H5L_CREATE_EXTERNAL

H5L_CREATE_EXTERNAL

Creates an external link, a soft link to an object in a different file

Procedure:

H5L_CREATE_EXTERNAL(target_file_name, target_obj_name, link_loc_id, link_name, lcpl_id, lapl_id))

Signature:

```
herr_t H5Lcreate_external( const char *target_file_name, const char *target_obj_name, hid_t link_loc_id,
const char *link_name, hid_t lcpl_id, hid_t lapl_id )
```

```
SUBROUTINE h5lcreate_external_f(file_name, obj_name, link_loc_id, link_name, &
                         hdferr, lcpl_id, lapl_id)
  IMPLICIT NONE
  CHARACTER(LEN=*), INTENT(IN) :: file_name
                        ! Name of the file containing the target object. Neither
                        ! the file nor the target object is required to exist.
                        ! May be the file the link is being created in.
  CHARACTER(LEN=*), INTENT(IN) :: obj_name
                        ! Name of the target object, which need not already exist.
  INTEGER(HID_T), INTENT(IN) :: link_loc_id
                        ! The file or group identifier for the new link.
  CHARACTER(LEN=*), INTENT(IN) :: link_name
                        ! The name of the new link.
  INTEGER, INTENT(OUT) :: hdferr
                        ! Error code:
                        ! 0 on success and -1 on failure
  INTEGER(HID_T), OPTIONAL, INTENT(IN) :: lcpl_id
                        ! Link creation property list identifier.
  INTEGER(HID_T), OPTIONAL, INTENT(IN) :: lapl_id
                        ! Link access property list identifier.
END SUBROUTINE h5lcreate_external_f
```

Parameters:

const char * `target_file_name`	IN: Name of the target file containing the target object
const char *`target_obj_name`	IN: Path within the target file to the target object
hid_t `link_loc_id`	IN: Location identifier where the new link is to be created; may be a file, group, dataset, named datatype or attribute identifier
const char * `link_name`	IN: Name of the new link, relative to `link_loc_id`
hid_t `lcpl_id`	IN: Link creation property list identifier
hid_t `lapl_id`	IN: Link access property list identifier

Description:

H5L_CREATE_EXTERNAL creates a new external link. An external link is a soft link to an object in a different HDF5 file from the location of the link, i.e., to an external object.

`target_file_name` identifies the target file containing the target object; `target_obj_name` specifies the path of the target object within that file. `target_obj_name` must be an absolute pathname in `target_file_name`, i.e., it must start at the target file's root group, but it is not interpreted until an application attempts to traverse it.

`link_loc_id` and `link_name` specify the location and name, respectively, of the new link. `link_name` is interpreted relative to `link_loc_id`

`lcpl_id` is the link creation property list used in creating the new link.

`lapl_id` is the link access property list used in traversing the new link. Note that an external file opened by the traversal of an external link is always opened with the weak file close degree property setting, `H5F_CLOSE_WEAK` (see H5P_SET_FCLOSE_DEGREE); any file close degree property setting in `lapl_id` is ignored.

An external link behaves similarly to a soft link, and like a soft link in an HDF5 file, it may *dangle*: the target file and object need not exist at the time that the external link is created.

When the external link `link_name` is accessed, the library will search for the target file `target_file_name` as described below:

- If `target_file_name` is a relative pathname, the following steps are performed:
  - The library will get the prefix(es) set in the environment variable `HDF5_EXT_PREFIX` and will try to prepend each prefix to `target_file_name` to form a new `target_file_name`.
  - If the new `target_file_name` does not exist or if `HDF5_EXT_PREFIX` is not set, the library will get the prefix set via H5P_SET_ELINK_PREFIX and prepend it to `target_file_name` to form a new `target_file_name`.
  - If the new `target_file_name` does not exist or no prefix is being set by H5P_SET_ELINK_PREFIX, then the path of the file associated with `link_loc_id` is obtained. This path can be the absolute path or the current working directory plus the relative

path of that file when it is created/opened. The library will prepend this path to `target_file_name` to form a new `target_file_name`.
- If the new `target_file_name` does not exist, then the library will look for `target_file_name` and will return failure/success accordingly.

- If `target_file_name` is an absolute pathname, the library will first try to find `target_file_name`. If `target_file_name` does not exist, `target_file_name` is stripped of directory paths to form a new `target_file_name`. The search for the new `target_file_name` then follows the same steps as described above for a relative pathname. See examples below illustrating how `target_file_name` is stripped to form a new `target_file_name`.

Note that `target_file_name` is considered to be an absolute pathname when the following condition is true:

- For Unix, the first character of `target_file_name` is a slash (`/`).

  For example, consider a `target_file_name` of `/tmp/A.h5`. If that target file does not exist, the new `target_file_name` after stripping will be `A.h5`.
- For Windows, there are 6 cases:
  1. `target_file_name` is an absolute drive with absolute pathname.

     For example, consider a `target_file_name` of `/tmp/A.h5`. If that target file does not exist, the new `target_file_name` after stripping will be `A.h5`.
  2. `target_file_name` is an absolute pathname without specifying drive name.

     For example, consider a `target_file_name` of `/tmp/A.h5`. If that target file does not exist, the new `target_file_name` after stripping will be `A.h5`.
  3. `target_file_name` is an absolute drive with relative pathname.

     For example, consider a `target_file_name` of `/tmp/A.h5`. If that target file does not exist, the new `target_file_name` after stripping will be `tmp\A.h5`.
  4. `target_file_name` is in UNC (Uniform Naming Convention) format with server name, share name, and pathname.

     For example, consider a `target_file_name` of `/tmp/A.h5`. If that target file does not exist, the new `target_file_name` after stripping will be `A.h5`.
  5. `target_file_name` is in Long UNC (Uniform Naming Convention) format with server name, share name, and pathname.

     For example, consider a `target_file_name` of `/tmp/A.h5`. If that target file does not exist, the new `target_file_name` after stripping will be `A.h5`
  6. `target_file_name` is in Long UNC (Uniform Naming Convention) format with an absolute drive and an absolute pathname.

     For example, consider a `target_file_name` of `/tmp/A.h5`. If that target file does not exist, the new `target_file_name` after stripping will be `A.h5`

The library opens target file `target_file_name` with the file access property list that is set via H5P_SET_ELINK_FAPL when the external link `link_name` is accessed. If no such property list is set, the library uses the file access property list associated with the file of `link_loc_id` to open the target file.

If an application requires additional control over file access flags or the file access property list, see H5P_SET_ELINK_CB; this function enables the use of an external link callback function as described in H5L_ELINK_TRAVERSE_T.

Restriction: A file close degree property setting (H5P_SET_FCLOSE_DEGREE) in the external link file access property list or in the external link callback function will be ignored. A file opened by means of traversing an external link is always opened with the weak file close degree property setting, `H5F_CLOSE_WEAK`.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Example:

Coming Soon!

History:

Release	Change
1.8.0	Function introduced in this release.

--- Last Modified: April 25, 2019 | 12:36 PM