Suite

In addition to the source code for the HDF library, Versions 3.2 and higher include a test suite. There are two test modules: one for C and one for FORTRAN. Each module tests all of the routines in all of the application interfaces and in the low-level interface. The exact form of these test modules may vary from one release to the next; consult the release code and online test documentation for details.

Every effort has been made to ensure that the test programs provide a thorough and accurate assessment of the health of the HDF library. Although the test suite will greatly improve the reliability of HDF code, it is almost inevitable that some parts of the code will remain untested. Therefore, no guarantees can be made on the basis of test suite performance.

### Sample HDF Programs

Sample programs, illustrating some of the common techniques employed by HDF programmers, are available on the HDF web site at https://support.hdfgroup.org/services/learning.html.

To help users become familiar with HDF, each release includes several sample programs illustrating common techniques employed by HDF programmers.

## Some HDF Conventions

The HDF specification described in the previous chapter is not sufficient to guarantee its success. It is also important that HDF programmers and users adhere to certain conventions. Some guidelines are implicit in the discussions in other sections of this document. Others are presented in the document *NCSA HDF Calling Interfaces and Utilities* for Versions 3.2 and earlier, or in the *HDF User’s Guide* and the *HDF Reference Manual* for Versions 3.3 and 4.x.

Guidelines not covered elsewhere are introduced in this section.

Naming and Assigning Tags

Tags that are to be made available to a general population of HDF users should be assigned and controlled by The HDF Group (THG). Tags of this type are given numbers in the range 1 to 32,767. If you have an application that fits this criterion, contact THG at the address listed in the front matter at the beginning of this manual and specify the tags you would like. For each tag, your specifications should include a suggested name, information about the type and structure of the data that the tag will refer to, and information about how the tag will be used. Your specifications should be similar to those contained in Chapter , Tag Specifications. THG will assign a set of tags for your application and will include your tag descriptions in the HDF documentation.

Tags in the range 32,768 to 64,999 are user-definable. That is, you can assign them for any private application. If you use tags in this range, be aware that they may conflict with other people's private tags.

Using Reference Numbers to Organize Data Objects

The HDF library itself uses reference numbers solely to distinguish among objects with the same tag. While application programmers may find it convenient to impart some meaning to reference numbers, they should be forewarned that the HDF library will be ignorant of any such meaning.

|  |
| --- |
| **Note:** Users are discouraged from assigning any meaning to reference numbers beyond that imparted by the HDF library. |

Multiple References

Multiple references to a single data element are quite common in HDF. The low-level routine Hdupdd generates a new reference to data that is already pointed to by another DD. If Hdupdd is used several times, there may be several DDs that point to the same data element.

It is important to note that when a multiply-referenced data element is deleted or moved, the various DDs that previously pointed to the data element are *not* automatically deleted or adjusted to point to the data element in its new location. Consequently, each DD to be deleted or moved should be checked for multiple references and handled appropriately.

1. This is a simplified illustration of the HDF software layers. Though the basic principles illustrated here continue to apply, the introduction of netCDF and multiple-file HDF data structures renders the implementation considerably more complex. [↑](#footnote-ref-1)