Administering the File System Namespace

8

This chapter describes the file system namespace and the procedures for creating file contexts.

The FNS File System Namespace	*page 117*
Creating File Contexts	*page 120*
Administering File Contexts	*page 126*

The FNS File System Namespace

Files may be named relative to users, hosts, organizations, and sites in FNS by appending the `fs` namespace identifier to the name of the object, and following this with the name of the file. For example, an engineering organization's tools directory might be named `org/engineering/fs/tools`.

The initial context is located under `/xfn` in the file system's root directory. Thus a user might access the `tools` directory by typing

```
% cd /xfn/org/engineering/fs/tools
```

Existing applications can access this directory just as they would any other directory. Applications do not need to be modified in any way or use the XFN API.

8

NFS File Servers

NFS is Sun's distributed file system. The files associated with an object will generally reside on one or more remote NFS file servers. In the simplest case, the namespace identifier `fs` corresponds to the root of an exported NFS file system, as shown in Figure 8-1.

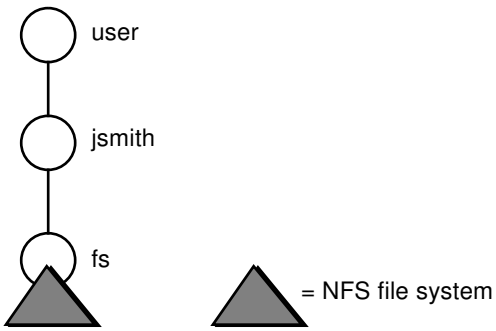

Figure 8-1 NFS File System—Simple Case

In contrast, an object's file system may be composed of multiple—and possibly overlapping—remote mounts, woven together into a "virtual" directory structure managed by FNS.

Figure 8-2 illustrates how this capability might be used to piece together an organization's file system from three separate file servers. The `project` directory, along with its `lib` subdirectory, resides on one file server, while the `src` subdirectory resides on another. Users and applications need not be aware of the use of multiple servers; they see a single, seamless namespace.

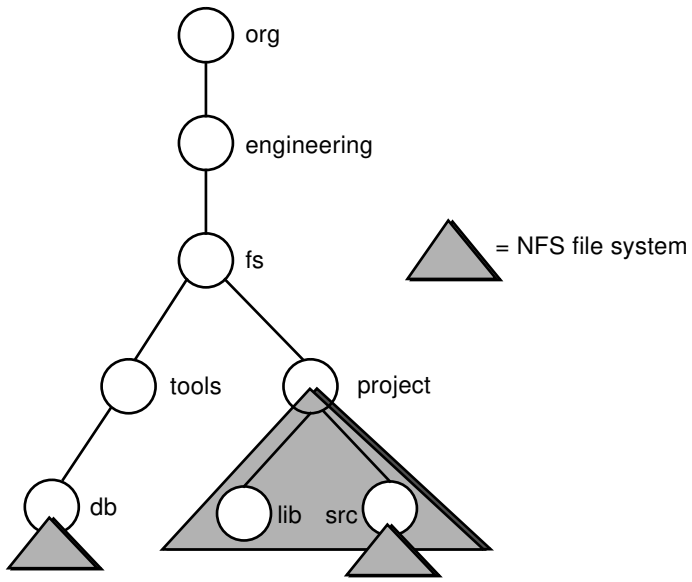

Figure 8-2 NFS File System—Multiple Servers

The Automounter

For efficiency, the automounter (see *NFS Administration Guide*) is used to mount FNS directories on demand. The default `/etc/auto_master` configuration file contains the line:

```
/xfn      -xfn
```

which tells the automounter that the FNS namespace is “mounted” under `/xfn`, as specified by XFN.

Since the automounter is used to mount directories named through FNS, the subdirectories of an FNS directory cannot be listed until they have been mounted. For example, suppose the file system of the sales organization is composed of multiple NFS file systems. The following `ls` command shows only two file systems that have been visited recently and are currently mounted:

```
% ls /xfn/org/sales/fs
  customers    products
```

To see the entire listing, use the `fnlist(1)` command.:

```
% fnlist org/sales/fs
  Listing 'org/sales/fs':
  products
  goals
  customers
  incentives
```

Creating File Contexts

The `fncreate_fs(1M)` command creates file contexts for organizations and sites. It may also be used to override the default file contexts for users and hosts that are created by the `fncreate(1M)` command. See "File Context" on page 82.

There are two methods of using the `fncreate_fs` command. The context bindings may be provided by an input file (See "Creating the Input File" on page 121) or on the command line (See "Using Command-line Input" on page 123).

The two methods of `fncreate_fs` have the following syntax:

```
fncreate_fs [-v] [-r] -f input_file composite_name
fncreate_fs [-v] [-r] composite_name [mount_options][mount_location...]
```

The `fncreate_fs` options `-v` and `-r` are described in Table 8-1.

Table 8-1 `fncreate_fs` Command Options

Option	Description
-v	Sets verbose output, displaying information about the contexts being created and modified.
-r	Replaces the bindings in the context named by *composite_name*—and all of its subcontexts—with *only* those specified in the input. This is equivalent to destroying the context (and, recursively, its subcontexts), and then running `fncreate_fs` without this option. The `-r` option should be used with care.

The `fncreate_fs` command manipulates FNS contexts and bindings of the `onc_fn_fs` reference type. It uses an address of type `onc_fn_fs_mount` to represent each remote mount point. The data associated with an address of this type are the corresponding mount options and locations in a single, XDR-encoded string.

Creating the Input File

The input file supplies the names and values to be bound in the context of *composite_name*. Its format is based upon and similar, but not identical, to the format of indirect automount maps (see *NFS Administration Guide*). The input file contains an entry with the form:

```
name [mount_options] [mount_location...]
```

For each entry a reference to the mount locations and the corresponding mount options is bound to the name *composite_name*/*name*.

The *name* field may be a simple atomic name or a slash-separated hierarchical name. It may also be "`.`" (dot), in which case the reference is bound directly to *composite_name*.

The *mount_location* field specifies the host or hosts that serve the files for *composite_name/name*. In a simple NFS mount, *mount_location* takes the form:

```
host:path
```

where *host* is the name of the server from which to mount the file system and *path* is the path name of the directory to mount.

The *mount_options* field begins with a hyphen ("–"). This is followed by a comma-separated list (with no spaces) of the mount options to use when mounting the directory. These options also apply to any subcontexts of *composite_name/name* that do not specify mount options of their own.

If *mount_options* and *mount_location* are both omitted, then no reference is bound to *composite_name/name*. Any existing reference is unbound.

Using the example from Figure 8-1 on page 118, suppose you want `jsmith`'s file system to be an NFS mount of the directory `/export/home/jsmith` from host `svr1`. The command would be run as follows:

```
% fncreate_fs -f infile user/jsmith/fs
```

with `infile` containing

```
.     svr1:/export/home/jsmith
```

To set up the file system illustrated in Figure 8-2 on page 119, run the command

```
% fncreate_fs -f infile org/engineering/fs
```

with `infile` containing

```
tools/db      svr1:/export/db
project       svr1:/export/proj
project/src   svr2:/export/src
```

To change the NFS mounts for `project` and its subcontext `src` to be read-only, you can change `infile` as follows:

```
tools/db        svr1:/export/db
project -ro     svr1:/export/proj
project/src     svr2:/export/src
```

The `-ro` is unnecessary in the third line. Since `src` is a subcontext of `project`, it will inherit the `-ro` mount option from above.

The following input file would make all of the mounts read-only except for `org/engineering/fs/project/src`.

```
.           -ro
tools/db            svr1:/export/db
project             svr1:/export/proj
project/src -rw     svr2:/export/src
```

Using Command-line Input

The `fncreate_fs(1M)` command also allows the binding description to be provided on the command line:

```
fncreate_fs composite_name [mount_options] [mount_location ...]
```

This is equivalent to using the first form of the command and providing a one-line input file containing "`.`" in the *name* field, and the given mount options and locations. The previous example in which `jsmith`'s file system was set could be set from the command line as follows:

```
% fncreate_fs user/jsmith/fs svr1:/export/home/jsmith
```

Similarly, the hierarchy in Figure 8-2 on page 119 could have been set up by running the sequence of commands:

```
% fncreate_fs org/engineering/fs/tools/db svr1:/export/db
% fncreate_fs org/engineering/fs/project svr1:/export/proj
% fncreate_fs org/engineering/fs/project/src svr2:/export/src
```

To make all three of the mounts read-only, you would run this command:

```
% fncreate_fs org/engineering/fs -ro
```

Advanced Input Formats

The following two sections apply to both input file and command-line input formats.

Multiple Mount Locations

Multiple *mount_location* fields may be specified for NFS file systems that are exported from multiple, functionally equivalent locations:

```
% fncreate_fs org/sales/fs svr1:/sales svr2:/sales
```

The automounter will attempt to choose the best server from among the alternatives provided. If several locations in the list share the same path name, they may be combined using a comma-separated list of host names:

```
% fncreate_fs org/sales/fs svr1,svr2:/sales
```

The hosts may be weighted, with the weighting factor appended to the host name as a nonnegative integer in parentheses: the lower the number, the more desirable the server. The default weighting factor is zero (most desirable).

The following example illustrates one way to indicate that `svr2` is the preferred server:

```
% fncreate_fs org/sales/fs svr1(2),svr2(1):/sales
```

See *NFS Administration Guide* for additional information on how the automounter interprets the *mount_location* field.

Variable Substitution

Variable names, prefixed by `$`, may be used in the *mount_options* or *mount_location* fields of `fncreate_fs`. For example, a mount location may be given as

```
svr1:/export/$CPU
```

The automounter will substitute client-specific values for these variables when mounting the corresponding file systems. In the above example, `$CPU` is replaced by the output of `uname -p`; for example, `sparc`.

See *NFS Administration Guide* for additional information on how the automounter treats variables substitution.

Backward Compatibility Input Format

For additional compatibility with automount maps, the following input file format is also accepted by `fncreate_fs`:

```
name                    [mount_options] [mount_location ...] \
        /offset1        [mount_options1] mount_location1 ... \
        /offset2        [mount_options2] mount_location2 ... \
        ...
```

where each *offset* field is a slash-separated hierarchy. The backslash (\) indicates the continuation of a single long line. This is interpreted as being equivalent to

```
name                    [mount_options]  [mount_location ...]
name/offset1            [mount_options1]  mount_location1 ...
name/offset2            [mount_options2]  mount_location2 ...
...
```

The first line is omitted if both *mount_options* and *mount_location* are omitted. This format is for compatibility only. It provides no additional functionality, and its use is discouraged.

Administering File Contexts

File contexts may be inspected using the `fnlist(1)` and `fnlookup(1)` commands, and may be pruned or destroyed using `fnunbind(1)` and `fndestroy(1M)`. These commands and sample output are described in Chapter 6, "Administering FNS on NIS+." Refer also to the man page for each command.