H5O_VISIT1

Expand all   Collapse all

- Jump to ...
- Summary
- Description
- Example
- Switch language ...
- C
- C++
- FORTRAN
- JAVA

Summary
Description
Example
JAVA
FORTRAN
C++
C

H5O_VISIT1

Recursively visits all objects accessible from a specified object

As of HDF5-1.12 this function has been deprecated in favor of the function H5O_VISIT3 or the macro H5O_VISIT.

Procedure:

H5O_VISIT1 (obj_id, idx_type, order, op, op_data)

Signature:

```
herr_t H5Ovisit1 ( hid_t obj_id, H5_index_t idx_type, H5_iter_order_t order, H5O_iterate1_t op, void
*op_data )
```

```
SUBROUTINE h5ovisit_f(object_id, index_type, order, op, op_data, &
            return_value, hdferr)
    INTEGER(HID_T), INTENT(IN) :: object_id
    INTEGER, INTENT(IN) :: index_type
    INTEGER, INTENT(IN) :: order

    TYPE(C_FUNPTR):: op
    TYPE(C_PTR)    :: op_data
    INTEGER, INTENT(OUT) :: return_value
    INTEGER, INTENT(OUT) :: hdferr
```

Parameters:

hid_t obj_id	IN: Location identifier of the object at which the recursive iteration begins; may be a file, group, dataset, named datatype or attribute identifier
H5_index_t idx_type	IN: Type of index; valid values include: H5_INDEX_NAME H5_INDEX_CRT_ORDER
H5_iter_order_t order	IN: Order in which index is traversed; valid values include: H5_ITER_DEC H5_ITER_INC H5_ITER_NATIVE
H5O_iterate1_t op	IN: Callback function passing data regarding the object to the calling application
void *op_data	IN: User-defined pointer to data required by the application for its processing of the object

Description:

H5O_VISIT1 is a recursive iteration function to visit the object `obj_id` and, if `obj_id` is a group, all objects in and below it in an HDF5 file, thus providing a mechanism for an application to perform a common set of operations across all of those objects or a dynamically selected subset. For non-recursive iteration across the members of a group, see H5L_ITERATE1.

If `obj_id` is a group identifier, that group serves as the root of a recursive iteration. If `obj_id` is a file identifier, that file's root group serves as the root of the recursive iteration. If `obj_id` is any other type of object, such as a dataset or named datatype, there is no iteration.

Two parameters are used to establish the iteration: `idx_type` and `order`.

`idx_type` specifies the index to be used. If the links in a group have not been indexed by the index type, they will first be sorted by that index then the iteration will begin; if the links have been so indexed, the sorting step will be unnecessary, so the iteration may begin more quickly. Valid values include the following:

H5_INDEX_NAME	Alpha-numeric index on name
H5_INDEX_CRT_ORDER	Index on creation order

Note that the index type passed in `idx_type` is a *best effort* setting. If the application passes in a value indicating iteration in creation order and a group is encountered that was not tracked in creation order, that group will be iterated over in alpha-numeric order by name, or *name order*. (*Nam e order* is the native order used by the HDF5 library and is always available.)

`order` specifies the order in which objects are to be inspected along the index specified in `idx_type`. Valid values include the following:

H5_ITER_INC	Increasing order
H5_ITER_DEC	Decreasing order
H5_ITER_NATIVE	Fastest available order

The prototype of the callback function `op` is as follows (as defined in the source code file `H5Opublic.h`):

```
typedef herr_t (*H5O_iterate1_t)(hid_t obj, const char *name, const H5O_info1_t *info, void *op_data);
```

The parameters of this callback function have the following values or meanings:

obj	Object that serves as root of the iteration; same value as the H5O_VISIT1 `obj_id` parameter
name	Name of object, relative to `obj`, being examined at current step of the iteration
info	*H5O_info1_t* struct containing information regarding that object
op_data	User-defined pointer to data required by the application in processing the object

The *H5O_info1_t* struct is defined in `H5Opublic.h`:

src / H5Opublic.h [255:273] hdf5_1_12 HDFFV/hdf5

```
/* Information struct for object */
/* (For H5Oget_info/H5Oget_info_by_name/H5Oget_info_by_idx versions 1 & 2) */
typedef struct H5O_info1_t {
    unsigned long  fileno;  /* File number that object is located in */
    haddr_t   addr;  /* Object address in file */
    H5O_type_t   type;  /* Basic object type (group, dataset, etc.) */
    unsigned   rc;  /* Reference count of object    */
    time_t  atime;  /* Access time   */
    time_t  mtime;  /* Modification time  */
    time_t  ctime;  /* Change time   */
    time_t  btime;  /* Birth time   */
    hsize_t   num_attrs; /* # of attributes attached to object */
    H5O_hdr_info_t      hdr;            /* Object header information */
    /* Extra metadata storage for obj & attributes */
    struct {
        H5_ih_info_t   obj;             /* v1/v2 B-tree & local/fractal heap for
groups, B-tree for chunked datasets */
        H5_ih_info_t   attr;            /* v2 B-tree & heap for attributes */
    } meta_size;
```

The return values from an operator are:

- Zero causes the visit iterator to continue, returning zero when all group members have been processed.
- A positive value causes the visit iterator to immediately return that positive value, indicating short-circuit success.
- A negative value causes the visit iterator to immediately return that value, indicating failure.

The H5O_VISIT1 `op_data` parameter is a user-defined pointer to the data required to process objects in the course of the iteration. This pointer is passed back to each step of the iteration in the callback function’s `op_data` parameter.

H5L_VISIT1 and H5O_VISIT1 are companion functions: one for examining and operating on links; the other for examining and operating on the objects that those links point to. Both functions ensure that by the time the function completes successfully, every link or object below the specified point in the file has been presented to the application for whatever processing the application requires. These functions assume that the membership of the group being iterated over remains unchanged through the iteration; if any of the links in the group change during the iteration, the resulting behavior is undefined.

Programming Note for C++ Developers Using C Functions:

If a C routine that takes a function pointer as an argument is called from within C++ code, the C routine should be returned from normally.

Examples of this kind of routine include callbacks such as H5P_SET_ELINK_CB and H5P_SET_TYPE_CONV_CB and functions such as H5T_CONVERT and H5E_WALK2.

Exiting the routine in its normal fashion allows the HDF5 C library to clean up its work properly. In other words, if the C++ application jumps out of the routine back to the C++ “catch” statement, the library is not given the opportunity to close any temporary data structures that were set up when the routine was called. The C++ application should save some state as the routine is started so that any problem that occurs might be diagnosed.

Returns:

On success, returns the return value of the first operator that returns a positive value, or zero if all members were processed with no operator returning non-zero.

On failure, returns a negative value if something goes wrong within the library, or the first negative value returned by an operator.

Example:

History:

Release	Change
1.10.5	The macro H5O_VISIT was removed and the function H5O_VISIT1 was copied to H5O_VISIT
1.10.3	Function H5O_VISIT was copied to H5O_VISIT1, and the macro H5O_VISIT was created.
1.8.8	Fortran subroutine and data structure added.
1.8.0	C function introduced.

--- Last Modified: February 19, 2020 | 01:07 PM