NAME

nis_names, nis_lookup, nis_add, nis_remove, nis_modify, nis_freeresult – NIS+ namespace functions

SYNOPSIS

cc [*flag* …] *file…* **-lnsl** [*library…*]

#include <rpcsvc/nis.h>

nis_result *nis_lookup(const nis_name *name*, **const u_long** *flags***);**

nis_result *nis_add(const nis_name *name*, **const nis_object** **obj***);**

nis_result *nis_remove(const nis_name *name*, **const nis_object** **obj***);**

nis_result *nis_modify(const nis_name *name*, **const nis_object** **obj***);**

void nis_freeresult(nis_result **result***);**

MT-LEVEL

MT-Safe

DESCRIPTION

These functions are used to locate and manipulate all NIS+ objects (see **nis_objects**(3N)) except the NIS+ entry objects. To look up the NIS+ entry objects within a NIS+ table, refer to **nis_subr**(3N).

nis_lookup( ) resolves a NIS+ name and returns a copy of that object from a NIS+ server. **nis_add( )** and **nis_remove( )** add and remove objects to the NIS+ namespace, respectively. **nis_modify( )** can change specific attributes of an object that already exists in the namespace.

These functions should be used only with names that refer to an NIS+ Directory, NIS+ Table, NIS+ Group, or NIS+ Private object. If a name refers to an NIS+ entry object, the functions listed in **nis_subr**(3N) should be used.

nis_freeresult( ) frees all memory associated with a **nis_result** structure. This function must be called to free the memory associated with a NIS+ result. **nis_lookup( )**, **nis_add( ), nis_remove( ),** and **nis_modify( )** all return a pointer to a **nis_result structure** which *must* be freed by calling **nis_freeresult( )** when you have finished using it. If one or more of the objects returned in the structure need to be retained, they can be copied with **nis_clone_object**(3N) (see **nis_subr**(3N)).

nis_lookup( ) takes two parameters, the name of the object to be resolved in *name*, and a flags parameter, *flags*, which is defined below. The object name is expected to correspond to the syntax of a non-indexed NIS+ name (see **nis_tables**(3N)). The **nis_lookup( )** function is the *only* function from this group that can use a non-fully qualified name. If the parameter *name* is not a fully qualified name, then the flag **EXPAND_NAME** *must* be specified in the call. If this flag is not specified, the function will fail with the error NIS_BADNAME.

The *flags* parameter is constructed by logically ORing zero or more flags from the following list.

FOLLOW_LINKS When specified, the client library will ''follow'' links by issuing another NIS+ lookup call for the object named by the link. If the linked object is itself a link, then this process will iterate until the either a object is found that is not a *LINK* type object, or the library has followed 16 links.

HARD_LOOKUP When specified, the client library will retry the lookup until it is answered by a server. Using this flag will cause the library to block until at least one NIS+ server is available. If the network connectivity is impaired, this can be a relatively long time.

NO_CACHE When specified, the client library will bypass any object caches and will get the object from either the master NIS+ server or one of its replicas.

MASTER_ONLY When specified, the client library will bypass any object caches and any domain replicas and fetch the object from the NIS+ master server for the object's domain. This insures that the object returned is up to date at the cost of a possible performance degradation and failure if the master server is unavailable or physically distant.

EXPAND_NAME When specified, the client library will attempt to expand a partially qualified name by calling the function **nis_getnames( )** (see **nis_subr**(3N)) which uses the environment variable **NIS_PATH**.

The status value may be translated to ascii text using the function **nis_sperrno( )** (see **nis_error**(3N)).

On return, the *objects* array in the result will contain one and possibly several objects that were resolved by the request. If the FOLLOW_LINKS flag was present, on success the function could return several entry objects if the link in question pointed within a table. If an error occurred when following a link, the objects array will contain a copy of the link object itself.

The function **nis_add( )** will take the object *obj* and add it to the NIS+ namespace with the name *name*. This operation will fail if the client making the request does not have the *create* access right for the domain in which this object will be added. The parameter *name* must contain a fully qualified NIS+ name. The object members *zo_name* and *zo_domain* will be constructed from this name. This operation will fail if the object already exists. This feature prevents the accidental addition of objects over another object that has been added by another process.

The function **nis_remove( )** will remove the object with name *name* from the NIS+ namespace. The client making this request must have the *destroy* access right for the domain in which this object resides. If the named object is a link, the link is removed and *not* the object that it points to. If the parameter *obj* is not NULL, it is assumed to point to a copy of the object being removed. In this case, if the object on the server does not have the same object identifier as the object being passed, the operation will fail with the NIS_NOTSAMEOBJ error. This feature allows the client to insure that it is removing the desired object. The parameter *name* must contain a fully qualified NIS+ name.

The function **nis_modify( )** will modify the object named by *name* to the field values in the object pointed to by *obj*. This object should contain a copy of the object from the name space that is being modified. This operation will fail with the error NIS_NOTSAMEOBJ if the object identifier of the passed object does not match that of the object being modified in the namespace.

Note: Normally the contents of the member *zo_name* in the *nis_object* structure would be constructed from the name passed in the *name* parameter. However, if it is non-NULL the client library will use the name in the *zo_name* member to perform a rename operation on the object. This name *must not* contain any unquoted '.'(dot) characters. If these conditions are not met the operation will fail and return the NIS_BADNAME error code.

Results

These functions return a pointer to a structure of type **nis_result**:

```
struct nis_result {
        nis_error status;
        struct {
                        u_int objects_len;
                        nis_object *objects_val;
        } objects;
        netobj          cookie;
        u_long          zticks;
        u_long          dticks;
        u_long          aticks;
        u_long          cticks;
};
```

The *status* member contains the error status of the the operation. A text message that describes the error can be obtained by calling the function **nis_sperrno( )** (see **nis_error**(3N)).

The *objects* structure contains two members. *objects_val* is an array of *nis_object* structures; *objects_len* is the number of cells in the array. These objects will be freed by the call to **nis_freeresult( )**. If you need to keep a copy of one or more objects, they can be copied with the function **nis_clone_object( )** and freed with the function **nis_destroy_object( )** (see **nis_server**(3N)). Refer to **nis_objects**(3N) for a description of the **nis_object** structure.

The various ticks contain details of where the time was taken during a request. They can be used to tune one's data organization for faster access and to compare different database implementations (see **nis_db**(3N)).

zticks The time spent in the NIS+ service itself. This count starts when the server receives the request and stops when it sends the reply.

dticks The time spent in the database backend. This time is measured from the time a database call starts, until the result is returned. If the request results in multiple calls to the database, this is the sum of all the time spent in those calls.

aticks The time spent in any ''accelerators'' or caches. This includes the time required to locate the server needed to resolve the request.

cticks The total time spent in the request. This clock starts when you enter the client library and stops when a result is returned. By subtracting the sum of the other ticks values from this value, you can obtain the local overhead of generating a NIS+ request.

Subtracting the value in *dticks* from the value in *zticks* will yield the time spent in the service code itself. Subtracting the sum of the values in *zticks* and *aticks* from the value in *cticks* will yield the time spent in the client library itself. Note: all of the tick times are measured in microseconds.

RETURN VALUES

The client library can return a variety of error returns and diagnostics. The more salient ones are documented below.

NIS_SUCCESS The request was successful.

NIS_S_SUCCESS The request was successful, however the object returned came from an object cache and not directly from the server. If you do not wish to see objects from object caches you must specify the flag **NO_CACHE** when you call the lookup function.

NIS_NOTFOUND The named object does not exist in the namespace.

NIS_CACHEEXPIRED The object returned came from an object cache taht has *expired*. The time to live value has gone to zero and the object may have changed. If the flag **NO_CACHE** was passed to the lookup function then the lookup function will retry the operation to get an unexpired copy of the object.

NIS_NAMEUNREACHABLE
A server for the directory of the named object could not be reached. This can occur when there is a network partition or all servers have crashed. See the **HARD_LOOKUP** flag.

NIS_UNKNOWNOBJ The object returned is of an unknown type.

NIS_TRYAGAIN The server connected to was too busy to handle your request. For the *add*, *remove*, and *modify* operations this is returned when either the master server for a directory is unavailable or it is in the process of checkpointing its database. It can also be returned when the server is updating it's internal state. And in the case of **nis_list( )** if the client specifies a callback and the server does not have enough resources to handle the callback.

NIS_SYSTEMERROR A generic system error occurred while attempting the request. Most commonly the server has crashed or the database has become corrupted. Check the syslog record for error messages from the server.

NIS_NOT_ME A request was made to a server that does not serve the name in question. Normally this will not occur, however if you are not using the built in location mechanism for servers you may see this if your mechanism is broken.

NIS_NOMEMORY Generally a fatal result. It means that the service ran out of heap space.

NIS_NAMEEXISTS An attempt was made to add a name that already exists. To add the name, first remove the existing name and then add the new object or modify the existing named object.

NIS_NOTMASTER An attempt was made to update the database on a replica server.

NIS_INVALIDOBJ The object pointed to by *obj* is not a valid **NIS+** object.

NIS_BADNAME The name passed to the function is not a legal **NIS+** name.

NIS_LINKNAMEERROR The name passed resolved to a *LINK* type object and the contents of the link pointed to an invalid name.

NIS_NOTSAMEOBJ An attempt to remove an object from the namespace was aborted because the object that would have been removed was not the same object that was passed in the request.

NIS_NOSUCHNAME This hard error indicates that the named directory of the table object does not exist. This occurs when the server that should be the parent of the server that serves the table, does not know about the directory in which the table resides.

NIS_NOSUCHTABLE The named table does not exist.

NIS_MODFAIL The attempted modification failed.

NIS_FOREIGNNS The name could not be completely resolved. When the name passed to the function would resolve in a namespace that is outside the NIS+ name tree, this error is returned with a NIS+ object of type **DIRECTORY**, which contains the type of namespace and contact information for a server within that namespace.

NIS_RPCERROR This fatal error indicates the RPC subsystem failed in some way. Generally there will be a **syslog**(3) message indicating why the RPC request failed.

ENVIRONMENT

NIS_PATH If the flag **EXPAND_NAME** is set, this variable is the search path used by **nis_lookup( )**.

SEE ALSO

nis_error(3N), **nis_objects**(3N), **nis_server**(3N), **nis_subr**(3N), **nis_tables**(3N)

NOTES

You cannot modify the name of an object if that modification would cause the object to reside in a different domain.

You cannot modify the schema of a table object.