Happendable

intn Happendable(int32 *h_id*)

h_id	IN:	Access identifier returned by **Hstartwrite**

Purpose Specifies that the specified element can be appended to

Return value Returns `SUCCEED` (or `0`) if data element can be appended and `FAIL` (or `-1`) otherwise.

Description If a data element is at the end of a file **Happendable** allows **Hwrite** to append data to it, converting it to linked-block element only when necessary.

Hcache

intn Hcache(int32 *file_id*, intn *cache_switch*)

file_id	IN:	File identifier returned by **Hopen**
cache_switch	IN:	Flag to enable or disable caching

Purpose Enables low-level caching for the specified file.

Return value Returns `SUCCEED` (or `0`) if successful and `FAIL` (or `-1`) otherwise.

Description If *file_id* is set to `CACHE_ALL_FILES`, then the value of *cache_switch* is used to modify the default file cache setting.

Valid values for *cache_switch* are: `TRUE` (or `1`) to enable caching and `FALSE` (or `0`) to disable caching.

Hdeldd

intn Hdeldd(int32 *file_id*, uint16 *tag*, uint16 *ref*)

file_id	IN:	File identifier returned by **Hopen**
tag	IN:	Tag of data descriptor to be deleted
ref	IN:	Reference number of data descriptor to be deleted

Purpose Deletes a tag and reference number from the data descriptor list.

Return value Returns `SUCCEED` (or `0`) if successful and `FAIL` (or `-1`) otherwise.

Description Once the data descriptor is removed, the data in the data object becomes inaccessible and is marked as such. To remove inaccessible data from an HDF file, use the utility `hdfpack`.

Hdeldd only deletes the specified tag and reference number from the data descriptor list. Data objects containing the deleted tag and reference number are not automatically updated. For example, if the tag and reference number deleted from the descriptor list referenced an object in a vgroup, the tag and reference number will still exist in the vgroup even though the data is inaccessible.

Hendaccess

intn Hendaccess(int32 *h_id*)

h_id	IN:	Access identifier returned by **Hstartread**, **Hstartwrite**, or **Hnextread**

Purpose Terminates access to a data object by disposing of the access identifier.

Return value Returns `SUCCEED` (or `0`) if successful and `FAIL` (or `-1`) otherwise.

Description The number of active access identifiers is limited to `MAX_ACC` as defined in the `hlimits.h` header file. Because of this restriction, it is very important to call **Hendaccess** immediately following the last operation on a data element.

When developing new interfaces, a common mistake is to omit calling **Hendaccess** for all of the elements accessed. When this happens, **Hclose** will return `FAIL`, and a dump of the error stack will report the number of active access identifiers. Refer to the Reference Manual page on **HEprint**.

This is a difficult problem to debug because the low levels of the HDF library cannot determine who and where an access identifier was originated. As a result, there is no automated method of determining which access identifiers have yet to be released.

Hendbitaccess

intn Hendbitaccess(int32 *h_id*, intn *flushbit*)

h_id	IN:	Identifier of the bit-access element to be disposed of
flushbit	IN:	Specifies how the leftover bits are to be flushed

Purpose Disposes of the specified bit-access file element.

Return value Returns `SUCCEED` (or `0`) if successful and `FAIL` (or `-1`) otherwise.

Description If called after a bit-write operation, **Hendbitaccess** flushes all buffered bits to the dataset, then calls **Hendaccess**.

"Leftover bits" are bits that have been buffered, but are fewer than the number of bits defined by `BITNUM`, which is usually set to `8`.

Valid codes for *flushbit* are: 0 for flush with zeros, 1 for flush with ones and -1 for dispose of leftover bits

Hexist

intn Hexist(int32 *h_id*, uint16 *search_tag*, uint16 *search_ref*)

h_id	IN:	Access identifier returned by **Hstartread**, **Hstartwrite**, or **Hnextread**
search_tag	IN:	Tag of the object to be searched for
search_ref	IN:	Reference number of the object to be searched for

Purpose Locates an object in an HDF file.

Return value Returns `SUCCEED` (or `0`) if successful and `FAIL` (or `-1`) otherwise.

Description Simple interface to **Hfind** that determines if a given tag/reference number pair exists in a file. Wildcards apply.

Hfind performs all validity checking; this is just a *very* simple wrapper around it.

Hfidinquire

intn Hfidinquire(int32 *file_id*, char **filename*, intn **access*, intn **attach*)

file_id	IN:	File identifier returned by **Hopen**
filename	OUT:	Complete path and filename for the file
access	OUT:	Access mode file is opened with
attach	OUT:	Number of access identifiers attached to the file

Purpose Returns file information through a reference of its file identifier.

Return value Returns `SUCCEED` (or `0`) if successful and `FAIL` (or `-1`) otherwise.

Description Gets the complete path name, access mode, and number of access identifiers associated with a file. The *filename* parameter is a pointer to a character pointer which will be modified when the function returns. Upon completion, *filename* is set to point to the file name in internal storage. All output parameters must be non-null pointers.

Hfind

intn Hfind(int32 *file_id*, uint16 *search_tag*, uint16 *search_ref*, uint16 **find_tag*, uint16 **find_ref*, int32 **find_offset*, int32 **find_length*, intn *direction*)

file_id	IN:	File identifier returned by **Hopen**
search_tag	IN:	The tag to search for or `DFTAG_WILDCARD`
search_ref	IN:	Reference number to search for or `DFREF_WILDCARD`
find_tag	IN/OUT:	If (**find_tag* == 0) and (**find_ref* == 0) then start the search from either the beginning or the end of the file. If the object is found, the tags of the object will be returned here.
find_ref	IN/OUT:	If (**find_tag* == 0) and (**find_ref* == 0) then start the search from either the beginning or the end of the file. If the object is found, the reference numbers of the object will be returned here.
find_offset	OUT:	Offset of the data element found
find_length	OUT:	Length of the data element found
direction	IN:	Direction to search in `DF_FORWARD` searches forward from the current location, and `DF_BACKWARD` searches backward from the current location

Purpose Locates the next object to be searched for in an HDF file.

Return value Returns `SUCCEED` (or `0`) if successful and `FAIL` (or `-1`) otherwise.

Description **Hfind** searches for the next data element that matches the specified tag and reference number. Wildcards apply. If *direction* is `DF_FORWARD`, searching is forward from the current position in the file, otherwise `DF_BACKWARD` specifies backward searches from the current position in the file.

If *find_tag* and *find_ref* are both set to `0`, this indicates the beginning of a search, and the search will start from the beginning of the file if the direction is `DF_FORWARD` and from the end of the file if the direction is `DF_BACKWARD`.

Hgetbit

intn Hgetbit(int32 *h_id*)

h_id	IN:	Bit-access element identifier

Purpose Reads one bit from the specified bit-access element.

Return value Returns the bit read (or `0` or `1`) if successful and `FAIL` (or `-1`) otherwise.

Description This function is a wrapper for **Hbitread**.

Hgetelement

int32 Hgetelement(int32 *file_id*, uint16 *tag*, uint16 *ref*, uint8 **data*)

file_id	IN:	File identifier returned by **Hopen**
tag	IN:	Tag of the data element to be read
ref	IN:	Reference number of the data element to be read
data	OUT:	Buffer the element will be read into

Purpose Reads the data element for the specified tag and reference number and writes it to the *data* buffer.

Return value Returns the number of bytes read if successful and `FAIL` (or `-1`) otherwise.

Description It is assumed that the space allocated for the buffer is large enough to hold the data.

Hinquire

intn Hinquire(int32 *h_id*, int32 **file_id*, uint16 **tag*, uint16 **ref*, int32 **length*, int32 **offset*, int32 **position*, int16 **access*, int16 **special*)

h_id	IN:	Access identifier returned by **Hstartread**, **Hstartwrite**, or **Hnextread**
file_id	OUT:	File identifier returned by **Hopen**
tag	OUT:	Tag of the element pointed to
ref	OUT:	Reference number of the element pointed to
length	OUT:	Length of the element pointed to
offset	OUT:	Offset of the element in the file
position	OUT:	Current position within the data element
access	OUT:	The access type for this data element
special	OUT:	Special code

Purpose Returns access information about a data element.

Return value Returns `SUCCEED` (or `0`) if the access identifier points to a valid data element and `FAIL` (or `-1`) otherwise.

Description If *h_id* is a valid access identifier the access type (read or write) is set regardless of whether or not the return value is `FAIL` (or `-1`). If *h_id* is invalid, the function returns `FAIL` (or `-1`) and the access type is set to zero. To avoid excess information, pass `NULL` for any unnecessary pointer.

Hlength

int32 Hlength(int32 *file_id*, uint16 *tag*, uint16 *ref*)

file_id	IN:	File identifier returned by **Hopen**
tag	IN:	Tag of the data element
ref	IN:	Reference number of the data element

Purpose Returns the length of a data object specified by the tag and reference number.

Return value Returns the length of data element if found and `FAIL` (or `-1`) otherwise.

Description **Hlength** calls **Hstartread**, **HQuerylength**, and **Hendaccess** to determine the length of a data element. **Hlength** uses **Hstartread** to obtain an access identifier for the specified data object.

Hlength will return the correct data length for linked-block elements, however it is important to remember that the data in linked-block elements is not stored contiguously.

Hnewref

uint16 Hnewref(int32 *file_id*)

file_id	IN:	File identifier returned by **Hopen**

Purpose Returns a reference number that can be used with any tag to produce a unique tag /reference number pair.

Return value Returns the reference number if successful and `0` otherwise.

Description Successive calls to **Hnewref** will generate reference number values that increase by one each time until the highest possible reference number has been returned. At this point, additional calls to **Hnewref** will return an increasing sequence of unused reference number values starting from `1`.

Hnextread

intn Hnextread(int32 *h_id*, uint16 *tag*, uint16 *ref*, int *origin*)

h_id	IN:	Access identifier returned by **Hstartread** or previous **Hnextread**
tag	IN:	Tag to search for
ref	IN:	Reference number to search for
origin	IN:	Position to begin search: `DF_START` or `DF_CURRENT`

Purpose Searches for the next data descriptor that matches the specified tag and reference number.

Return value Returns `SUCCEED` (or `0`) if successful and `FAIL` (or `-1`) otherwise.

Description Wildcards apply. If origin is `DF_START`, the search will start at the beginning of the data descriptor list. If origin is `DF_CURRENT`, the search will begin at the current position. Searching backwards from the end of a data descriptor list is not yet implemented.

If the search is successful, the access identifier reflects the new data element, otherwise it is not modified.

Hnumber/hnumber

int32 Hnumber(int32 file_id, uint16 *tag*)

file_id	IN:	File identifier returned by **Hopen**
tag	IN:	Tag to be counted

Purpose Returns the number of instances of a tag in a file.

Return value Returns the number of instances of a tag in a file if successful, and `FAIL` (or `-1`) otherwise.

Description **Hnumber** determines how many objects with the specified tag are in a file. To determine the total number of objects in a file, set the *tag* argument to `DFTAG_WILDCARD`. Note that a return value of zero is not a fail condition.

FORTRAN

```
integer function hnumber(file_id, tag)

integer file_id, tag
```

Hoffset

int32 Hoffset(int32 *file_id*, uint16 *tag*, uint16 *ref*)

file_id	IN:	File identifier returned by **Hopen**
tag	IN:	Tag of the data element
ref	IN:	Reference number of the data element

Purpose Returns the offset of a data element in the file.

Return value Returns the offset of the data element if the data element exists and `FAIL` (or `-1`) otherwise.

Description **Hoffset** calls **Hstartread**, **HQueryoffset**, and **Hendaccess** to determine the length of a data element. **Hoffset** uses **Hstartread** to obtain an access identifier for the specified data object.

Hoffset will return the correct offset for a linked-block element, however it is important to remember that the data in linked-block elements is not stored contiguously. The offset returned by **Hoffset** only reflects the position of the first data block.

Hoffset should not be used to determine the offset of an external element. In this case, **Hoffset** returns zero, an invalid offset for HDF files.

Hputbit

intn Hputbit(int32 *h_id*, intn *bit*)

h_id	IN:	Bit-access element identifier
bit	IN:	Bit to be written

Purpose Writes one bit to the specified bit-access element.

Return value Returns `SUCCEED` (or `0`) if successful and `FAIL` (or `-1`) otherwise.

Description This function is a wrapper for **Hbitwrite**.

Hputelement

int32 Hputelement(int32 *file_id*, uint16 *tag*, uint16 *ref*, uint8 **data*, int32 *length*)

file_id	IN:	File identifier returned by **Hopen**
tag	IN:	Tag of the data element to add or replace
ref	IN:	Reference number of the data element to add or replace
data	IN:	Pointer to data buffer
length	IN:	Length of data to write

Purpose Writes a data element or replaces an existing data element in a HDF file.

Return value Returns the number of bytes written if successful and `FAIL` (or `-1`) otherwise.

Hread

int32 Hread(int32 *h_id*, int32 *length*, VOIDP *data*)

h_id	IN:	Access identifier returned by **Hstartread**, **Hstartwrite**, or **Hnextread**
length	IN:	Length of segment to be read
data	OUT:	Pointer to the data array to be read

Purpose Reads the next segment in a data element.

Return value Returns the length of segment actually read if successful and `FAIL` (or `-1`) otherwise.

Description **Hread** begins reading at the current file position, reads the specified number of bytes, and increments the current file position by one. Calling **Hread** with the *length* = `0` reads the entire data element. To reposition an access identifier before writing data, use **Hseek**.

If *length* is longer than the data element, the read operation is terminated at the end of the data element, and the number of read bytes is returned. Although only one access identifier is allowed per data element, it is possible to interlace reads from multiple data elements in the same file. It is assumed that data is large enough to hold the specified data length.

Hseek

intn Hseek(int32 *h_id*, int32 *offset*, intn *origin*)

h_id	IN:	Access identifier returned by **Hstartread**, **Hstartwrite**, or **Hnextread**
offset	IN:	Number of bytes to seek to from the origin
origin	IN:	Position of the offset origin

Purpose Sets the access pointer to an offset within a data element.

Return value Returns `SUCCEED` (or `0`) if successful and `FAIL` (or `-1`) otherwise.

Description Sets the seek position for the next **Hread** or **Hwrite** operation by moving an access identifier to the specified position in a data element. The *origin* and the *offset* arguments determine the byte location for the access identifier. If *origin* is set to `DF_START`, the offset is added to the beginning of the data element. If *origin* is set to `DF_CURRENT`, the offset is added to the current position of the access identifier.

Valid values for *origin* are: `DF_START` (the beginning of the file) or `DF_CURRENT` (the current position in the file).

This routine fails if the access identifier if *h_id* is invalid or if the seek position is outside the range of the data element.

Hsetlength

int32 Hsetlength(int32 *file_id*, int32 *length*)

file_id	IN:	File identifier returned by **Hopen**
length	IN:	Length of the new element

Purpose Specifies the length of a new HDF element.

Return value Returns `SUCCEED` (or `0`) if successful and `FAIL` (or `-1`) otherwise.

Description This function can only be used when called after **Hstartaccess** on a new data element and before any data is written to that element.

Hshutdown

int32 Hshutdown( )

Purpose Deallocates buffers previously allocated in other H routines.

Return value Returns `SUCCEED` (or `0`) if successful and `FAIL` (or `-1`) otherwise.

Description Should only be called by the function **HDFend**.

Htagnewref

int32 Htagnewref(int32 *file_id*, uint16 *tag*)

file_id	IN:	Access identifier returned by **Hstartread** or **Hnextread**
tag	IN:	Tag to be identified with the returned reference number

Purpose Returns a reference number that is unique for the specified file that will correspond to the specified tag. Creates a new tag/reference number pair.

Return value Returns the reference number if successful and `0` otherwise.

Description Successive calls to **Hnewref** will generate a increasing sequence of reference number values until the highest possible reference number value has been returned. It will then return unused reference number values starting from `1` in increasing order.

Htrunc

int32 Htrunc(int32 *h_id*, int32 *trunc_len*)

h_id	IN:	Access identifier returned by **Hstartread** or **Hnextread**
trunc_len	IN:	Length to truncate element

Purpose Truncates the data object specified by the *h_id* to the length *trunc_len*.

Return value Returns the length of a data element if found and `FAIL` (or `-1`) otherwise.

Description **Htrunc** does not handle special elements.

Hwrite

int32 Hwrite(int32 *h_id*, int32 *length*, VOIDP *data*)

h_id	IN:	Access identifier returned by **Hstartwrite**
len	IN:	Length of segment to be written
data	IN:	Pointer to the data to be written

Purpose Writes the next data segment to a specified data element.

Return value Returns the length of the segment actually written if successful and `FAIL` (or `-1`) otherwise.

Description **Hwrite** begins writing at the current position of the access identifier, writes the specified number of bytes, then moves the access identifier to the position immediately following the last accessed byte. Calling **Hwrite** with *length* = `0` results in an error condition. To reposition an access identifier before writing data, use **Hseek**.

If the space allocated in the data element is smaller than the length of data, the data is truncated to the length of the data element. Although only one access identifier is allowed per data element, it is possible to interlace writes to more than one data element in a file.

HDFclose/hdfclose

intn HDFclose(int32 *file_id*)

file_id	IN:	File identifier returned by **Hopen**

Purpose Closes the access path to the file.

Return value Returns `SUCCEED` (or `0`) if successful and `FAIL` (or `-1`) otherwise.

Description The file identifier *file_id* is validated before the file is closed. If the identifier is valid, the function closes the access path to the file.

If there are still access identifiers attached to the file, the error code `DFE_OPENAID` is returned and the file is not closed. This is a common occurrence when developing new interfaces. See **Hendaccess** for further discussion of this problem.

FORTRAN

```
integer function hdfclose(file_id)

integer file_id
```

HDFopen/hdfopen

int32 HDFopen(char **filename*, intn *access*, int16 *n_dds*)

filename	IN:	Complete path and filename for the file to be opened
access	IN:	File access code
n_dds	IN:	Number of data descriptors in a block if a new file is to be created

Purpose Provides an access path to an HDF file by reading all the data descriptor blocks into memory.

Return value Returns the file identifier if successful and `FAIL` (or `-1`) otherwise.

Description If given a new file name, **HDFopen** will create a new file using the specified access type and number of data descriptors. If given an existing file name, **HDFopen** will open the file using the specified access type and ignore the *n_dds* argument.

HDF provides several file access code definitions:

DFACC_READ - Open for read only. If file does not exist, an error condition results.
DFACC_CREATE - If file exists, delete it, then open a new file for read/write.
DFACC_WRITE - Open for read/write. If file does not exist, create it.

If a file is opened and an attempt is made to reopen the file using `DFACC_CREATE`, HDF will issue the error `DFE_ALROPEN`. If the file is opened with read only access and an attempt is made to reopen the file for write access using `DFACC_RDWR`, `DFACC_WRITE`, or `DFACC_ALL`, HDF will attempt to reopen the file with read and write permissions.

Upon successful exit, the named file is opened with the relevant permissions, the data descriptors are set up in memory, and the associated *file_id* is returned. For new files, the appropriate file headers are also set up.

FORTRAN

```
integer function hdfopen(filename, access, n_dds)

character*(*) filename

integer access, n_dds
```

HEclear

VOID HEclear( )

Purpose Clears all information on reported errors from the error stack.

Return value None.

Description **HEpush** creates an error stack. **HEclear** is then used to clear this stack after any errors are processed.

HEpush

VOID HEpush(int16 *error_code*, char **funct_name*, char **file_name*, intn *line*)

error_code	IN:	HDF error code corresponding to the error
funct_name	IN:	Name of function in which the error occurred
file_name	IN:	Name of file in which the error occurred
line	IN:	Line number in the file that error occurred

Purpose Pushes a new error onto the error stack.

Return value None.

Description **HEpush** pushes the file name, function name, line number, and generic description of the error onto the error stack. **HEreport** can then be used to give a more case-specific description of the error.

If the stack is full, the error is ignored. **HEpush** assumes that the character strings *funct_name* and *file_name* are in semi-permanent storage, so only pointers to the strings are saved.

HEreport

VOID HEreport(char **format*, ...)

format	IN:	Output string specification

Purpose Adds a text string to the description of the most-recently-reported error (only one text string per error).

Return value None

Description **HEpush** places on the error stack the file name, function name, line number, and a generic description of the error type. **HEreport** can then be used to give a more case-specific description of the error. Only one additional annotation can be attached to each error report.

The format argument must conform to the string specification requirements of `printf`.

HEvalue

int16 HEvalue(int32 *level*)

level	IN:	Level of the error stack to be returned

Purpose Returns an error from the specified level of the error stack.

Return value The error code if successful for `DFE_NONE` otherwise.