* Low-level Interface

# Chapter Overview

This chapter provides a detailed description of the routines that make up the HDF low-level interface, sometimes referred to as the H-level interface.

## Introduction

HDF supports several interfaces which can be categorized as high-level and low-level interfaces:

* High-level interfaces support utilities and applications.
* Low-level interface functions perform basic operations on HDF files.

These levels are illustrated in Figure 4b.

* HDF Software Layers



This chapter is concerned only with the low-level interface.

Using these routines of the low-level interface, you will be able to build and manipulate HDF objects of any type, including those of your own design. All HDF applications use them as basic building blocks.

The low-level routines are all written in C.

## New Low-level Routines with Version 3.2 and Higher

The low-level routines described in this chapter are new with HDF Version 3.2 and higher; they replace the routines provided with earlier versions. The new routines provide better performance and increased functionality and users are strongly advised to use them in new applications. The old routines are supported through emulation, but may be eliminated from the HDF library in a future release.

The new lower layer incorporates the following improvements:

* More consistent data and function types
* More meaningful and extensive error reporting
* Simplification of key lower-level functions
* Simplified techniques to facilitate portability
* Support for alternate forms of physical storage, such as linked blocks storage and storage of the data portion of an object in an external file
* A version tag to indicate which version of the HDF library last changed a file
* Support for simultaneous access to multiple files
* Support for simultaneous access to multiple objects within a single file

The previous lower layer was called the DF layer because all routines began with the lettersDF (e.g., DFopen and DFclose)**.** The new lower layer is called the H layer because all routines begin with the letterH (e.g.,Hopen, Hclose, and Hwrite). The source modules containing these routines begin with the letter h (see Table 3a, "HDF Version 4.x Source Code Modules"):

hfile.c Basic I/0 routines

herr.c Error-handling routines

hkit.c General purpose routines

hblocks.c Routines to support linked block storage

hextelt.c Routines to support external storage of HDF data elements

hchunks.c Routines to support chunked elements

## Overview of the Low-level Interface

This section provides the name and purpose of each public function and selected private routines of the low-level interface. The private routines are intended only for internal use by the library. Detailed specifications for many of these routines are provided in Appendix , Function Specifications; detailed specifications for all of these routines are provided as comments in the distributed source code.

Summary of Prefixes

The low-level functions are named with the following prefixes.

* Low-level routine prefixes

|  |  |
| --- | --- |
| H | General and file-handling operations |
| HC | Compression special element operations |
| HD | DD block operations |
| HL | Linked block special element operations |
| HMC | Chunking special element operations |
| HR | Raster image special element operations |
| HT | Tag/ref operations |
| HX | External file special element operations |
| \*P | Routine private to the library. No guarantee of stable external interface; may change without notice. |
| \*I | Static routine private to the library. No guarantee of stable external interface; may change without notice. |

Opening and Closing HDF Files

These functions are used to open and close HDF files:

HopenProvides an access path to an HDF file and reads all of the DD blocks in the file into memory

HcloseCloses the access path to a file

HDerr Closes a file and returns FAIL

HsetaccesstypeSets the I/O access type (serial, parallel, ...)

Locating Elements for Access and Getting Information

These routines locate elements or acquire other information about an HDF file or its data objects. Except for Hendaccess, they initialize the element that they locate and return an *access ID* that is used in later references to the data element. Calls can include wildcards so that one can search for unknown tags and reference numbers (tag/refs).

Hstartread Locates an existing data element with matching tag/ref and returns an access ID for reading it

Hnextread Continues the search with the same access ID

Hendaccess Disposes of access ID for a tag/ref pair

Hinquire Returns access information about a data element

HishdfDetermines whether a file is an HDF file

HnumberReturns the number of occurrences of a specified tag/ref pair in a file

HexistDetermines whether an object exists in an HDF file

Hmpset Sets pagesize and maximum number of pages to cache on the next open/create operation

Hmpget Gets last pagesize and max number of pages cached for open/create

HgetlibversionReturns version information for the current HDF library

HgetfileversionReturns version information for an HDF file

HPgetdiskblock Gets the offset of a free block in the file

HPfreediskblock Releases a block in a file to be re-used

Reading and Writing Entire Data Elements

There are two sets of routines for reading and writing data elements. The routines described here are used to store and retrieve entire data elements.

Hputelement Adds or replaces elements in a file

Hgetelement Reads data elements in a file

A second set of routines, described in the next section, may be used if you wish to access only part of a data element.

Reading and Writing Part of a Data Element

The second set of routines for reading and writing data elements makes it possible to read or write all or part of a data element. One of the access routines Hstartread or Hstartwrite must be called before these Hwrite, Hread, or Hseek:

Hstartwrite Sets up writing to the object with the supplied tag/ref. If the object exists, it will be modified; otherwise it will be created.

Hwrite Writes data to a data element where the last Hwrite or Hseek stopped. If the space reserved is less than the length to write, then only as data as can fit in the allocated space is written.

Hread Reads a portion of a data element. It starts at the last position left by an Hread or Hseek call and reads any data that remains in the element up to a specified number of bytes.

HseekSets the access pointer to an offset within a data element. The next time Hread or Hwrite is called, the access occurs from this new position. The location to seek can be specified as an offset from the current location, from the start of the element, or from the end of the element.

HtruncTruncates a data set to a specified length.

Manipulating Data Descriptors (DDs)

The routines listed here perform operations on DDs without modifying the data to which the DDs refer. The first list indicates public routines that are available to users and applications; the second list indicates private routines that are used internally by the library.

Public user routines

Hdupdd Generates new references to a data element that is already referenced from somewhere else

Hdeldd Deletes a tag/ref pair from the list of DDs

HDcheck\_tagref Checks to see whether a tag/ref pair is in the DD list

HDreuse\_tagref Reuses a data descriptor of a tag/ref pair in a DD list of an HDF file

Hnewref Returns a reference number that is unique in the file

Htagnewref Returns a reference number that is unique in the file for a given tag

Hfind Locates the next object of a search in an HDF file

Private library routines (internal)

HTPcreate Creates (and attaches to) a tag/ref pair and inserts it into the DD list

HTPselect Attaches to an existing DD in the DD list

HTPendaccess Ends access to an attached DD in the DD list

HTPdelete Marks a tag/ref pair as free and ends access to it

HTPupdate Changes the offset and/or length of a data object

HTPinquire Gets the DD information for a DD (i.e. tag/ref/offset/length)

HTPis\_special Checks whether a DD identifier is associated with a special tag

HTPstart Initializes the DD list from disk, i.e., creates the DD list in memory

HTPinit Creates a new DD list in memory

HTPsync Flushes the DD list to disk

HTPend Closes the DD list to disk

Creating Special Data Elements

Prior to release 3.2, any data element had to be stored contiguously and all of the objects in an HDF file had to be in the same physical file. The contiguous requirement caused many problems, especially with regard to appending to existing objects. If you wanted to append data to an object, the entire data element had to be deleted and rewritten to the end of the file. Later HDF versions introduced alternate methods of storing HDF objects: linked blocks and external elements at the release of HDF Version 3.2 and chunking at HDF Version 4.1. These special elements, plus compression, are discussed in detail in Chapter 10, Extended Tags and Special Elements.

Linked blocks improve storage management by allowing elements in a single HDF file to be non-contiguous. The routines listed here operate on linked blocks The first list indicates the public routines that are available to users and applications; the second list indicates the private routines that are used internally by the library.

Public user routines

HLcreate Creates a new linked-block special data element

HLconvert Converts an AID into a linked-block element

HDinqblockinfo Returns information about linked blocks

Private library routines (internal)

HLPread Reads some data out of a linked-block element

HLPwrite Writes out some data to a linked-block element

HLPinquire Returns information about a linked-block element

HLPendacess Closes a linked-block AID

HLPinfo Returns information about a linked-block element

HLPstread Opens an access record for reading

HLPstwrite Opens an access record for writing

HLPseek Sets position for the next access

External elements allow a single HDF object to be stored in an external file. The following routines operate on external elements:

HXcreate Creates a new external file special data element

HXsetcreatedir Sets the directory variable for creating external file

HXsetdir Sets the directory variable for locating external file

It is not currently possible to store a single object (such as a very large data set) in multiple files. Nor can multiple objects be stored in one external file.

Once they are created with the routines HLcreate and HXcreate, these special data elements can be accessed with the routines used for normal data elements. These routines have two modes of operation. Calling HLcreate with a tag/ref that does not exist in a file will create a new element with the given tag/ref pair which will be stored as linked blocks. On the other hand, if the tag/ref pair already exists in the file, the referenced object will be promoted to linked block status. All data which had been stored in the object before the promotion will be retained. HXcreate behaves similarly.

Chunking allows elements in large arrays to be stored as chunks in such a way that I/O performance can be significantly improved. The routines listed here perform operations on chunking elements. The first list indicates the public routines that are available to users and applications; the second list indicates the private routines that are used internally by the library.

Public user routines

HMCcreate Creates a chunked element.

HMCwriteChunk Writes out the specified chunk to a chunked element.

HMCreadChunk Reads the specified chunk from a chunked element.

HMCsetMaxcache Sets the maximum number of chunks to cache.

HMCPcloseAID Closes the file but keeps AID active (for Hnextread()).

Private library routines (internal)

HMCPstread Opens an access record for reading.

HMCPstwrite Opens an access record for writing.

HMCPseek Sets the seek position.

HMCPchunkread Reads a single chunk from a chunked element.

HMCPread Reads a more arbitrarily sized piece of data from a chunked element.

HMCPchunkwrite Writes out a single chunk of data to a chunked element.

HMCPwrite Writes out a more arbitrarily sized piece of data to a chunked element.

HMCPinquire Implements Hinquire for a chunked element.

HMCPendacess Closes a chunked element AID

HMCPinfo Returns information about a chunked element.

Compression provides for the compression of data sets. The routines listed here perform those compression operations. The first list indicates the public routines that are available to users and applications; the second list indicates the private routines that are used internally by the library.[[1]](#footnote-1)

Public user routines

HCcreate Create a compressed data element

Private library routines (internal)

HCIinit\_coder Set the coder function pointers

HCIinit\_model Set the model function pointers

HCIread\_header Read the compression header info from a file

HCIstaccess Start accessing a compressed data element.

HCIwrite\_header Write the compression header info to a file

HCPcloseAID Get rid of the compressed data element data structures

HCPdecode\_header Decode the compression header info from a memory buffer

HCPencode\_header Encode the compression header info to a memory buffer

HCPendaccess Close the compressed data element and free the AID

HCPinfo return info about a compressed element

HCPinquire Inquire information about the access record and data element.

HCPmstdio\_endaccess   
Close the compressed data element

HCPmstdio\_inquire Inquire information about the access record and data element

HCPmstdio\_read Read in a portion of data from a compressed data element

HCPmstdio\_seek Seek to offset within the data element

HCPmstdio\_stread start read access for compressed file

HCPmstdio\_stwrite start write access for compressed file

HCPmstdio\_write Write out a portion of data from a compressed data element

HCPquery\_encode\_header   
Query the length of compression header for a memory buffer

HCPread Read in a portion of data from a compressed data element.

HCPseek Seek to offset within the data element

HCPstread Start read access on a compressed data element.

HCPstwrite Start write access on a compressed data element.

HCPwrite Write out a portion of data from a compressed data element.

HRPcloseAID Free memory but keep AID active

HRPconvert Wrap an existing raster image with the special element routines.

HRPendacess Free AID

HRPinfo Return info about a compressed raster element

HRPinquire Retreive information about a compressed raster element

HRPread Read some data out of compressed raster element

HRPseek Set the seek posn

HRPstread Open an access record for reading

HRPstwrite Open an access record for reading

HRPwrite Write data out to a compressed raster image

General special element routines: In addition to the routines specific to a particular type of special element, the library provides general routines for use on any special element:

HDget\_special\_info Gets information about a special element

HDset\_special\_info Resets information about a special element

Development Routines

The HDF library provides the following developer-level routines that simplify the task of writing HDF applications. Many of these routines mirror basic C library functions which are, unfortunately, not always completely portable in their library form:

HDgettagnameReturns a pointer to a text string describing a given tag

HDgetspace Allocates space

HDfreespaceFrees space

HDclearspace Creates storage on the heap for a number of items of the given size

HDregetspace Resizes to the new given size

HDstrcat Appends a string to the end of another string

HDstrcmp Compares two strings

HDstrncmp Compares two strings up to a given number of characters

HDstrcpy Copies a string from one location to another

HDstrncpy Copies a string from one location to another up to a given number of characters

HDstrlen Returns the length of a string

HDstrchr Returns the position of a character within a string

HDstrrchr Returns the position of the last occurence of a character within a string

HDstrtol Converts the initial portion of a string to a type long int representation

HDc2fstr Converts a C string into a Fortran string using the in place approach

HDf2cstring Converts a Fortran string to a C string

HDpackFstring Converts a C string to a Fortran string

HDflush Flushes the HDF file to disk

HDgettagnum Returns the tag number for a text description of a tag

HDgetNTdesc Returns a text description of a number type

HDfidtoname Returns the filename that the given file identifier corresponds to

Hexist Locates an object in an HDF file

HDgetc Reads a byte from a data element

HDputc Writes a byte to a data element

Hlength Returns the length of a data element

Hoffset Gets the offset of a data element in a file

Htrunc Truncates a dataset to the given length

Hcache Sets low-level caching for a file

HDvalidfid Checks whether a file identifier is valid

Error Reporting

The HDF library incorporates the notion of an *error stack.* This allows much of the context to be known when trying to decipher an error message.

Error reporting is handled by the following routines:

HEprintPrints out all of the errors on the error stack to a specified file

HEclear Clears the error stack

HERROR Reports an error and pushes the following information onto the error stack.

* Error type
* Source file name
* Line number and the name of the function reporting the error

HEreport Adds a text string to the description of the most recently reported error  
(Note: only one text string per error)

HEstring Returns error description

HEpush Pushes an error onto the stack

HEvalue Returns an error off the error stack

Standard C does not enable the code inside a function to know the name of the function. Therefore, to use the macro HERROR to report errors, there must exist a variable FUNC which points to a string containing the name of the reporting function.

Other

The Hsync routine has been defined and implemented to synchronize a file with its image in memory. Currently it is not very useful because the HDF software includes no buffering mechanism and the two images are always identical. Hsync will become useful when buffering is implemented:

Hsync Synchronizes the stored version of an HDF file with the image in memory

1. These are the general compression functions. Additional compression functions, specific to each compression style, can be found in the compression style-specific modules in the source code distribution. As of HDF Version 1.4r4, those modules included the files c\*.c (e.g., cdeflate.c., crle.c) in the directory ./hdf/src/. [↑](#footnote-ref-1)