H5O_VISIT_BY_NAME1

Expand all Collapse all

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

H5O_VISIT_BY_NAME1

Recursively visits all objects starting from a specified object

Procedure:

H5O_VISIT_BY_NAME1(loc_id, object_name, index_type, order, op, op_data, lapl_id)

Signature:

```
herr_t H5Ovisit_by_name1( hid_t loc_id, const char *object_name, H5_index_t index_type, H5_iter_order_t
order,H5O_iterate_t op, void *op_data, hid_t lapl_id )
```

```
SUBROUTINE h5ovisit_by_name_f(loc_id, object_name, index_type, order, &
       op, op_data, return_value, hdferr, lapl_id)
    USE, INTRINSIC :: ISO_C_BINDING
    IMPLICIT NONE
    INTEGER(HID_T)  , INTENT(IN)            :: loc_id
    CHARACTER(LEN=*), INTENT(IN)            :: object_name
    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
    INTEGER(HID_T)  , INTENT(IN) , OPTIONAL  :: lapl_id
```

Parameters:

hid_t `loc_id`	IN: Location identifier; may be a file, group, dataset, named datatype or attribute identifier
const char *`object_name`	IN: Name of the object, generally relative to `loc_id`, that will serve as root of the iteration

H5_index_t `index_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_iterate_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
hid_t `lapl_id`	IN: Link access property list identifier

Description:

H5O_VISIT_BY_NAME1 is a recursive iteration function to visit the object specified by the `loc_id` / `object_name` parameter pair and, if that object 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_ITERATE.

The object serving as the root of the iteration is specified by the `loc_id` / `object_name` parameter pair. `loc_id` specifies a file or an object in a file; `object_name` specifies either an object in the file (with an absolute name based in the file's root group) or an object name relative to `loc_id`. If `loc_id` fully specifies the object that is to serve as the root of the iteration, `object_name` should be `'.'` (a dot). (Note that when `loc_id` fully specifies the the object that is to serve as the root of the iteration, the user may wish to consider using H5O_VISIT instead of H5O_VISIT_BY_NAME.)

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

`index_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 unnecesary, 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 `index_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*. (*Name 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 `index_type`. Valid values include the following:

`H5_ITER_INC`	Increasing order
`H5_ITER_DEC`	Decreasing order
`H5_ITER_NATIVE`	Fastest available order

The op callback function and the effect of the callback function's return value on the application are described in H5O_VISIT.

The *H5O_info_t* struct is defined in `H5Opublic.h` and described in the H5O_GET_INFO function entry.

The H5O_VISIT_BY_NAME1 `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.

`lapl_id` is a link access property list. In the general case, when default link access properties are acceptable, this can be passed in as `H5P_DEFAULT`. An example of a situation that requires a non-default link access property list is when the link is an external link; an external link may require that a link prefix be set in a link access property list (see H5P_SET_ELINK_PREFIX).

H5L_VISIT_BY_NAME and H5O_VISIT_BY_NAME 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.

> **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:

Coming Soon!

History:

Release	Change
1.10.3	Function renamed to `H5Ovisit_by_name1`.
1.8.11	Fortran subroutine introduced in this release.
1.8.0	Function introduced in this release.

--- Last Modified: April 25, 2019 | 01:31 PM