Administering FNS on NIS+ 6

This chapter describes the setup and administration of the FNS implementation on top of the NIS+ environment. Use the following procedures for a standard setup (in which contexts are created automatically for you). If you wish to set up contexts individually, then specific procedures in “Creating FNS Contexts Individually” on page 73 will apply.

The following sections contain information to aid you in administering the FNS namespace after it has been set up.

6

Setting Up FNS

Setting up FNS involves preparing the NIS+ environment that FNS will use and then creating FNS contexts for organizations, users, hosts, services, and sites. Depending on the size of the organization, you should allow several hours for the FNS setup to be completed, not including the required hardware and software preparation or any NIS+ preparation.

Estimating Resource Requirements

Before proceeding with any installation procedure, you must first ensure that the machines on which NIS+ servers for supporting FNS will run have sufficient memory and disk storage.

For example, to support an FNS environment with 1200 users and hosts, you will need

- A minimum of 20 Mbytes of disk space beyond the space needed for NIS+
- An additional 40 Mbytes of swap space

Setting Up NIS+ Service for FNS

It is recommended, though not required, that NIS+ objects used by FNS and standard NIS+ domain information be supported on separate sets of servers. This avoids placing additional loads on the standard NIS+ service. It also allows you to keep the administration of FNS's use of NIS+ and the standard NIS+ service separate.

All NIS+ objects used by FNS are kept under the `ctx_dir` directory of an NIS+ domain, at the same level as the domain's `org_dir` directory. The NIS+ domain must be already set up before setting up FNS. That is, NIS+ domain tables, such as hosts and passwd, must already exist and be populated.

Before setting up the FNS namespace, do the following:

1. **Set the `NIS_GROUP` environment variable to the name of the group that will be administering the FNS objects.**
   In fact, the `fncreate` command will not let you complete the FNS setup without setting the variable first. In this example, `NIS_GROUP` is set to `fns_admins.wiz.com`. When `fncreate` creates user and host contexts, they will be owned by those hosts and users, and not by the administrator

who executed the command. Setting NIS_GROUP allows the administrators who are members of the group to subsequently modify these contexts even thought they do not own the objects.

```
# set NIS_GROUP=fns_admins.wiz.com;export NIS_GROUP
```

2. **To set up separate servers, create the** ctx_dir **directory for the NIS+ domain. Assign a master server to service it with the NIS+ command** nismkdir**.**
   The example shows how nismkdir creates the ctx_dir directory and assigns the machine fns_mserver as the master server for that directory. Include the trailing dot as shown.

```
# nismkdir -m fns_mserver ctx_dir.wiz.com.
```

3. **Use the** nisls **command to verify that the** ctx_dir **directory has been created.**

```
# nisls wiz.com.
ctx_dir
groups_dir
org_dir
```

Setting Up the FNS Namespace

The FNS namespace is created by the fncreate command. This command creates the contexts for the specified organization and all its subcontexts, including contexts for users and hosts in the NIS+ domain corresponding to the organization.

1. **For a standard setup, use the syntax of `fncreate` as shown.**
   This creates the organization context for the root NIS+ domain, `wiz.com.`, contexts for all users found in the `passwd.org_dir` table, and contexts for all hosts found in the `hosts.org_dir` table in the `wiz.com` NIS+ domain.

```
# fncreate -t org org/wiz.com./
```

   After setting up the FNS namespace, you should checkpoint the `ctx_dir` directory before performing other FNS operations.

2. **Use `nisping` to checkpoint the `ctx_dir` directory:**

```
# /usr/lib/nis/nisping -C ctx_dir.wiz.com.
```

For an organization with a few thousand users and hosts, the initial `fncreate` for an organization will typically take several hours; the subsequent checkpoint will also typically take several hours.

Replicating FNS Service

Additional replicas should be added to serve the `ctx_dir` directory after the FNS namespace has been set up on the master server. Replicas enhance availability and read performance of the servers.

1. **Use the `nismkdir -s` command to add a replica `fns_rserver` for the `ctx_dir` directory and send the contents of the directory to the replica.**

```
# nismkdir -s fns_rserver ctx_dir.wiz.com.
```

2. **Checkpoint the `ctx_dir` directory periodically with the `nisping` command.**
   The recommended period is every few days. The period you choose depends on how frequently changes are made to the FNS namespace.

```
# /usr/lib/nis/nisping -C ctx_dir.wiz.com.
```

At this point, you are done with the initial FNS setup.

Creating FNS Contexts Individually

FNS contexts are created using the `fncreate(1M)` command. This section describes the `fncreate` command and its other options. Use this section to create FNS contexts individually rather than for the entire organization as described in "Setting Up the FNS Namespace" on page 71.

The `fncreate` command has the following syntax,

```
fncreate -t context_type [-f input_file] [-o][-r reference_type][-s][-v] [-D] composite_name
```

where *context_type* specifies one of the following: `org`, `hostname`, `username`, `host`, `user`, `service`, `site`, `nsid`, `generic`, or `fs`.

`fncreate` creates a context of the specified type and binds it to the given composite name. It also creates subcontexts for the context.

Table 6-1 `fncreate` Command Options

Option	Description
`-t`	Specifies the type of context to create.
`-f`	Creates a context for every host or user listed in *input_file*. This option can only be used with the `-t username` or `-t hostname` option and is useful for creating contexts for a subset of users and hosts found in the corresponding NIS+ passwd and hosts tables, respectively.
`-o`	Creates only the context specified[1]. Without the `-o` option, subcontexts are created according to the FNS policies.
`-r`	Specifies the *reference_type* of the generic context being created. It can only be used with the `-t generic` option.
`-s`	Creates new contexts for composite names already in use. Otherwise, no new contexts are created for names already bound.
`-D`	Displays information about the NIS+ object associated with a context each time a context is created. This option is useful for debugging.
`-v`	Displays information about the creation as each context is created.

1. The `org` context is the exception where the contexts for hostname, username, and service are created but not populated.

When creating contexts bound to namespace identifiers, the name without the underscore (for example, `user`) is used to create the context and the name with the underscore (for example, `_user`) is then bound to the reference of the newly created context. The is done regardless of whether the name with or without the underscore is specified in the command line.

For example, the command

```
# fncreate -t username org/sales/_user
```

creates a context for `org/sales/user` and adds a binding for `org/sales/_user` to the context of `org/sales/user`.

Organization Context

Use the org type to create an organization context. The composite name must be that of an existing NIS+ domain. An NIS+ domain is an NIS+ directory with an org_dir subdirectory. Associated host and passwd tables for the domain must also exist.

Assume the root NIS+ domain is wiz.com. In the example

```
# fncreate -t org org/sales/
```

there must be an NIS+ domain named sales. When the new context is created, a ctx_dir directory, if it does not already exist, is created under the directory of the domain, sales.wiz.com. The previous example created an organization context for the composite name org/sales/ and, in addition, created hostname, username, and service subcontexts for it, which in turn, created host and user contexts, and service subcontexts for hosts and users.

Effectively, the following commands are run:

```
fncreate -t hostname org/sales/host/
fncreate -t username org/sales/user/
fncreate -t service org/sales/service/
```

When fncreate -o -t org is used, only the org context is created. The hostname, username, and service contexts are also created but not populated with host and user contexts.

The org context is owned by the administrator who executed the fncreate command, as are the hostname, username, and service subcontexts. The host and user contexts, however, and their subcontexts are owned by the hosts or users for which the contexts were created. In order for the administrator to subsequently manipulate host and user contexts, the NIS_GROUP environment variable must have been set accordingly at the time fncreate is executed. See “Setting Up NIS+ Service for FNS” on page 70 for instructions.

All Hosts Context

The `hostname` type creates a hostname context in which host contexts can be created and bound. Host contexts and their subcontexts are created for each host name found in the NIS+ `hosts.org_dir` table unless the `-o` option is used. When the `-o` option is used, only the `hostname` context is created.

For example, running the command

```
# fncreate -t hostname org/sales/host/
```

creates the `hostname` context and effectively runs the command

```
# fncreate -t hostname org/sales/host/hname/
```

for each host name, *hname*, found in the `hosts.org_dir` table. It also adds a binding for `org/sales/_host/` that is bound to the reference of `org/sales/host/`.

The `hostname` context is owned by the administrator who executed the `fncreate` command. A host context and its subcontexts are owned by the host for which the contexts were created. That is, each host owns its own host context and subcontexts.

The `-f` option can be used to create contexts for a subset of the hosts found in the NIS+ table `hosts.org_dir`. It creates contexts for those hosts listed in the given input file.

Single Host Context

The `host` type creates the context and subcontexts for a host. The command automatically creates a `service` context for the host and a binding for `fs` unless the `-o` option is used. When the `-o` option is used, only the `host` context is created.

For example, the command

```
# fncreate -t host org/sales/host/capsule/
```

creates a context for the host named `capsule` and effectively runs the commands

```
fncreate -t service org/sales/host/capsule/service/
fncreate -t fs org/sales/host/capsule/fs/
```

The `host` context and its subcontexts are owned by the host. In the above example, the host `capsule`, with NIS+ principal name `capsule.sales.wiz.com`, owns the contexts `org/sales/host/capsule/`, `org/sales/host/capsule/service/`, and `org/sales/host/capsule/fs`.

The `hostname` context (`org/sales/host` in the above example) to which the host belongs must already exist. The host name supplied should already exist in the NIS+ `hosts.org_dir` table.

Host Aliases

Alias host names may exist in an NIS+ `hosts.org_dir` table. These appear in the table as a set of hosts with the same canonical name but different alias names.

In FNS, a single host with multiple alias names has a single host context. Alias names for that host in the `hostname` context are bound to the reference of that host context.

All–Users Context

The `username` type creates a username context in which user contexts can be created and bound. User contexts and their subcontexts are created for each user name found in the NIS+ `passwd.org_dir` table unless the `-o` option is used. When the `-o` option is used, only the `username` context is created.

For example, running the command

```
# fncreate -t username org/sales/user/
```

creates the username context and effectively runs the command

```
fncreate -t user org/sales/user/uname/
```

for each user, *uname*, that appears in the passwd.org_dir table. It also adds a binding for org/sales/_user/ that is bound to the reference of org/sales/user/.

The username context is owned by the administrator who executed the fncreate command. A user context and its subcontexts are owned by the user for which the contexts were created. Each user owns his or her own user context and subcontexts.

The -f option can be used to create contexts for a subset of the users found in the NIS+ table passwd.org_dir. It creates contexts for those users listed in the given input file.

Single User Context

The user type creates the user context and subcontexts for a user. A service subcontext and a binding for fs are created under the user context unless the -o option is used. When the -o option is used, only the user context is created.

For example, the command

```
# fncreate -t user org/sales/user/jjones/
```

creates the user context for the user named jjones and effectively runs the commands

```
fncreate -t service org/sales/user/jjones/service/
fncreate -t fs org/sales/user/jjones/fs/
```

The user context and its subcontexts are owned by the user for whom the contexts were created. In the above example, the contexts created are owned by the user jjones with NIS+ principal name jjones.sales.wiz.com.

The `username` context (`org/sales/user` in the above example) to which the user belongs must already exist. The user name supplied should already exist in the NIS+ `passwd.org_dir` table.

Service Context

The `service` type creates the `service` context in which service names can be bound. There is no restriction on what type of references may be bound in a service context. The policies depend on the applications that use the `service` context. For example, a group of desktop applications may bind references for a calendar, a telephone directory, a fax service, and a printer in a service context.

For example, the command

```
# fncreate -t service org/sales/service/
```

creates a service context for the organization `sales`. Because the terminal atomic name is a namespace identifier, `fncreate` also adds a binding for `org/sales/_service/` that is bound to the reference of `org/sales/service/`. After executing this command, names such as `org/sales/service/calendar` and `org/sales/service/fax` can then be bound in this service context.

The `service` context supports a hierarchical namespace, with slash-separated left-to-right names. The service namespace can be partitioned for different services. Continuing with the desktop applications example, a group of plotters may be named under the `service` context after the creation of the `plotter` context.

```
# fncreate -t service org/sales/service/plotter
```

Names such as `org/sales/service/plotter/speedy` and `org/sales/service/plotter/production` could then be bound under the `service` context.

Note – Because the terminal atomic name is not a namespace identifier, no additional binding is added (as was the case with `service` and `_service`).

The `service` context created is owned by the administrator who ran the `fncreate` command.

Printer Context

The `printer` context is created under the `service` context of the respective composite name. Refer to Chapter 9, "Administering the Printer Namespace," for more information.

Generic Context

The `generic` type creates a context for binding names used by applications.

A generic context is similar to a service context except it can have a different reference type. The `-r` option is used to specify the reference type for the generic context being created. If it is omitted, the reference type is inherited from its parent generic context or, if the parent context is not a generic context, the reference type used is a default generic reference type.

Like a service context, there is no restriction on what type of references may be bound in a generic context. The policies depend on the applications that use the generic context.

For example, the command

```
# fncreate -t generic -r WIDC_comm org/sales/service/extcomm
```

creates a generic context with the `WIDC_comm` reference type under the `service` context of the organization `sales`. Names such as `org/sales/service/extcomm/modem` can then be bound in this generic context.

The `generic` context supports a hierarchical namespace, with slash-separated left-to-right names, which allows an application to partition its namespace for different services. Continuing with the example above, a `generic` subcontext for `modem` can be created running the command

```
# fncreate -t generic org/sales/service/extcomm/modem
```

Names such as `org/sales/service/extcomm/modem/secure` and `org/sales/service/extcomm/modem/public` could then be bound under the `modem` context.

The generic context created is owned by the administrator who ran the `fncreate` command.

Site Context

The `site` type creates contexts in which site names can be bound.

For example, the command

```
# fncreate -t site org/sales/site/
```

creates a site context. Because the terminal atomic name is a namespace identifier, `fncreate` also adds a binding for `org/sales/_site/` that is bound to the reference of `org/sales/site/`.

The `site` context supports a hierarchial namespace, with dot-separated right-to-left names, which allows sites to be partitioned by their geographical coverage relationships.

For example, the commands

```
# fncreate -t site org/sales/site/east
# fncreate -t site org/sales/site/maynard.east
```

create a site context `east` and a site subcontext `maynard.east` for it.

Note – Because these terminal atomic names are not namespace identifiers, no additional binding are added (as was the case with `site` and `_site`).

The `site` context created is owned by the administrator who ran the `fncreate` command.

File Context

The fs type creates a file system context (or file context) for a user or a host. For example, the command

```
# fncreate -t fs org/sales/user/jjones/fs/
```

creates the fs context for user jjones. Because the terminal atomic name is a namespace identifier, fncreate also adds a binding for org/sales/user/jjones/_fs/ that is bound to the reference of org/sales/user/jjones/fs/.

The fs context of a user is the user's home directory as it is stored in the NIS+ passwd.org_dir table. The fs context of a host is the set of NFS file systems that the host exports (see *NFS Administration Guide*).

Use the fncreate_fs command to create file contexts for organizations and sites or to create file contexts other than the defaults for users and hosts. See Chapter 8, "Administering the File System Namespace," for details.

The fs context created is owned by the administrator who ran the fncreate command.

Namespace Identifier Context

The nsid (namespace identifier) type creates a context in which namespace identifiers can be bound.

For example, the command

```
# fncreate -t nsid org/sales/site/maynard.east/
```

creates the nsid context for the site maynard.east and permits the creation of subcontexts such as service/. Continuing with this example, you could then execute the command

```
# fncreate -t service org/sales/site/maynard.east/service/
```

to create the service context for maynard.east.

The nsid context created is owned by the administrator who ran the fncreate command.

Managing and Examining FNS Contexts

A number of tools are provided for examining and managing FNS contexts. The commands and their syntax are shown as follows. For additional information, see the man page for the tool. Note that fnbind has two usages.

```
fnlookup [-v][-L] composite_name

fnlist [-l][-v] [composite_name]

fnbind [-s][-v][-L] name new_name

fnbind -r [-s] [-v] new_name [-O | -U] ref_type {[-O | -U] | address_type [-c|-x] address_contents}+

fnunbind composite_name

fnrename [-sv] context_name old_atomic_name new_atomic_name

fndestroy composite_name
```

Displaying the Binding

fnlookup displays the binding of the given composite name.

Table 6-2 fnlookup Command Options

Option	Description
-v	Displays the binding in more detail
-L	Displays the reference to which the XFN link is bound

For example, the command

```
# fnlookup user/jjones/
Reference type: onc_fn_user
Address type: onc_fn_nisplus
 context type: user
```

shows the binding of the user `jjones`.

```
# fnlookup -v user/jjones/
Reference type: onc_fn_user
Address type: onc_fn_nisplus
 length: 52
 context type: user
 representation: normal
 version: 0
 internal name: fns_user_jjones.ctx_dir.sales.wiz.com.
```

Suppose `user/James.Jones` is linked to `user/jjones`. The first command in the following example shows what `user/James.Jones` is bound to (an XFN link). The second command follows the XFN link, `user/jjones`, and shows what `user/jjones` is bound to (the `user` context).

```
# fnlookup user/James.Jones
Reference type: fn_link_ref
Address type: fn_link_addr
 Link name: user/jjones

# fnlookup -L user/James.Jones
Reference type: onc_fn_user
Address type: onc_fn_nisplus
 context type: user
```

Listing the Context

`fnlist` lists the contents of the context identified by the given name.

Table 6-3 `fnlist` Command Options

Option	Description
`-v`	Displays the binding in more detail
`-l`	Displays the bindings of the names bound in the named context

For example, the command

```
# fnlist user/
Listing 'user/':
jjones
mladd
jsmith
James.Jones
```

shows the bindings under the `user` context.

If no name is given, the command lists the contents of the initial context.

```
# fnlist
Listing '':
_myorgunit
...
_myself
thishost
myself
_orgunit
_host
_thisens
myens
thisens
org
orgunit
thisuser
_thishost
myorgunit
_user
thisorgunit
host
_thisorgunit
_myens
user
```

When the `-l` option is given, the bindings of the names bound in the named context are displayed.

```
# fnlist -l user/
Listing bindings 'user/':
name: mladd
Reference type: onc_fn_user
Address type: onc_fn_nisplus
 context type: user
name: jsmith
Reference type: onc_fn_user
Address type: onc_fn_nisplus
 context type: user
name: James.Jones
Reference type: fn_link_ref
Address type: fn_link_addr
 Link name: user/jjones
name: jjones
Reference type: onc_fn_user
Address type: onc_fn_nisplus
 context type: user
```

When the -v option is given in conjunction with the -l option, the bindings are displayed in detail.

```
# fnlist -lv user/
Listing bindings 'user/':
name: mladd
Reference type: onc_fn_user
Address type: onc_fn_nisplus
 length: 52
 context type: user
 representation: normal
 version: 0
 internal name: fns_user_mladd.ctx_dir.sales.wiz.com.
name: jsmith
Reference type: onc_fn_user
Address type: onc_fn_nisplus
 length: 52
 context type: user
 representation: normal
 version: 0
 internal name: fns_user_jsmith.ctx_dir.sales.wiz.com.
name: James.Jones
Reference type: fn_link_ref
Address type: fn_link_addr
 length: 11
 data: 0x75 0x73 0x65 0x72 0x2f 0x6a 0x6a 0x6f 0x6e 0x65
user/jjones
name: jjones
Reference type: onc_fn_user
Address type: onc_fn_nisplus
 length: 52
 context type: user
 representation: normal
 version: 0
 internal name: fns_user_jjones.ctx_dir.sales.wiz.com.
```

Binding a Composite Name to a Reference

`fnbind` allows you to bind a composite name to a reference. There are two uses of this command. The first usage allows the user to bind the reference of an existing name to a new name. The second usage allows the user to bind a reference constructed using arguments in the command line to a name.

Table 6-4 `fnbind` Command Options

Option	Description
`-s`	Supersedes any existing binding of the original composite name
`-v`	Prints out the reference used for the binding
`-L`	Creates an XFN link using *name* and binding it to *new_name*
`-c`	Stores address contents without XDR encoding
`-x`	Interprets address contents as a hexadecimal input string and store it as is
`-r`	Creates a reference with a specified type and bind the reference to a name specified on the command line
`-O`	Interprets and stores type string as ASN.1 dot-separated integer list
`-U`	Interprets and stores type string as a DCE UUID

The first usage of `fnbind` is

```
fnbind [-s][-v][-L] name new_name
```

For example, the command

```
# fnbind myorgunit/service/printer user/mladd/service/printer
```

binds the name `user/mladd/service/printer` to the reference of `myorgunit/service/printer`.

If the given *new_name* is already bound, `fnbind -s` must be used or the operation will fail. In the above example, if `user/mladd/service/printer` is already bound, the `-s` option must be used to overwrite the existing binding with that of `myorgunit/service/printer`.

```
# fnbind -s myorgunit/service/printer user/mladd/service/printer
```

The -v option prints out the reference used for the binding.

```
# fnbind -v myorgunit/service/printer user/mladd/service/printer
Reference type: onc_printers
Address type: onc_fn_printer_nisplus
```

The following command constructs an XFN link out of `user/jjones` and binds it to the name `user/James.Jones`

```
# fnbind -L user/jjones user/James.Jones
```

Similarly, the following creates a link from `user/mladd/service/printer` to `myorgunit/service/printer`.

```
# fnbind -sL myorgunit/service/printer user/mlad/service/printer
```

The second usage of `fnbind` constructs a reference using arguments in the command line and binds it to the given name.

```
fnbind -r [-s] [-v] new_name [-O | -U] ref_type {[-O | -U] | address_type [-c|-x] address_contents}+
```

For example

```
# fnbind -r thisorgunit/service/calendar onc_calendar onc_cal_str staff@exodus
```

binds the name `thisorgunit/service/calendar` to the reference with type `onc_calendar` and address type `onc_cal_str` and address contents of `staff@exodus`.

By default, the address contents supplied in the command line is XDR-encoded before being stored in the reference. If the -c option is given, the address contents are stored in the clear, not as an XDR-encoded string. If the -x option is given, the address contents supplied in the command line are interpreted as a hexadecimal string and stored (and not XDR-encoded).

By default, the reference and address types of the reference to be constructed uses the FN_ID_STRING identifier format. If the -O option is given, the identifier format is FN_ID_ISO_OID_STRING, an ASN.1[1] dot-separated integer list string. If the -U option is given, the identifier format is FN_ID_DCE_UUID, a DCE UUID[2] in string form. For example, the following command binds to the name thisorgunit/service/nx a reference with OIDs as reference and address types and a hexadecimal string as the address contents.

```
# fnbind -r thisorgunit/service/nx -O 1.2.99.6.2.1 -O 1.2.99.6.2.3 -x ef12eab67290
```

Removing a Composite Name

fnunbind removes the given composite name from the namespace. Note that this does not remove the object associated with the name; it only unbinds the name from the object. For example, the command

```
# fnunbind user/jjones/service/printer/color
```

removes the binding associated with the name user/jjones/service/printer/color.

Renaming an Existing Binding

fnrename renames an existing binding. The following example renames the binding of clndr to calendar, in the context named by user/jjones/service/.

```
# fnrename user/jjones/service/ clndr calendar
```

The -s option is used to overwrite any binding to *new_atomic_name*.

1. See ISO 8824: 1990, Information Technology — Open Systems Interconnection — Specification of Abstract Syntax Notation One (ASN.1)

2. See X/Open Preliminary Specification, October 1993, X/Open DCE: Remote Procedure Call (ISBN: 1-872630-95-2)

Destroying the Named Context

fndestroy removes the given composite name from the namespace and destroys the context named by the composite name.

For example, the command

```
# fndestroy user/jjones/
```

unbinds the name user/jones/ from the namespace and destroys the context named by user/jjones/.

If the composite name identifies a context to be removed, the command fails if the context contains subcontexts.

Managing and Examining FNS Attributes

The fnattr command lets you update and examine attributes associated with FNS named objects. The four main options are for adding, deleting, listing, and modifying an attribute. In each of these cases, the identifier format is FN_ID_STRING, unless the option -O or -U is used.

Adding an Attribute

The -a option is for adding an attribute or adding a value to an attribute. You need to specify the composite name the attribute is associated with, the attribute identifier, and the values to add.

```
fnattr -a [-s] composite_name [-O | -U] identifier value1 [value2+]
```

The following example adds the attribute identifier model and the value hplaser to thisorgunit/service/printer.

```
# fnattr -a thisorgunit/service/printer model hplaser
```

The `-s` option means “add in supersede” mode. If an attribute with the specified identifier already exists, `-s` removes all of its values and replaces them with the values added. If this option is omitted, the resulting values for the specified attribute includes the existing values and the new values added.

```
# fnattr -as thisorgunit/service/printer model hplaser
```

The example above will first remove any existing values associated with `model` and add `hplaser` as the value.

Deleting an Attribute

To delete an attribute associated with an FNS named object, use the `-d` option. You can control what to delete:

- If an identifier is not specified, all the attributes associated with the named object are removed.
- If an identifier is specified, but without values, the entire attribute identified by *identifier* is removed.
- If individual values are specified, then only those values are removed from the attribute. Removal of the last value of an attribute is the same as removing the attribute itself.

```
fnattr -d composite_name [[-O | -U] identifier value1 [value2+]]]
```

The following command deletes all the attributes associated with `thisorgunit/service/printer`.

```
# fnattr -d thisorgunit/service/printer
```

Listing an Attribute

The `-l` option is for listing attributes and their values. The command syntax is

```
fnattr -l composite_name [[-O | -U] identifier]
```

The following example lists the values of the `model` attribute of `thisorgunit/service/printer`.

```
# fnattr -l thisorgunit/service/printer model
laser
postscript
```

If an identifier is not specified, all the attributes associated with the named object are displayed.

Modifying an Attribute

The `-m` option lets you modify an attribute value. The command syntax is

```
fnattr -m composite_name [-O | -U] identifier old_value new_value
```

For example,

```
# fnattr -m thisorgunit/service/printer model postscript laser
```

replaces the value `postscript` with `laser`. Other attributes and values associated with `thisorgunit/service/printer` are not affected.

Other Options

The `-O` option assumes the format of the attribute identifier is an ASN.1 dot-separated integer string list (`FN_ID_ISO_OID_STRING`).

The `-U` option assumes the format of the attribute identifier is a DCE UUID string form (`FN_ID_DCE_UUID`).

Maintaining Consistency Between NIS+ and FNS

A key task of the system administrator is to maintain consistency between FNS and NIS+ by ensuring that the contents of FNS contexts and NIS+ tables correspond. This correspondence is initially accomplished by the `fncreate` command, which ensures that FNS contexts are correctly created and populated and are consistent with NIS+ domain and table information. After the FNS contexts have been set up, the correspondence needs to be maintained as new users and hosts are added to and removed from the system.

The Solstice™ AdminTools™ product that adds user and host information to NIS+ also updates FNS. When updates to FNS or NIS+ are made independent of the Solstice AdminTools product, the resulting inconsistencies can be resolved by the use of the FNS tool, `fncheck`. `fncheck` checks for inconsistencies between user and host names in FNS, and user and host names in NIS+.

Checking Naming Inconsistencies

The `fncheck` command checks for naming inconsistences between the `hostname` and `username` contexts, and the NIS+ standard system tables `hosts.org_dir` and `passwd.org_dir`, respectively. `fncheck` lists host and user names that are in the FNS namespace but not in the NIS+ standard system tables. It also lists host or user names that are in the NIS+ standard system tables but not in the FNS namespace.

The command syntax is

```
fncheck [-r] [-s] [-u] [-t hostname|username] [domain_name]
```

Table 6-5 `fncheck` Command Options

Option	Description
`-t`	Specifies the type of context to check
`-s`	Lists host or user names from the NIS+ standard system tables that are not in the FNS namespace
`-r`	Lists host or user names from the FNS namespace that do not have entries in the corresponding NIS+ standard system tables
`-u`	Updates the FNS namespace based on information in the relevant NIS+ standard system tables

The `-t` option is used to specify the contexts to check. For the `-t hostname` option, the hostname context associated with the organization is checked against the NIS+ `hosts.org_dir` table of the same organization. For the `-t username` option, the username context associated with the organization is checked against the NIS+ passwd.org_dir table in the same organization. When the -t option is omitted, both the hostname and username contexts are checked.

When the -r option is used in conjunction with the -u option, items that appear only in the FNS context are removed from the FNS context. When the -s option is used in conjunction with the -u option, items that appear only in the NIS+ standard system tables are added to the FNS context. If neither -r or -s are specified, items are added and removed from the FNS context to make it consistent with the corresponding NIS+ table.

Advanced FNS and NIS+ Issues

This section provides detailed information on the relationship between NIS+ objects and FNS objects. This information is useful when you must change the access control of FNS objects.

Mapping FNS Contexts to NIS+ Objects

FNS contexts are stored as NIS+ objects. All contexts associated with an organization are stored under the ctx_dir directory of the associated NIS+ domain, which resides at the same level as the org_dir directory of the same domain.

Use the -v option for the fnlookup or fnlist command to see the detailed description of references. The internal name field displays the name of the corresponding NIS+ object.

Browsing FNS Structures Using NIS+ Commands

The NIS+ command, `nisls`, can be used to list the NIS+ objects used by FNS. For example, the following commands list the contents of the NIS+ domain directory and its `ctx_dir` subdirectory.

```
# nisls wiz.com.
wiz.com.:
eng
sales
org_dir
ctx_dir
```

```
# nisls ctx_dir.wiz.com.
ctx_dir_Wiz.COM.:
fns
fns_user
fns_host
fns_host_alto
fns_host_mladd
fns_host_elvira
fns_user_jjones
fns_user_jsmith
fns_user_aw
```

Use the `niscat` command to list the contents of the `fns_hosts` table.

```
# niscat fns_host.ctx_dir
alto *BINARY* *BINARY*
mladd *BINARY* *BINARY*
elvira *BINARY* *BINARY*
```

Checking Access Control

Use `niscat -o` to see the access control of a context.

```
# niscat -o fns_host.ctx_dir
Object Name:fns_host
Owner: alto.wiz.com.
Group: admin.wiz.com.
Domain: ctx_dir.wiz.com.
Access Rights:r-c-rmcdrmcdr-c-
Time to Live:53:0:56
Object Type:TABLE
Table Type:H
Number of Columns:3
Character Separator:
Search Path:
Columns:
[0] Name:   atomicname
    Attributes:(SEARCHABLE, TEXTUAL DATA,CASE INSENSITIVE)
    Access Rights:r-c-rmcdrmcdr-c-
[1] Name:   reference
    Attributes:(BINARY DATA)
    Access Rights:r-c-rmcdrmcdr-c-
[2] Name:   flags
    Attributes:(BINARY DATA)
    Access Rights:r-c-rmcdrmcdr-c-
```

To see the access control of a particular binding, use the name of the binding entry in the parent context's binding table (that is, the name displayed in the internal name field in the output of `fnlookup -v` and `fnlist -v`):

```
# niscat -o "[atomicname=alto],fns_host.ctx_dir"
Object Name:fns_host
Owner:  alto.wiz.com.
Group: admin.wiz.com.
Domain: ctx_dir.wiz.com.
Access Rights:r-c-rmcdrmcdr-c-
Time to Live:12:0:0
Object Type:ENTRY
     Entry data of type H
     [1] - [5 bytes] 'alto'
     [2] - [104 bytes] '0x00 ...'
     [3] - [1 bytes] 0x01
```

To change the access control or ownership of a particular context, use the commands

- `nischown`
- `nischmod`
- `nischgrp`

and give as an argument either the binding entry or the bindings table, depending on the object the operation is to affect.

Significance of Double Slashes

In the name, `org//`, the double slashes identifies the context of namespace identifiers associated with the root organizational name, as in `org//service/printer`.

In contrast, `org/` points to the root of an organizational context. Each organizational context has suborganizations as well as a pointer to the context that contains namespace identifiers such as `service`, `user`, and `host`. `org/` names the root organizational context in which you can name suborganizations, as in `org/sales.finance`.

Significance of Trailing Slash

The trailing / names objects in the next naming system. You need it whenever you are going from one naming system to another. For example, the name, `org/sales.finance` names the context for naming suborganizations of the `sales.finance` organization, as in `org/audit.sales.finance.`

On the other hand, `org/sales.finance/` names the context for naming namespace identifiers of the `sales.finance` organization, as in `org/sales.finance/service/printer`.

Error Messages

When an error occurs, FNS commands print out the remaining part of the name on which the operation failed. The part of the name that has not been printed has been processed successfully.

For example, a user attempted to create a context for `org//service/trading/bb`. The name `org//service/` was resolved successfully, but `trading` was not found in the context named by `org//service/`. Thus, `trading/bb` is displayed as the part of the name that remains when the operation failed:

```
Error in creating 'org//service/trading/bb': Name Not Found:
'trading/bb'
```

In another example, a user attempted to destroy the context `org//service/dictionary/english`, but could not carry out the operation because the context named was not empty. The pair of single quotes (`''`) indicates that FNS was able to resolve the complete name given, but could not complete the operation as requested:

```
Error in destroying 'org//service/dictionary/english': Context
Not Empty: ''
```

FNS Message Descriptions

The FNS messages and their descriptions are as follows. These messages are encoded in the `FN_status_t` object as status codes. See either Table 10-3 on page 151 or the `FN_status_t(3)` man page for the corresponding status codes.

`attribute no permission`

The caller did not have permission to perform the attempted attribute operation.

`attribute value required`

The operation attempted to create an attribute without a value, and the specific naming system does not allow this.

`authentication failure`

The operation could not be completed because the principal making the request cannot be authenticated with the name service involved. If the service is NIS+, check that you are identified as the correct principal (run the command `nisdefaults`) and that your machine has specified the correct source for public keys. Check that the `/etc/nsswitch.conf` file has the entry, `publickey: nisplus`.

`bad reference`

FNS could not interpret the contents of the reference. This may result if the contents of the reference has been corrupted or when the reference identifies itself as an FNS reference, but FNS doesn't know how to decode it.

`communication failure`

FNS could not communicate with the name service to complete the operation.

`configuration error`

An error resulted because of configuration problems. Examples: (1) the bindings table are removed out-of-band (outside of FNS), (2) and a host is in the NIS+ hosts table but does not have a corresponding FNS host context.

`context not empty`

An attempt has been made to remove a context that still contains bindings.

`continue operation using status values`

The operation should be continued using the remaining name and the resolved reference returned in the status.

`error`

An error that cannot be classified as one of the other errors listed above occurred while processing the request. Check the status of the name services involved in the operation and see whether any of them are experiencing extraordinary problems.

`illegal name`

The name supplied is not a legal name.

`incompatible code sets`

The operation involved character strings from incompatible code sets, or the supplied code set is not supported by the implementation.

`insufficient resources`

The name service used by FNS does not have sufficient resources to complete the request. Check memory and disk availability on the name servers involved.

`invalid attribute identifier`

The attribute identifier is in a format not acceptable to the naming system, or its contents are not valid for the format specified for the identifier.

`invalid attribute value`

The value supplied is not in the correct form for the given attribute.

`invalid enumeration handle`

The enumeration handle supplied is invalid. The handle could have been from another enumeration, an update operation may have occurred during the enumeration, or there may have been some other reason.

`invalid syntax attributes`

The syntax attributes supplied are invalid or insufficient to fully specify the syntax.

`link error`

An error occurred while resolving an XFN link with the supplied name.

`link loop limit reached`

A nonterminating loop was detected due to XFN links encountered during composite name resolution, or the implementation-defined limit was exceeded on the number of XFN links allowed for a single operation.

`malformed link`

A malformed link reference was found during an `fn_ctx_lookup_link()` operation. The name supplied resolved to a reference that was not a link.

`name in use`

The name supplied is already bound in the context.

`name not found`

The name supplied was not found.

`no permission`

The operation failed because of access control problems.

`no such attribute`

The object did not have an attribute with the given identifier.

`no supported address`

No shared library could be found under the `/usr/lib/fn` directory for any of the address types found in the reference bound to the FNS name. Shared libraries for an address type are named according to this convention: `fn_ctx_`*address_type*`.so`.

For example, a reference with address type `onc_fn_nisplus` would have a shared library in the path name: `/usr/lib/fn/fn_ctx_onc_fn_nisplus.so`.

`not a context`

The reference does not correspond to a valid context.

`operation not supported`

The operation is not supported by the context. For example, trying to destroy an organization is not supported.

`partial result returned`

The operation returned a partial result.

`success`

Operation succeeded.

`syntax not supported`

The syntax type is not supported.

`too many attribute values`

The operation attempted to associate more values with an attribute than the naming system supports.

`unavailable`

The name service upon which the operation depends is unavailable.

Troubleshooting

This section presents problem scenarios with a description of probable causes, diagnoses, and solutions.

Cannot Obtain Initial Context

Symptom

I get the message "Cannot obtain initial context."

Possible Cause

This is caused by an installation problem.

Diagnosis

Check that FNS has been installed properly by looking for the file, /usr/lib/fn/fn_ctx_initial.so.

Solution

Install the fn_ctx_initial.so library.

Nothing in Initial Context

Symptom

I run fnlist to look at what is in the initial context but see nothing.

Possible Cause

This is caused by an NIS+ configuration problem. The organization associated with the user and machine running the fn* commands do not have an associated ctx_dir directory.

Diagnosis

Use the nisls command to see whether there is a ctx_dir directory.

Solution

If there is no ctx_dir directory, run fncreate -t org/*nis+_domain_name*/ to create the ctx_dir directory.

"No Permission" Messages

Symptom

I get "no permission" messages.

Possible Cause

"No permission" messages mean that you do not have access to perform the command.

Diagnosis

Check permission using the appropriate NIS+ commands, described in "Advanced FNS and NIS+ Issues" on page 96. Use the `nisdefaults` command to determine your NIS+ principal name.

Another area to check is whether you are using the right name. For example, `org//` names the context of the root organization. Make sure you have permission to manipulate the root organization. Or maybe you meant to specify `myorgunit/`, instead.

Solution

If you do have permission, then the appropriate credentials probably have not been acquired.

This could be caused by the following:

- A `keylogin` has not been performed (defaults to NIS+ principal nobody).
- A `keylogin` was made to a source other than NIS+.
  - Check that the `/etc/nsswitch.conf` file has a `publickey: nisplus` entry.
  - This might manifest itself as an authentication error.

`fnlist` *Does Not List Suborganizations*

Symptom

I run `fnlist` with an organization name, expecting to see suborganizations, but instead see nothing.

Possible Cause

This is caused by an NIS+ configuration problem. Suborganizations must be NIS+ domains. By definition, an NIS+ domain must have a subdirectory named `org_dir`.

Diagnosis

Use the `nisls` command to see what subdirectories exist. Run `nisls` on each subdirectory to verify which subdirectories have an `org_dir`. The subdirectories with an `org_dir` are suborganizations.

Solution

Not applicable.

Cannot Create Host- or User-related Contexts

Symptom

When I run `fncreate -t` for the `user`, `username`, `host`, or `hostname` contexts, nothing happens.

Possible Cause

You have not set the `NIS_GROUP` environment variable. When you create a user or host context it is owned by the host or user, and not by the administrator who set up the namespace. Therefore, `fncreate` requires that the `NIS_GROUP` variable be set to enable the administrators who are part of that group to subsequently manipulate the contexts.

Diagnosis

Check the `NIS_GROUP` environment variable.

Solution

The `NIS_GROUP` environment variable should be set to the group name of the administrators who will administer the contexts.

Cannot Remove a Context I Created

Symptom

When I run `fndestroy` on the `host` or `user` context the context is not removed.

Possible Cause

You do not own the `host` or `user` context. When you create a user or host context it is owned by the host or user, and not by the administrator who set up the namespace.

Diagnosis

Check the NIS_GROUP environment variable.

Solution

The NIS_GROUP environment variable needs to be set to the group name of the administrator who will administer the contexts.

"Name in Use" With fnunbind

Symptom

I get "name in use" when trying to remove bindings. It works for certain names but not for others.

Possible Cause

You cannot unbind the name of a context. This restriction is in place to avoid leaving behind contexts that have no name ("orphaned contexts").

Diagnosis

Run the fnlist command on the name to verify that it is a context.

Solution

If the name is a context, use the fndestroy command to destroy the context.

"Name in Use" With fnbind/fncreate -s

Symptom

I use the -s option with fnbind and fncreate, but for certain names I get "name in use."

Possible Cause

fnbind -s and fncreate -s overwrite the existing binding if it already exists; but if the old binding is one that must be kept to avoid orphaned contexts, the operation fails with a "name in use" error because the binding could not be removed. This is done to avoid orphaned contexts.

Diagnosis

Run the `fnlist` command on the name to verify that it is a context.

Solution

Run the `fndestroy` command to remove the context *before* running `fnbind` or `fncreate` on the same name.

`fndestroy/fnunbind` *Does Not Return "Operation Failed"*

Symptom

When I do an `fndestroy` or `fnunbind` on certain names that I know do *not* exist, I receive no indication that the operation failed.

Possible Cause

The operation did not fail. The semantics of `fndestroy` and `fnunbind` are that if the terminal name is not bound, the operation returns success.

Diagnosis

Run the `fnlookup` command on the name. You should receive the message, "name not found."

Solution

Not applicable.