NAME

nis+, NIS+, nis – a new version of the network information name service

DESCRIPTION

NIS+ is a new version of the network information nameservice. This version differs in several significant ways from version 2, which is referred to as NIS or YP in earlier releases. Specific areas of enhancement include the ability to scale to larger networks, security, and the administration of the service.

The man pages for NIS+ are broken up into three basic categories. Those in section 1 are the user commands that are most often executed from a shell script or directly from the command line. Section 1M man pages describe utility commands that can be used by the network administrator to administer the service itself. The NIS+ programming API is described by man pages in section 3N.

All commands and functions that use NIS version 2 are prefixed by the letters **yp** as in **ypmatch**(1), **ypcat**(1), **yp_match**(3N), and **yp_first**(3N). Commands and functions that use the new replacement software NIS+ are prefixed by the letters **nis** as in **nismatch**(1), **nischown**(1), **nis_list**(3N), and **nis_add_entry**(3N). A complete list of NIS+ commands is in the **LIST OF COMMANDS** section.

This man page introduces the NIS+ terminology. It also describes the NIS+ namespace, authentication, and authorization policies.

NIS+ NAMESPACE

The naming model of NIS+ is based upon a tree structure. Each node in the tree corresponds to an NIS+ object. There are six types of NIS+ objects: *directory*, *table*, *group*, *link*, *entry*, and *private*.

NIS+ Directory Object

Each NIS+ namespace will have at least one NIS+ directory object. An NIS+ directory is like a UNIX file system directory which contains other NIS+ objects including NIS+ directories. The NIS+ directory that forms the root of the NIS+ namespace is called the root directory. There are two special NIS+ directories: **org_dir** and **groups_dir**. The **org_dir** directory consists of all the system-wide administration tables, such as **passwd**, **hosts**, and **mail_aliases**. The **groups_dir** directory consists of NIS+ group objects which are used for access control. The collection of **org_dir**, **groups_dir** and their parent directory is referred to as an NIS+ domain. NIS+ directories can be arranged in a tree-like structure so that the NIS+ namespace can match the organizational or administrative hierarchy.

NIS+ Table Object

NIS+ tables (not files), contained within NIS+ directories, store the actual information about some particular type. For example, the **hosts** system table stores information about the IP address of the hosts in that domain. NIS+ tables are multicolumn and the tables can be searched through any of the searchable columns. Each table object defines the schema for its table. The NIS+ tables consist of NIS+ entry objects. For each entry in the NIS+ table, there is an NIS+ entry object. NIS+ entry objects conform to the schema defined by the NIS+ table object.

NIS+ Group Object

NIS+ group objects are used for access control at group granularity. NIS+ group objects, contained within the **groups_dir** directory of a domain, contain a list of all the NIS+ principals within a certain NIS+ group. An NIS+ principal is a user or a machine making NIS+ requests.

NIS+ Link Object

NIS+ link objects are like UNIX symbolic file-system links—they are typically used for shortcuts in the NIS+ namespace.

Refer to **nis_objects**(3N) for more information about the NIS+ objects.

NIS+ NAMES

The NIS+ service defines two forms of names, *simple* names and *indexed* names. Simple names are used by the service to identify NIS+ objects contained within the NIS+ namespace. Indexed names are used to identify NIS+ entries contained within NIS+ tables. Furthermore, entries within NIS+ tables are returned to the caller as NIS+ objects of type *entry*. NIS+ objects are implemented as a union structure which is described in the file **<rpcsvc/nis_object.x>**. The differences between the various types and the meanings of the components of these objects are described in **nis_objects**(3N).

Simple Names

Simple names consist of a series of labels that are separated by the '.'(dot) character. Each label is composed of printable characters from the ISO Latin 1 set. Each label can be of any nonzero length, provided that the fully qualified name is fewer than **NIS_MAXNAMELEN** octets including the separating dots. (See **<rpcsvc/nis.h>** for the actual value of **NIS_MAXNAMELEN** in the current release.) Labels that contain special characters (see **Grammar**) must be quoted.

The NIS+ namespace is organized as a singly rooted tree. Simple names identify nodes within this tree. These names are constructed such that the leftmost label in a name identifies the leaf node and all of the labels to the right of the leaf identify that object's parent node. The parent node is referred to as the leaf's *directory*. This is a naming directory and should not be confused with a file system directory.

For example, the name *example.simple.name.* is a simple name with three labels, where *example* is the leaf node in this name, the directory of this leaf is *simple.name.* which by itself is a simple name. The leaf of which is *simple* and its directory is simply *name*.

The function **nis_leaf_of**(3N) returns the first label of a simple name. The function **nis_domain_of**(3N) returns the name of the directory that contains the leaf. Iterative use of these two functions can break a simple name into each of its label components.

The name '.' (dot) is reserved to name the *global root* of the namespace. For systems that are connected to the Internet, this global root will be served by a Domain Name Service. When an NIS+ server is serving a root directory whose name is not '.'(dot) this directory is referred to as a *local root*.

NIS+ names are said to be *fully qualified* when the name includes all of the labels identifying all of the directories, up to the global root. Names without the trailing dot are called *partially* qualified.

Indexed Names

Indexed names are compound names that are composed of a search criterion and a simple name. The search criterion component is used to select entries from a table; the simple name component is used to identify the NIS+ table that is to be searched. The search criterion is a series of column names and their desired values enclosed in bracket ‘**[ ]**’ characters. These criteria take the following form:

[*column_name***=***value***,** *column_name***=***value***, ...]**

A search criterion is combined with a simple name to form an indexed name by concatenating the two parts, separated by a ‘,’(comma) character as follows.

[*search-criterion* **],***table.directory.*

When multiple column name/value pairs are present in the search criterion, only those entries in the table that have the appropriate value in all columns specified are returned. When no column name/value pairs are specified in the search criterion, **[ ]**, *all* entries in the table are returned.

Grammar

The following text represents a context-free grammar that defines the set of legal NIS+ names. The terminals in this grammar are the characters ‘.’ (dot), ‘[’ (open bracket), ‘]’ (close bracket), ‘,’ (comma), ‘=’ (equals) and whitespace. Angle brackets (‘<’ and ‘>’), which delineate non-terminals, are not part of the grammar. The character ‘|’ (vertical bar) is used to separate alternate productions and should be read as ‘‘this production **OR** this production’’.

name	::=	. | <*simple name*> | <*indexed name*>
simple name	::=	<*string*>. | <*string*>.<*simple name*>
indexed name	::=	<*search criterion*>,<*simple name*>
search criterion	::=	[<*attribute list*>]
attribute list	::=	<*attribute*> | <*attribute*>,<*attribute list*>
attribute	::=	<*string*> = <*string*>
string	::=	ISO Latin 1 character set except the character ’/’ (slash). The initial character may not be a terminal character or the characters ’@’ (at), ’+’ (plus), or (‘–’) hyphen.

Terminals that appear in strings must be quoted with ‘"’ (double quote). The ‘"’ character may be quoted by quoting it with itself ‘""’.

Name Expansion

The NIS+ service only accepts fully qualified names. However, since such names may be unwieldy, the NIS+ commands in section 1 employ a set of standard expansion rules that will attempt to fully qualify a partially qualified name. This expansion is actually done by the NIS+ library function **nis_getnames**(3N) which generates a list of names using the default NIS+ directory search path or the **NIS_PATH** environment variable. The default NIS+ directory search path includes all the names in its path. **nis_getnames( )** is invoked by the functions **nis_lookup**(3N) and **nis_list**(3N) when the **EXPAND_NAME** flag is used.

The **NIS_PATH** environment variable contains an ordered list of simple names. The names are separated by the ':' (colon) character. If any name in the list contains colons, the colon should be quoted as described in the **Grammar** section. When the list is exhausted, the resolution function returns the error **NIS_NOTFOUND**. This may mask the fact that the name existed but a server for it was unreachable. If the name presented to the list or lookup interface is fully qualified, the **EXPAND_NAME** flag is ignored.

In the list of names from the **NIS_PATH** environment variable, the '$' (dollar sign) character is treated specially. Simple names that end with the label '$' have this character replaced by the default directory (see **nis_local_directory**(3N)). Using "$" as a name in this list results in this name being replaced by the list of directories between the default directory and the global root that contain at least two labels.

Below is an example of this expansion. Given the default directory of *some.long.domain.name.*, and the **NIS_PATH** variable set to **fred.bar.:org_dir.$:$**. This path is initially broken up into the list:

1 **fred.bar.**

2 **org_dir.$**

3 **$**

The dollar sign in the second component is replaced by the default directory. The dollar sign in the third component is replaced with the names of the directories between the default directory and the global root that have at least two labels in them. The effective path value becomes:

1 **fred.bar.**

2a **org_dir.some.long.domain.name.**

3a **some.long.domain.name.**

3b **long.domain.name.**

3c **domain.name.**

Each of these simple names is appended to the partially qualified name that was passed to the **nis_lookup**(3N) or **nis_list**(3N) interface. Each is tried in turn until **NIS_SUCCESS** is returned or the list is exhausted.

If the NIS_PATH variable is not set, the path ''$'' is used.

The library function **nis_getnames**(3N) can be called from user programs to generate the list of names that would be attempted. The program **nisdefaults**(1) with the –**s** option can also be used to show the fully expanded path.

Concatenation Path

Normally all the entries for a certain type of information are stored within the table itself. However, there are times when it is desirable for the table to point to other tables where entries can be found. For example, you may want to store all the IP addresses in the host table for their own domain, and yet want to be able to resolve hosts in some other domain without explicitly specifying the new domain name. NIS+ provides a mechanism for concatenating different but related tables with a "NIS+ Concatenation Path". With a concatenation path, you can create a sort of flat namespace from a hierarchical structure. You can also create a table with no entries and just point the hosts or any other table to its parent domain. Note that with such a setup, you are moving the administrative burden of managing the tables to the parent domain. The concatenation path will slow down the request response time because more tables and more servers are searched. It will also decrease the availability if all the servers are incapacitated for a particular directory in the table path.

The NIS+ Concatenation Path is also referred to as the "table path". This path is set up at table creation time through **nistbladm**(1). You can specify more than one table to be concatenated and they will be searched in the given order. Note that the NIS+ client libraries, by default, will not follow the concatenation path set in site-specific tables. Refer to **nis_list**(3N) for more details.

Namespaces

The NIS+ service defines two additional *disjoint* namespaces for its own use. These namespaces are the NIS+ *Principal* namespace, and the NIS+ *Group* namespace. The names associated with the group and principal namespaces are syntactically identical to simple names. However, the information they represent *cannot* be obtained by directly presenting these names to the NIS+ interfaces. Instead, special interfaces are defined to map these names into NIS+ names so that they may then be resolved.

Principal Names

NIS+ principal names are used to uniquely identify users and machines that are making NIS+ requests. These names have the form:

principal.domain

Here *domain* is the fully qualified name of an NIS+ directory where the named principal's credentials can be found. See **Directories and Domains** for more information on domains. Note that in this name, *principal*, is not a leaf in the NIS+ namespace.

Credentials are used to map the identity of a host or user from one context such as a process UID into the NIS+ context. They are stored as records in an NIS+ table named *cred*, which always appears in the *org_dir* subdirectory of the directory named in the principal name.

This mapping can be expressed as a replacement function:

principal.domain –>**[cname**=*principal.domain* **],cred.org_dir**.*domain*

This latter name is an NIS+ name that can be presented to the **nis_list**(3N) interface for resolution. NIS+ principal names are administered using the **nisaddcred**(1M) command.

The *cred* table contains five columns named *cname*, *auth_name*, *auth_type*, *public_data*, and *private_data*. There is one record in this table for each identity mapping for an NIS+ principal. The current service supports two such mappings:

LOCAL This mapping is used to map from the UID of a given process to the NIS+ principal name associated with that UID. If no mapping exists, the name *nobody* is returned. When the effective UID of the process is 0 (for example, the superuser), the NIS+ name associated with the host is returned. Note that UIDs are sensitive to the context of the machine on which the process is executing.

DES This mapping is used to map to and from a Secure RPC ‘‘netname’’ into an NIS+ principal name. See **secure_rpc**(3N) for more information on netnames.

Note that since netnames contain the notion of a domain, they span NIS+ directories.

The NIS+ client library function **nis_local_principal**(3N) uses the *cred.org_dir* table to map the UNIX notion of an identity, a process' UID, into an NIS+ principal name. Shell programs can use the program **nisdefaults**(1) with the **-p** switch to return this information.

Mapping from UIDs to an NIS+ principal name is accomplished by constructing a query of the form:

[auth_type=LOCAL, auth_name=*uid***],cred.org_dir.***default-domain***.**

This query will return a record containing the NIS+ principal name associated with this UID, in the machine's default domain.

The NIS+ service uses the DES mapping to map the names associated with Secure RPC requests into NIS+ principal names. RPC requests that use Secure RPC include the *netname* of the client making the request in the RPC header. This netname has the form:

unix.*UID@domain*

The service constructs a query using this name of the form:

[auth_type=DES, auth_name=*netname***],cred.org_dir.***domain***.**

where the domain part is extracted from the netname rather than using the default domain. This query is used to look up the mapping of this netname into an NIS+ principal name in the domain where it was created.

This mechanism of mapping UID and netnames into an NIS+ principal name guarantees that a client of the NIS+ service has only one principal name. This principal name is used as the basis for authorization which is described below. All objects in the NIS+ namespace and all entries in NIS+ tables must have an owner specified for them. This owner field always contains an NIS+ principal name.

Group Names

Like NIS+ principal names, NIS+ group names take the form:

group_name**.domain**

All objects in the NIS+ namespace and all entries in NIS+ tables may optionally have a *group owner* specified for them. This group owner field, when filled in, always contains the fully qualified NIS+ group name.

The NIS+ client library defines several interfaces (**nis_groups**(3N)) for dealing with NIS+ groups. These interfaces internally map NIS+ group names into an NIS+ simple name which identifies the NIS+ group object associated with that group name. This mapping can be shown as follows:

group.domain **–>** *group***.groups_dir***.domain*

This mapping eliminates collisions between NIS+ group names and NIS+ directory names. For example, without this mapping, a directory with the name *engineering.foo.com.*, would make it impossible to have a group named *engineering.foo.com.*. This is due to the restriction that within the NIS+ namespace, a name unambiguously identifies a single object. With this mapping, the NIS+ *group* name *engineering.foo.com.* maps to the NIS+ *object* name *engineering.groups_dir.foo.com.*

The contents of a group object is a list of NIS+ principal names, and the names of other NIS+ groups. See **nis_groups**(3N) for a more complete description of their use.

NIS+ SECURITY

NIS+ defines a security model to control access to information managed by the service. The service defines access rights that are selectively granted to individual clients or groups of clients. Principal names and group names are used to define clients and groups of clients that may be granted or denied access to NIS+ information. These principals and groups are associated with NIS+ domains as defined below.

The security model also uses the notion of a class of principals called *nobody*, which contains all clients, whether or not they have authenticated themselves to the service. The class *world* includes any client who has been authenticated.

Directories and Domains

Some directories within the NIS+ namespace are referred to as NIS+ *Domains*. Domains are those NIS+ directories that contain the subdirectories *groups_dir* and *org_dir*. Further, the subdirectory *org_dir* should contain the table named *cred*. NIS+ Group names and NIS+ Principal names **always** include the NIS+ domain name after their first label.

Authentication

The NIS+ name service uses Secure RPC for the integrity of the NIS+ service. This requires that users of the service and their machines must have a Secure RPC key pair associated with them. This key is initially generated with either the **nisaddcred**(1M) or **nisclient**(1M) commands and modified with the **chkey**(1) or **nispasswd**(1) commands.

The use of Secure RPC allows private information to be stored in the name service that will not be available to untrusted machines or users on the network.

In addition to the Secure RPC key, users need a mapping of their UID into an NIS+ principal name. This mapping is created by the system administrator using the **nisclient**(1M) or **nisaddcred**(1M) command.

Users that will be using machines in several NIS+ domains must insure that they have a *local* credential entry in each of those domains. This credential should be created with the NIS+ principal name of the user in their ''home'' domain. For the purposes of NIS+ and Secure RPC, the home domain is defined to be the one where your Secure RPC key pair is located.

Authorization

The NIS+ service defines four access rights that can be granted or denied to clients of the service. These rights are *read*, *modify*, *create*, and *destroy*. These rights are specified in the object structure at creation time and may be modified later with the **nischmod**(1) command. In general, the rights granted for an object apply only to that object. However, for purposes of authorization, rights granted to clients reading *directory* and *table* objects are granted to those clients for all of the objects ''contained'' by the parent object. This notion of containment is abstract. The objects do not actually contain other objects within them. Note that *group* objects do contain the list of principals within their definition.

Access rights are interpreted as follows:

read This right grants read access to an object. For directory and table objects, having read access on the parent object conveys read access to all of the objects that are direct children of a directory, or entries within a table.

modify — This right grants modification access to an existing object. Read access is not required for modification. However, in many applications, one will need to read an object before modifying it. Such modify operations will fail unless read access is also granted.

create — This right gives a client permission to create new objects where one had not previously existed. It is only used in conjunction with directory and table objects. Having create access for a table allows a client to add additional entries to the table. Having create access for a directory allows a client to add new objects to an NIS+ directory.

destroy — This right gives a client permission to destroy or remove an existing object or entry. When a client attempts to destroy an entry or object by removing it, the service first checks to see if the table or directory containing that object grants the client destroy access. If it does, the operation proceeds, if the containing object does not grant this right then the object itself is checked to see if it grants this right to the client. If the object grants the right, then the operation proceeds; otherwise the request is rejected.

Each of these rights may be granted to any one of four different categories.

owner — A right may be granted to the *owner* of an object. The owner is the NIS+ principal identified in the owner field. The owner can be changed with the **nischown**(1) command. Note that if the owner does not have modification access rights to the object, the owner cannot change any access rights to the object, unless the owner has modification access rights to its parent object.

group owner — A right may be granted to the *group owner* of an object. This grants the right to any principal that is identified as a member of the group associated with the object. The group owner may be changed with the **nischgrp**(1) command. The object owner need not be a member of this group.

world — A right may be granted to everyone in the *world*. This grants the right to all clients who have authenticated themselves with the service.

nobody — A right may be granted to the *nobody* principal. This has the effect of granting the right to any client that makes a request of the service, regardless of whether they are authenticated or not.

Note that for bootstrapping reasons, directory objects that are NIS+ domains, the *org_dir* subdirectory and the *cred* table within that subdirectory must have *read* access to the *nobody* principal. This makes navigation of the namespace possible when a client is in the process of locating its credentials. Granting this access does not allow the contents of other tables within *org_dir* to be read (such as the entries in the password table) unless the table itself gives "real" access rights to the *nobody* principal.

Directory Authorization

Additional capabilities are provided for granting access rights to clients for directories. These rights are contained within the *object access rights* (OAR) structure of the directory. This structure allows the NIS+ service to grant rights that are not granted by the directory object to be granted for objects contained by the directory of a specific type.

An example of this capability is a directory object which does not grant create access to all clients, but does grant create access in the OAR structure for *group* type objects to clients who are members of the NIS+ group associated with the directory. In this example the only objects that could be created as children of the directory would have to be of the type *group*.

Another example is a directory object that grants create access only to the owner of the directory, and then additionally grants create access through the OAR structure for objects of type *table*, *link*, *group*, and *private* to any member of the directory's group. This has the effect of giving nearly complete create access to the group with the exception of creating subdirectories. This restricts the creation of new NIS+ domains because creating a domain requires creating both a *groups_dir* and *org_dir* subdirectory.

Note that there is currently no command line interface to set or change the OAR of the directory object.

Table Authorization

As with directories, additional capabilities are provided for granting access to entries within tables. Rights granted to a client by the access rights field in a table object apply to the table object and all of the entry objects ''contained'' by that table. If an access right is not granted by the table object, it may be granted by an entry within the table. This holds for all rights except *create*.

For example, a table may not grant read access to a client performing a **nis_list**(3N) operation on the table. However, the access rights field of entries within that table may grant read access to the client. Note that access rights in an entry are granted to the owner and group owner of the *entry* and not the owner or group of the table. When the list operation is performed, all entries that the client has read access to are returned. Those entries that do not grant read access are not returned. If none of the entries that match the search criterion grant read access to the client making the request, no entries are returned and the result status contains the NIS_NOTFOUND error code.

Access rights that are granted by the rights field in an entry are granted for the entire entry. However, in the table object an additional set of access rights is maintained for each column in the table. These rights apply to the equivalent column in the entry. The rights are used to grant access when neither the table nor the entry itself grant access. The access rights in a column specification apply to the owner and group owner of the entry rather than the owner and group owner of the table object.

When a read operation is performed, if read access is not granted by the table and is not granted by the entry but *is* granted by the access rights in a column, that entry is returned with the correct values in all columns that are readable and the string ∗**NP**∗ (No Permission) in columns where read access is not granted.

As an example, consider a client that has performed a list operation on a table that does not grant read access to that client. Each entry object that satisfied the search criterion specified by the client is examined to see if it grants read access to the client. If it does, it is included in the returned result. If it does not, then each column is checked to see if it grants read access to the client. If any columns grant read access to the client, data in those columns is returned. Columns that do not grant read access have their contents replaced by the string ∗**NP**∗. If none of the columns grant read access, then the entry is not returned.

LIST OF COMMANDS

The following lists all commands and programming functions related to NIS+:

NIS+ User Commands

nisaddent(1M)	add /etc files and NIS maps into their corresponding NIS+ tables
niscat(1)	display NIS+ tables and objects
nischgrp(1)	change the group owner of a NIS+ object
nischmod(1)	change access rights on a NIS+ object
nischown(1)	change the owner of a NIS+ object
nischttl(1)	change the time to live value of a NIS+ object
nisdefaults(1)	display NIS+ default values
niserror(1)	display NIS+ error messages
nisgrep(1)	utilities for searching NIS+ tables
nisgrpadm(1)	NIS+ group administration command
nisln(1)	symbolically link NIS+ objects
nisls(1)	list the contents of a NIS+ directory
nismatch(1)	utilities for searching NIS+ tables
nismkdir(1)	create NIS+ directories
nispasswd(1)	change NIS+ password information
nisrm(1)	remove NIS+ objects from the namespace
nisrmdir(1)	remove NIS+ directories
nisshowcache(1M)	NIS+ utility to print out the contents of the shared cache file
nistbladm(1)	NIS+ table administration command
nistest(1)	return the state of the NIS+ namespace using a conditional expression

NIS+ Administrative Commands

aliasadm(1M)	manipulate the NIS+ aliases map
nis_cachemgr(1M)	NIS+ utility to cache location information about NIS+ servers
nisaddcred(1M)	create NIS+ credentials
nisaddent(1M)	create NIS+ tables from corresponding /etc files or NIS maps
nisclient(1M)	initialize NIS+ credentials for NIS+ principals
nisd(1M)	NIS+ service daemon
nisd_resolv(1M)	NIS+ service daemon
nisinit(1M)	NIS+ client and server initialization utility
nislog(1M)	display the contents of the NIS+ transaction log
nisping(1M)	send ping to NIS+ servers
nispopulate(1M)	populate the NIS+ tables in a NIS+ domain
nisserver(1M)	set up NIS+ servers
nissetup(1M)	initialize a NIS+ domain
nisshowcache(1M)	NIS+ utility to print out the contents of the shared cache file

	nisstat(1M)	report NIS+ server statistics
	nisupdkeys(1M)	update the public keys in a NIS+ directory object
	rpc.nisd(1M)	NIS+ service daemon
	rpc.nisd_resolv(1M)	NIS+ service daemon
	sysidnis(1M)	system configuration
NIS+ Programming API	**__nis_map_group**(3N)	NIS+ group manipulation functions
	db_add_entry(3N)	NIS+ Database access functions
	db_checkpoint(3N)	NIS+ Database access functions
	db_create_table(3N)	NIS+ Database access functions
	db_destroy_table(3N)	NIS+ Database access functions
	db_first_entry(3N)	NIS+ Database access functions
	db_free_result(3N)	NIS+ Database access functions
	db_initialize(3N)	NIS+ Database access functions
	db_list_entries(3N)	NIS+ Database access functions
	db_next_entry(3N)	NIS+ Database access functions
	db_remove_entry(3N)	NIS+ Database access functions
	db_reset_next_entry(3N)	NIS+ Database access functions
	db_standby(3N)	NIS+ Database access functions
	db_table_exists(3N)	NIS+ Database access functions
	db_unload_table(3N)	NIS+ Database access functions
	nis_add(3N)	NIS+ namespace functions
	nis_add_entry(3N)	NIS+ table functions
	nis_addmember(3N)	NIS+ group manipulation functions
	nis_checkpoint(3N)	misc NIS+ log administration functions
	nis_clone_object(3N)	NIS+ subroutines
	nis_creategroup(3N)	NIS+ group manipulation functions
	nis_db(3N)	NIS+ Database access functions
	nis_destroy_object(3N)	NIS+ subroutines
	nis_destroygroup(3N)	NIS+ group manipulation functions
	nis_dir_cmp(3N)	NIS+ subroutines
	nis_domain_of(3N)	NIS+ subroutines
	nis_error(3N)	display NIS+ error messages
	nis_first_entry(3N)	NIS+ table functions
	nis_freenames(3N)	NIS+ subroutines
	nis_freeresult(3N)	NIS+ namespace functions
	nis_freeservlist(3N)	miscellaneous NIS+ functions
	nis_freetags(3N)	miscellaneous NIS+ functions
	nis_getnames(3N)	NIS+ subroutines
	nis_getservlist(3N)	miscellaneous NIS+ functions
	nis_groups(3N)	NIS+ group manipulation functions
	nis_ismember(3N)	NIS+ group manipulation functions
	nis_leaf_of(3N)	NIS+ subroutines
	nis_lerror(3N)	display some NIS+ error messages
	nis_list(3N)	NIS+ table functions

nis_local_directory(3N)	NIS+ local names
nis_local_group(3N)	NIS+ local names
nis_local_host(3N)	NIS+ local names
nis_local_names(3N)	NIS+ local names
nis_local_principal(3N)	NIS+ local names
nis_lookup(3N)	NIS+ namespace functions
nis_map_group(3N)	NIS+ group manipulation functions
nis_mkdir(3N)	miscellaneous NIS+ functions
nis_modify(3N)	NIS+ namespace functions
nis_modify_entry(3N)	NIS+ table functions
nis_name_of(3N)	NIS+ subroutines
nis_names(3N)	NIS+ namespace functions
nis_next_entry(3N)	NIS+ table functions
nis_objects(3N)	NIS+ object formats
nis_perror(3N)	display NIS+ error messages
nis_ping(3N)	misc NIS+ log administration functions
nis_print_group_entry(3N)	NIS+ group manipulation functions
nis_print_object(3N)	NIS+ subroutines
nis_remove(3N)	NIS+ namespace functions
nis_remove_entry(3N)	NIS+ table functions
nis_removemember(3N)	NIS+ group manipulation functions
nis_rmdir(3N)	miscellaneous NIS+ functions
nis_server(3N)	miscellaneous NIS+ functions
nis_servstate(3N)	miscellaneous NIS+ functions
nis_sperrno(3N)	display NIS+ error messages
nis_sperror(3N)	display NIS+ error messages
nis_sperror_r(3N)	display NIS+ error messages
nis_stats(3N)	miscellaneous NIS+ functions
nis_subr(3N)	NIS+ subroutines
nis_tables(3N)	NIS+ table functions
nis_verifygroup(3N)	NIS+ group manipulation functions

NIS+ Files and Directories

nisfiles(4)	NIS+ database files and directory structure

FILES

<rpcsvc/nis_object.x>	protocol description of an NIS+ object
<rpcsvc/nis.x>	defines the NIS+ protocol using the RPC language as described in the *ONC+ Developers Guide*.
<rpcsvc/nis.h>	should be included by all clients of the NIS+ service

SEE ALSO

nischown(1), **nisdefaults**(1), **nismatch**(1), **nispasswd**(1), **admintool**(1M), **newkey**(1M), **nisaddcred**(1M), **nisclient**(1M), **nispopulate**(1M), **nisserver**(1M), **nis_add_entry**(3N), **nis_domain_of**(3N), **nis_getnames**(3N), **nis_groups**(3N), **nis_leaf_of**(3N), **nis_list**(3N), **nis_local_directory**(3N), **nis_lookup**(3N), **nis_objects**(3N)

ONC+ Developers Guide

Describes the application programming interfaces for networks including NIS+

NIS+ and DNS Setup and Configuration Guide

Describes how to plan for and configure an NIS+ namespace

NIS+ and FNS Administration Guide

Describes how to administer a running NIS+ namespace and modify its security

NIS+ Transition Guide

Describes how to make the transition from NIS to NIS+

Solaris Advanced User's Guide

Describes the **admintool**(1M) window interface for modifying the data in NIS+ tables