NEXTSTEP In Focus, Summer 1993 (Volume 3, Issue 3).

The Tough Stuff

Alan M. Marcum and Marc Majka

Some things in NetInfo aren't obvious, even with the information we've presented so far. Other things have a subtlety that is critically important. This article addresses some of those issues: making changes that aren't as simple as you'd think, fixing mistakes in **trusted_networks**, starting NetInfo by hand, figuring out where client processes are bound, understanding console messages, backing up the NetInfo database, and recovering from disasters.

MAKING ªSIMPLEº CHANGES THAT AREN'T SO SIMPLE

A few changes to a NetInfo domain that seem simple and straightforward really aren't. Fouling up some of these changes can render a database or an entire domain unusable. Here's how to make these changes correctly.

Changing the master server for a domain

To change a master server of a domain you must make sure that all clones in the domain reference the new master. The name of the master server is stored in the domain's root directory, as the value of the **master** property. The format of this value is *machine*/*tag*, where *machine* is the name of the computer running the master server for the domain, and *tag* is that server's database tag.

For example, at Rhino Aviation, to change the root domain master from **super21/Rhino** to **sabre/Rhino**, you'd need to follow these steps:

1. Ensure that a clone of the domain is running on **sabre**. At Rhino, there is an appropriate clone. If there weren't, you'd create a clone with, for example, NetInfoManager.
2. Make certain that all the server processes for the domain are up and running.
3. If you created a new clone in Step 1, ensure that the new clone has all the data from the domain. In Release 3, a newly built clone database usually isn't completely filled until about 30 minutes after it's initially built, or until the computer running the clone is rebooted.
4. Change the value of the **master** property from **super21/Rhino** to **sabre/Rhino**.
5. Wait for the change of the **master** property's value to propagate to all the clones. Typically, this takes only a few minutes. To verify that the change has been propagated, examine the root directory of each database in the domain, using tagged domain specifications to reference the specific database.
6. Reboot all computers in the domain.

This procedure ensures that all the clones of the domain have the correct notion of the master. If you don't wait for the change to propagate, clones with the old information never receive new updates.

Changing any computer's name

By convention, each local domain in the domain hierarchy has the same name as the server computer for the domain. This is because of the value of the **serves** property in the parent of the computer's local domain. If

you want to change a computer's name, it's helpful to make the computer name and the domain name match.

For example, assume you want to change the name of the computer **chaparral** to **chap**. **chaparral**'s local domain is a child of **/info**. (See Figure 2 in ªA Typical NetInfo Setup.º) Before the change, **/info** has a **/machines/chaparral** directory, which in turn has a **serves** property containing **chaparral/local**. If you change just the value of that directory's **name**, the computer and domain names won't matchÐthe computer will be **cha**p and the domain name will be **/info/chaparral**. While this isn't technically an error, it can be confusing.

So, when you change a computer name, also change the **serves** property in the parent domain. Also change the name in all domains it appears in and places it's used, like **/mounts** and **/locations/ntp**.

Changing a server computer's Internet address

Unless you have a two-level domain hierarchy, the computers providing NetInfo services are known both in the domain whose services are provided and in that domain's parent domain. So, if you change the Internet address of such a server computer, follow the guidelines for changing a computer's name, and be sure to change the Internet address in *all* the domains referencing the computer.

Changing the Internet address of the master server's computer

Changing the Internet address of the master server's computer requires a procedure nearly identical to that for changing the domain's master. You must make the change, wait for the change to propagate to all the clones of the domain, then reboot all the computers in the domain. Follow the guidelines for changing a computer's name and Internet address.

FIXING TRUSTED_NETWORKS MISTAKES

The **trusted_networks** property in the root directory of a domain restricts access to the information in a domain based on the requesting computer's Internet address (see the *NEXTSTEP Network and System Administration* manual). If you mistype the value of the **trusted_networks** property, it's possible that almost no computer on the network will be able to access the affected domain.

However, the **root** user on the computer running the master server for a domain can always access and modify the domain. So if you mistype the value for a **trusted_networks** property and can't access the domain, you can fix the problem as **root** on the computer running the master server for the domain.

You might need to reference the domain by tag, to ensure that you're communicating with the master server. The best way to do this is to use the loopback Internet address, 127.0.0.1. From the command line, this looks something like:

```
# niutil -read -t 127.0.0.1/network /
```

If you're using NetInfoManager, choose the Open by Tag command in the Domain menu, specifying **127.0.0.1** as the host and **network** as the tag. Using the Internet address instead of the name of the computer ensures that your request won't fail when NetInfo Manager tries to find the name.

STARTING NETINFO BY HAND

Sometimes when you encounter problems during system startup, you can fix them by booting in single-user mode and starting NetInfo by hand. The simplest way to do this is to run this command:

```
sh /etc/rc
```

Sometimes this hangs, though, and you must execute each step yourself. Here's how.

Starting, step-by-step

The sequence for starting NetInfo is different in NEXTSTEP Release 3.0 from Release 3.1. Here are the commands for starting NetInfo (including **lookupd**) under 3.0:

1. **mount -vat 4.3**
2. **mach_swapon -av**
3. **syslogd**
4. **nmserver &**
5. **portmap**
6. **nibindd**
7. **lookupd**

Note the ampersand (&) in Step 4. It causes the command to run in the background.

Here's the sequence for Release 3.1. Include Step 2 only on Intel-based computers.

1. **nmserver -nonet**
2. **driverLoader a**
3. **sh /etc/rc.net -h**
4. **mount -vat 4.3**

5. **mach_swapon -av**
6. **syslogd**
7. **portmap**
8. **nibindd**
9. **lookupd**

Invoking routing

One thing is missing from both of these sequences: enabling routing. If you need to enable routing to accomplish what you need to doÐfor example, if a needed server runs on a computer on another networkÐinvoke it just before running **nibindd** in either sequence.

How you invoke routing depends on how you configured it using HostManager. If you chose dynamic routing, run the command **routed -q**. If you chose a specific route, also known as a *static route*, use the following command:

```
route add default address 1
```

Replace *address* with the Internet address you specified in HostManager.

Modifying the command sequence

Sometimes you have to modify the startup procedure, so that you stop short of starting **lookupd**. You should do this if certain problems appear, such as if **lookupd** seems to hang. Other times you'll want to run **nibindd** in the background, so that you can retain control of the system if you have problems running **nibindd**. For more information on handling problems like these, see the articles on NetInfo startup and **lookupd** in this issue, the troubleshooting information in *NEXTSTEP Network and System Administration*, Majka 1992, and Cottle, ªThe Crash of the Master

NetInfo Server,º 1993.

Run the last step in either sequence, **lookupd**, only if you'll be using **lookupd**'s services. Frequently, you start NetInfo by hand to fix a NetInfo problem, and in these cases running **lookupd** can cause the system to hang.

DETERMINING WHAT'S BOUND WHERE

An interesting question that's asked frequently regarding NetInfo is, ªTo which server is a particular computer bound?º Unfortunately, this question can't really be answered, since *computers* aren't bound to NetInfo servers. Rather, *processes* communicate with NetInfo servers, and different processes on the same computer can communicate with different NetInfo server processes for a given domain.

When a **netinfod** process binds, it remembers the address and tag of its parent, but doesn't keep a connection with its parent. Only when a NetInfo client process asks the **netinfod** for its parent's address and tag does the **netinfod** probe to check if its parent is running. It rebinds the same way it did when it started up, if the probe fails.

You could write a program to ask a **netinfod** for its parent, but just asking the question can cause the daemon to rebind. Once binding is complete there are no network connections left, so it can be difficult to determine the current bindings.

When a process first connects to a NetInfo server, it starts with the local domain and climbs the hierarchy by asking for the address and tag of parent servers on the way up. If it needs to contact a server for a child domain at some point, it collects a list of all the servers for the child from the child's parent and connects to the first one that it finds running. There's no telling which it might contact. But remember that this contact

is established using TCP, which is a connection-oriented protocol. So, the connection remains in place until the processes involved shut it down, either voluntarily or forcibly.

By the way: If a client process makes a change to a domain, it automatically reconnects to the master.

Servers sometimes crash

Another reason it's difficult to know what bindings are in place is that bindings are dynamic. Even if you're the best system and network administrator, sometimes a NetInfo server process that's being used by a client process crashes, or appears to have crashed. Perhaps the server process is running on a computer on a distant subnet and a backhoe operator chops the telecommunications cable in two. Perhaps the server's computer fails. Perhaps a router's power supply blows up. You get the idea.

When this happens, the client process won't get an answer to its next request. Eventually, the client will try to find a new server.

When the client first connected to the server, it created a list of all the servers of the domain by looking through the domain's **/machines** directory to find hosts with a **serves** property of **./***tag*. The client saves this list for when its request to its current server times out.The client uses the list to connect to a serverÐthe same one as before or a new one, depending on the reasons behind the timeout and the results of the rebinding.

The NetInfo library

You might think all this contingency planning makes writing NetInfo clients horribly complicated. And, you'd be rightÐif it weren't for the NetInfo library. The library takes care of establishing a client's connection, maintaining that connection, and reconnecting after a timeout.

(Editor's note: An article in a future issue of NEXTSTEP in Focus will describe the NetInfo library in detail.)

Reconnecting to a server

If a client needs to reconnect to a server, it sends a message using the manycast technique described in ªNetInfo Binding and Connecting.º The reconnecting client sends a SunRPC request to each server it discovered in the initial binding process. The client connects to the first server that responds.

Where's the nibindd?

Recall that the first step in the binding process is to contact the **nibindd**s running on all the computers with potential parent servers. The client makes contact by sending a UDP broadcast of a SunRPC request to the portmapper. The portmapper forwards the request to the **nibindd**. Subsequently, the child domain's **netinfod** requests the port number for the **nibindd** running on the chosen parent computer.

You can examine the information the portmapper has regarding the SunRPC programs running on a computer, their program numbers and names, and the ports associated with each program. You use the **rpcinfo** command to do this. Here's an example:

```
sabre [~]-40% rpcinfo -p
   program vers proto   port
    100000    2   tcp    111  portmapper
    100000    2   udp    111  portmapper
 200100001    1   udp    755  netinfobind
 200100001    1   tcp    757  netinfobind
    100011    1   udp   2606  rquotad
    100001    1   udp   2607  rstat_svc
    100001    2   udp   2607  rstat_svc
    100001    3   udp   2607  rstat_svc
    100008    1   udp   2610  walld
```

```
 200100002    1   udp   2611  renderd
```

(For usage information, see the **rpcinfo**(8) UNIX manual page.)

In the example above, version 1 of the **netinfobind** SunRPC program, program number 200100001, is registered with the portmapper and is using TCP port 757 and UDP port 755.

Where's the netinfod?

nidomain -l *host* reports the NetInfo servers running on the specified computer (see ªNetInfo Binding and Connectingº). Furthermore, it reports the TCP and UDP port numbers used by each **netinfod** process. For example:

```
sabre-41% nidomain -l mite
tag=local udp=660 tcp=662
tag=network udp=664 tcp=666
```

Who's talking to whom?

Some clients, such as NetInfoManager, display the address and tag of the current server. In general, though, there's no way to ask a client about its connections. We need one more command to determine which connections are in use: **netstat**. This command shows the active sockets for each protocol in use. Figure 1 shows an excerpt from its output.

```
ranger-103% netstat
Active Internet connections
Proto Recv-Q Send-Q      Local Address  Foreign Address   (state)
tcp        0      0      ranger.3126    Tute.EDU.ftp      ESTABLISHED
tcp        0      0      ranger.1023    sabre.login       ESTABLISHED
tcp        0     96      ranger.758     ranger.780        ESTABLISHED
tcp        0      0      ranger.780     ranger.758        ESTABLISHED
tcp        0      0      ranger.758     ranger.2720       ESTABLISHED
tcp        0      0      ranger.2720    ranger.758        ESTABLISHED
tcp        0      0      nescorna.3204  mustang.2453      ESTABLISHED
```

```
tcp        0      0      ranger.758     ranger.2671      ESTABLISHED
tcp        0      0      ranger.2671    ranger.758       ESTABLISHED
tcp        0      0      ranger.775     cadet.696        ESTABLISHED
tcp        0      0      ranger.3156    cadet.696        ESTABLISHED
tcp        0      0      ranger.768     exec.679         ESTABLISHED
tcp        0      0      ranger.763     cadet.696        ESTABLISHED
udp        0      0      localhost.ntp  *.*
udp        0      0      ranger.ntp     *.*
```

Figure 1: *Sample netstat output*

In the **netstat** output, three columns are most important: ªProto,º ªLocal Address,º and ªForeign Address.º ªProtoº is the protocol used for the connection. ªLocal Addressº is the address of the local side of the connection; the format is *hostname.port*. ªForeign Addressº is the address of the remote side of the connection, using the same format as ªLocal Address.º If a port is referenced by number instead of name, it's because no translation from port number to service name was available; generally this means the port isn't well-known.

Which port for which service?

In the example in Figure 1 there are multiple connections to **ranger**'s port 758 and **cadet**'s port 696. That means multiple processes are communicating with some server on those computers over that remote port.

With ports that aren't well-known, it's pretty much impossible to determine which service is using the port. In some instances, portmapper has the information and you can use **rpcinfo** to obtain it. In other instances, the port might be used by a **netinfod**. The question is, how do you know?

The output from **nidomain -l** tells you which databases are being served on a computer, and the port numbers for contacting each server. For

example, here's the output for **ranger**, **cadet**, and **exec**:

```
ranger [~]-104% nidomain -l
tag=local udp=756 tcp=758
ranger [~]-105% nidomain -l cadet
tag=network udp=694 tcp=696
tag=local udp=693 tcp=695
ranger [~]-106% nidomain -l exec
tag=network udp=676 tcp=678
tag=local udp=675 tcp=677
tag=Rhino udp=677 tcp=679
```

In the fourth line after the headings in Figure 1, LocalAddress is ranger.780 and Foreign Address is ranger.758. The output from the first **nidomain -l** shows that **netinfod local** on **ranger** can be reached over TCP port 758. These two pieces of informationÐ**netstat**'s indication of a connection to **ranger's** TCP port 758 and **nidomain**'s output showing that **netinfod local** is using TCP port 758 on **ranger**Ðshow that some process on **ranger** has a connection to the local NetInfo server. However, in general there's no way to know which process it is.

Tools for examining connections

The floppy disk that accompanies printed copies of *NEXTSTEP In Focus* contains a program called **ni_connections**. It's a shell script that displays the connection information shown above. See the documentation files on the disk for more information on how to use it.

UNDERSTANDING NETINFO FAILURE AND SLEEPING MESSAGES

When a NetInfo server that some client is using goes down, and the client then attempts to get data from the server, you might see any of a number of messages on the console or in the system log. The table in

Figure 2 shows what they mean.

Message	Meaning
netinfo timeout, sleeping	The first possibility is that the client attempted to build its first connection and **netinfod local** didn't respond. The second is that the client was already connected to the server when it failed. When the client requested data from the server, it waited for two 5-second timeout periods and didn't get an answer. Either way, once the local NetInfo server responds, you'll see the "netinfo waking" message.
netinfo failure, sleeping	The client of a failed server timed out and has been sleeping for some time. The client is attempting to connect to a new server.
netinfo failure, aborting	A server failure occurred during writing. As a result, the client aborted the operation, rather than sleeping and retrying.
netinfo sleeping	The client attempted to reconnect to a server and failed.
multi_call timeout, sleeping	The client attempted to reconnect to a server, and none of the messages to the potential servers received a response. (The terminology here is inconsistentÐthe request was sent by manycasting, but the message says "multi_call.") Once the client receives a response, it displays the "netinfo waking" message.
netinfo waking	The client established a connection after a failure.

Figure 2: *Common NetInfo sleeping and timeout messages*

BACKING UP THE NETINFO DATABASE

A NetInfo database can be made up of more than one file. For example, in the **network.nidb** directory is a file called **Collection** (or possibly **collection** if you're using a release prior to 3.0), and zero or more ªextensionº files called **extension_***n*, where *n* is an integer. The NetInfo database is made up of *all* of these files.

This is very important to remember when you make backups of a NetInfo database. If you use an incremental backup technique, the backup created might be incompleteÐit might not contain all the extension files, for example. This complicates restoring the database from the backup. A common mistake is to restore an **.nidb** from incremental dump tapes. **dump** doesn't know how to delete old files, so you can end up with extra **extension_***n* files. These extra files confuse **netinfod**.

If you use incremental backups, we recommend backing up the NetInfo databases separately using, for example, **tar** or **cpio**.

Besides using **tar** or **cpio** to copy a **.nidb** directory, you *can* use a new feature of **nidump**, first available in Release 3.0: raw **nidump**. This feature allows you to obtain the contents of a NetInfo directory hierarchy in ASCII text format. Here's an example:

```
sabre-26% nidump -r /localconfig .
name = localconfig;
CHILDREN = ({
    name = NetWare;
    enable = YES;
}, {
    name = keyboard;
    keymap = /NextLibrary/Keyboards/USA;
}, {
    name = language;
    language = English;
}, {
    name = screens;
    _writers = (smarco, hkabir);
    CHILDREN = ({
        name = NeXTdimension;
        active = 1;
        bounds = "0 1120 0 832";
        slot = 2;
        unit = 0;
    }, {
```

```
        name = MegaPixel;
        active = 1;
        bounds = "1120 2240 0 832";
        slot = 0;
        unit = 0;
    });
});
```

This output shows four directories in **/localconfig**: **NetWare**, **keyboard**, **language**, and **screens**. The **/localconfig/screens** directory, for example, has two properties, **name** and **_writer**s. The **_writers** property in turn has two values, **smarco** and **hkabir**.

Because **nidump -r** provides information for a directory and all contained directories, you could conceivably use it as a backup tool. Be very careful, though, when reloading using **niload -r**. Experiment on a small directory in a test domain first so you understand fully what will happen. (Explaining exactly what happens under all circumstances is beyond the scope of this article.)

RECOVERING FROM DISASTERS

There are three types of disasters to discuss:

- A NetInfo database is destroyed.
- The only clone of a domain on a given subnet becomes unavailable for a long time, perhaps because its host computer's power supply fails.
- The master NetInfo server's computer fails catastrophically.

In the first case, a destroyed database, the easiest thing to do is to restore the database from your backups. Be sure to do this in single-user mode
with NetInfo shut down.

If restoring in single-user mode is impossible, remove any vestiges of the **.nidb** directory you're going to restore, start up the computer, restore the database, and reboot the computer. This procedure ensures there's no **netinfod** trying to work with a corrupted or partial database.

In the second scenario, if a clone fails, the easiest thing to do is to build a new clone. See the *NEXTSTEP Networking and System Administration* manual for details.

For the third case, failure of the master server computer, see Cottle, ªThe Crash of the Master NetInfo Server,º 1993.

Note: The document references in this and other articles in this issue refer to the books and articles listed in ªNEXTSTEP Networking References.º