nsrd.info micromanual

NetWorker Power User's Guide to nsradmin

Prepared by: Preston de Guise
preston@nsrd.info

Date: January 2010
Version: 1.0

1 Introduction

1.1 What is a micromanual?

To understand what a *micromanual* is, we first need to revisit what a standard IT or computer book looks like. Typically it will run into the size of several hundred pages, most of which the average power user will rarely use.

On the other hand, a micromanual is instead a short, concise guide aimed at providing a comprehensive overview of, and instructions for a specific topic in as small a space as possible. The three principles of a micromanual are:

- Your time as the reader is precious
- You don't want to read stuff targeted at beginners
- You'd prefer to spend less money and get just what you need

1.2 What is this micromanual?

This micromanual is *NetWorker Power User's Guide to nsradmin*, and will document the following topics:

1. An overview of the functionality of *nsradmin*.
2. Operational basics:
    a. Offline versus online configuration database access.
    b. Syntax.
    c. Starting and stopping backups.
    d. Append vs Update.
    e. Setting up regular backup components – clients, policies, groups, schedules and pools.
    f. Checking device states.
3. Advanced *nsradmin*:
    a. Mass updates.
    b. Scripting.
    c. Regular Expressions.
    d. Offline mode.

1.3 Expected Audience

It is expected that the reader of this manual:

- Has a strong familiarity with standard NetWorker configuration components such as groups, clients, policies, schedules, pools, etc., and is able to configure these components within NetWorker Management Console. (**No** step-by-step instructions are given for creating resource components in NMC – if these are required, this micromanual is not targeted at the reader.)
- Has at least a passing level of experience with either operating system scripting languages or cross-platform scripting languages such as Perl.
- Has a spare host (physical or virtual) that NetWorker can be installed on in evaluation mode for practice sessions. This can be any operating system or class of machine capable of running NetWorker (a server class system is not required).

It is assumed that while the readers will have good NetWorker experience, they will have little exposure to *nsradmin*.

Table of Contents

1 Introduction 2
1.1 What is a micromanual? 2
1.2 What is this micromanual? 2
1.3 Expected Audience 2

2 Warning 7

3 Document Conventions 8

4 Getting Started 9
4.1 Resource Database 9
4.2 Offline versus Online 10
4.3 Care and Respect 10

5 How to do the examples 11

6 Operational Basics 12
6.1 About NetWorker Commands 12
6.2 Running nsradmin 12
6.3 Syntax Overview 13
6.4 Starting and Stopping Backups 18
6.4.1 What you'll need 18
6.4.2 Monitoring 18
6.4.3 Starting a Backup 18
6.4.4 Stopping a Running Backup 20
6.4.5 Checking the status of a group 22
6.4.6 Cloning and Monitoring 23
6.5 Append vs Update 24
6.6 Setting up Regular Backup Components 26
6.6.1 Browse and Retention Policies 26
6.6.2 Schedules 28
6.6.3 Groups 31
6.6.4 Clients 33
6.6.5 Pools 35
6.6.6 Revisiting our Groups 37
6.7 Monitoring Devices 38
6.8 Deleting Resources 40
6.9 Closing Comments 43

7 Advanced nsradmin 44
7.1 Introduction 44
7.2 Bulk Activities 44
7.3 Scripting with nsradmin 47
7.3.1 Intended Goal 47
7.3.2 Introductory scripting 47
7.3.3 Preliminary Setup 48
7.3.4 A Client Creation Script 50
7.4 Connecting to the Client Services 51
7.5 Using regular expressions in nsradmin 54
7.6 Offline Mode 55

8 Appendix A – Test Setup Configuration 57
8.1 From NetWorker Management Console 57

8.2 From Unix/Linux Command Line 57
8.2.1 Resource Configuration Setup 57
8.2.2 Volume Setup 58
8.3 From Windows Command Line 58
8.3.1 Resource Configuration Setup 58
8.3.2 Volume Setup 59

Table of Figures

Figure 1: NMC view of a NetWorker policy resource ... 9

Figure 2: Using the append command with Windows paths ... 26

Figure 3: Daily schedule created in nsradmin, as viewed from NMC ... 30

Figure 4: Monthly schedule created in nsradmin, as viewed from NMC ... 31

Figure 5: Viewing a device resource in Windows ... 39

Figure 6: Using the create-policy.bat script on Windows ... 47

Figure 7: Output from the "create-client.bat" script on Windows ... 51

Figure 8: Running nsradmin in offline mode ... 56

Figure 9: Bootstrapping the NetWorker configuration required for the micromanual on Windows ... 59

Figure 10: Labelling media in the ADV_FILE devices on Windows ... 60

2 Warning

This micromanual describes steps that, if misused, could cause corruption to a NetWorker configuration database. As such, they should only be run on a freshly installed NetWorker lab server, rather than an active production server.

As is the case with all production systems, power-user commands have the capability to both significantly help successful operations, or to significantly hinder successful operations if used incorrectly. Before actively using any of the techniques described in this micromanual you should be completely familiar with their usage from self-training in a lab environment. Furthermore, you should always have an up to date bootstrap backup to recover should anything go wrong.

The author takes no responsibility for any damage to a system, or loss of functionality caused by running either the commands within this micromanual, or commands adapted from this micromanual against a NetWorker environment.

3 Document Conventions

Throughout the document, the following conventions will be used for formatting:

```
Boxed text in a standard weight text represents output of commands.
```

```
<Boxed, italicised text in angle brackets represent an in-session comment, not output
expected to be seen during the session.>
```

```
Boxed text in a bold weight text represents commands to be typed in.
```

```
Boxed text that is bold and italicised is part of a command to be typed in, but
you should substitute with local text (e.g., replacing a hostname).
```

```
Text in a dotted box represents scripts that should be saved to file, then executed at a
later step.
```

4 Getting Started

4.1 Resource Database

If you've worked with NetWorker from within the management console (or previously, from one of the OS-specific GUIs), you're more than likely aware of how configuration components such as clients, schedules, groups, policies, etc., look within the GUI. For example, here's what a policy looks like:

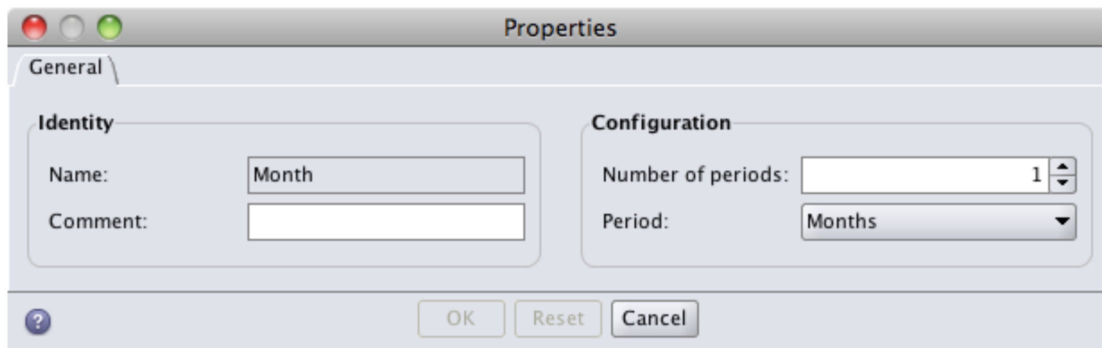

Figure 1: NMC view of a NetWorker policy resource

The configuration details for such a policy are maintained as part of a plain text file on the NetWorker server, stored within the 'res' directory. On a Unix/Linux system, this is typically in /nsr/res, and on a Windows system the *default* install location is "C:\Program Files\Legato\nsr\res".

Previously NetWorker only ever had 3 configuration files within the res directory:

- nsr.res – Most configuration options
- nsrjb.res – Jukebox, device and label template configuration options
- nsrla.res – Security/Port configuration options

Unfortunately, with just 3 files yet many resources within each file, corruption was not uncommon, and so in NetWorker 7.0, a new and much improved resource *database* structure was introduced. This saw the content of nsr.res and nsrjb.res split up and configured as individual files, located under a new directory, ('nsrdb') within the 'res' directory, and organised in a hashed structure. (Over time, *nsrla.res* was similarly split up, but organised into a hashed directory structure underneath 'nsrladb' in the same parent directory, 'res'.)

For instance, looking at a Unix NetWorker server, you may find directories and files such as the following:

```
# ls /nsr/res/nsrdb
01/ 02/ 03/ 04/ 05/ 06/ 07/ 08/ 09/

# ls /nsr/res/nsrdb/01
0b005d7500000000d2255549000000000a000001
15005d7500000000d2255549000000000a000001
1f00151400000000972e5d4a000000000a000001
1f00261700000000f4efcb4a000000000a000001
1f005d7500000000d2255549000000000a000001
```

Each one of those lengthy named files is a single NetWorker configuration resource – a policy, or a client, or a schedule, etc.

For example, the NetWorker resource file for the 'Month' policy on a server in my lab looks like:

```
comment:;
name: Month;
number of periods: 1;
period: Months;
type: NSR policy;
resource identifier: 62.0.93.117.0.0.0.0.210.37.85.73.0.0.0.0.10.0.0.1(1)
```

Now, the first thing to note is that *you should not*, unless directed to by your support provider, *ever* directly manipulate the actual files within the resource database[1]. While they may be plain text, they should be treated like binary database files and edited with the appropriate tools instead.

In this case, the appropriate tool for editing the resource database is *nsradmin*.

4.2 Offline versus Online

NetWorker's *nsradmin* utility supports two distinct modes of accessing a resource database. These methods are:

- **Online** – Instead of interacting directly with the files, *nsradmin* interacts with the appropriate NetWorker daemons on an actively running NetWorker server in order to retrieve, review and update information.
- **Offline** – If the server is not currently running, *nsradmin* can instead be pointed at either a configuration file or database, and interact directly with these files. Certain "dynamic" parts of the configuration that depend on access to the media database, etc., are not presented in this mode.

When working with *nsradmin*, it's always very important to ensure that you choose the right method. A simple rule is that if the NetWorker server is running, you should *never, **ever*** attempt to use *nsradmin* directly against the files in the configuration database. Doing so could cause serious corruption to your NetWorker environment requiring a bootstrap recovery to restore functionality.

We'll focus on *online* mode for this manual.

4.3 Care and Respect

A tool like *nsradmin* is, in the right hands, very powerful and instrumental in helping a NetWorker administrator maximise the backup environment. However, misused, *nsradmin* can cause significant problems to a configured environment – it could lead to situations where backups can no longer function, or recoveries cannot be done, or even where the NetWorker server won't start.

You should always use *nsradmin* with a great deal of respect and care. (Let the user beware, so to speak.)

[1] To further reinforce this point: it would be very, very rare of a support provider to advise you to directly edit any of the resource files, either.

5 How to do the examples

Throughout the manual, there will regularly be examples of commands that you should run. For this reason, you are required for the purposes of the training to install a temporary instance of NetWorker on a spare host or virtual machine.

Our test/lab environment for this micromanual will therefore be one where you have:

1. **Installed** the NetWorker server/client/storage node software appropriate to your operating system, downloaded from PowerLink or from your own local repository, on a workstation or laptop.
2. **Have not** applied any license keys – this will allow the NetWorker server to run in evaluation mode for 30 days, which is more than enough time to make your way through the manual.
3. **Configured** two disk backup units – devices of type “ADV_FILE”.

Do not use this environment for production backups.

Throughout this manual, we will assume that on your temporary NetWorker server you’ve created the following components prior to continuing to the next chapter:

1. A group called “Test”.
2. A client instance for the NetWorker server, with one or two handpicked directories as the save sets. Optimally, you should be looking for a total backup size of between 1 and 3 GB, so that there is enough occupied space to be able to observe backups, but not so much space that it takes a lengthy time for examples to finish.
3. A backup pool called “Test” that has the “Test” group assigned to it.
4. A backup *clone* pool called “Test Clone”.
5. Two advanced file type disk backup units:
    a. One labelled in the “Test” pool.
    b. The other labelled in the “Test Clone” pool.

Refer to “Appendix A – Test Setup Configuration” (page 57) for instructions on establishing this configuration.

6 Operational Basics

6.1 About NetWorker Commands

In order to maintain common commands across platforms, NetWorker uses the same command line "switch" argument on both Unix/Linux platforms and Windows platforms – the "dash option" method. For instance, on Unix/Linux to specify the name of a pool in the save command, you would use "-b poolName", and on Windows, you'd use *exactly the same* rather than say, "/b poolName".

Unless otherwise noted, you should find that any examples given in this manual will work on both a Windows system *and* a Unix/Linux system, regardless of any command prompt preceding the command. (Where there are sufficient differences between platforms, both Unix/Linux *and* Windows examples will be given.)

Most NetWorker commands do not technically support any *usage* request in the traditional way –instead, they will print usage information in response to any unknown command line option. Therefore, when wanting to see what options a NetWorker command will take, it's usually safe to run the command with a "-?" argument.

6.2 Running nsradmin

In order to run *nsradmin* on a host, you must *at least* have the NetWorker client software installed. On Unix/Linux platforms, *nsradmin* will usually be installed into */usr/sbin*, and on Windows platforms, the default install location will be "C:\Program Files\Legato\nsr\bin" (it should however be in the execution path).

To see the usage options for *nsradmin*, run:

```
[root@tara ~]# nsradmin -?
usage: nsradmin [-c] [-i file] [-s server] [-p {prognum | progname} ]
                [-v version] [query]...
usage: nsradmin [-c] [-i file] [-d resdir] [-t typefile] ... [query]...
usage: nsradmin [-c] [-i file] [-f resfile] [-t typefile] ... [query]...
```

When run with no arguments, *nsradmin* expects to connect to a NetWorker server running on the current host, and enter interactive mode:

```
[root@tara ~]# nsradmin
NetWorker administration program.
Use the "help" command for help, "visual" for full-screen mode.
nsradmin>
```

If the current machine is just a client however, you'll get an error sequence such as the following:

```
[root@fawn ~]# nsradmin
39078:nsradmin: RPC error: Program not registered

There does not appear to be a NetWorker server running on fawn.pmdg.lab.
```

(Note that the actual error output may vary depending on the version of NetWorker in use, and the platform you are running the command from.)

6.3 Syntax Overview

As with other NetWorker interactive commands, one of the most important keywords to remember when using *nsradmin* in interactive mode is *help*. Let's see what *nsradmin* will tell us about what it can do.

For the purposes of this topic, it is safe to connect to a running NetWorker server, since we will only be *looking* at the configuration, rather than manipulating it.

If you are running *nsradmin* on the NetWorker server, you can run *nsradmin* without any arguments; if you are running *nsradmin* from another client, you will need to run: *nsradmin –s serverName* as your command. Note however that you'll need to be authorised as a NetWorker administrator from the host (and user account) you connect from.

For instance, running *nsradmin* from a storage node, *fawn* in my lab and connecting to a backup server *tara*, I would run the command:

```
[root@fawn ~]# nsradmin -s tara
NetWorker administration program.
Use the "help" command for help, "visual" for full-screen mode.
nsradmin>
```

Once *nsradmin* is running in interactive mode, you'll see that it prints a "nsradmin>" prompt every time it expects you to *start* a command.

To get a list of the commands available, type *help* at the nsradmin> prompt:

```
nsradmin> help
Legal commands are:
         bind [query]
       create attrlist
       delete [query]
         edit [query]
         help [command]
        print [query] (set current query)
       server [name]
         show [attrlist]
        types
       update attrlist
       append attrlist
         quit
       visual [query]
       option [list]
        unset [list]
            . [query]
            ? [command]

Where:
      query ::= attrlist
      attrlist ::= attribute [; attribute]*
      attribute ::= name [: [value [, value]* ]
```

Note that not all commands will be the same on Windows and Unix/Linux. For instance, while "edit" and "visual" appear as valid commands when running help on Windows, attempting to use them will result in a warning message that the function isn't available – for example:

```
nsradmin> edit
External editors are not yet supported
edit operation failed.
```

(The "edit" and "visual" modes will not be central to our examples, and so the lack of support for these options on Windows will not present an issue.)

You can get additional information about a command by typing "*help command*", or "*? command*", such as:

```
nsradmin> help show

usage: show [attrlist]
      The show command is used to set and clear the show list, which
      determines which attributes will be displayed in the"print"
      command. If an argument attribute list is given, these attributes
      are added to the show list. If no argument is given, the show list
      is cleared so that all attributes will be printed. For example, to
      show only the attribute "name" you would type:

      show name

nsradmin> ? print

usage: print [query] (set current query)
      The print command sets the current query if a query argument is
      given, then it prints the resource descriptors that match the
      current query. If the show list is set (using the "show" command),
      only the attributes in the show list will be printed. For example,
      to print all resources of type "NSR client" you would type:

      print type: NSR client
```

Another "getting started" command that you should know of is the *types* command – this gives you a list of the supported NetWorker resource types for the server that you are running on. This will vary, of course, depending on the version of NetWorker that you are using:

```
nsradmin> types
                 Known types: NSR, NSR client, NSR device, NSR directive,
                              NSR group, NSR jukebox, NSR label, NSR license,
                              NSR notification, NSR policy, NSR pool,
                              NSR schedule, NSR Snapshot Policy, NSR stage,
                              NSR Storage Node;
```

Every resource has a *type* – so clients are of type "NSR client", policies are of type "NSR policy", etc.

To get a handle on *nsradmin* syntax, let's start by trying to view that Month policy we initially looked at in "4.1 Resource Database" (starting page 9). As you can see from the above, one of the resource types is *NSR policy*. To view a resource, you'd typically use the *print* command. From the help on the previous page, you'll recall that print has the syntax:

```
        print [query] (set current query)
```

(We'll get to "(set current query)" in a moment.)

A *query*, as we saw from the help:

```
Where:
      query ::= attrlist
      attrlist ::= attribute [; attribute]*
      attribute ::= name [: [value [, value]* ]
```

So, to start with, we just want to see all the policies. This means we're just after all the resources that are of type "NSR policy". This means our query consists of just a single attribute to start with, and that attribute is the following "name:value" pair:

```
type: NSR policy
```

So our total command is:

```
nsradmin> print type: NSR policy
                        type: NSR policy;
                        name: Month;
                     comment: ;
                      period: Months;
           number of periods: 1;

                        type: NSR policy;
                        name: Quarter;
                     comment: ;
                      period: Months;
           number of periods: 3;

                        type: NSR policy;
                        name: Year;
                     comment: ;
                      period: Years;
           number of periods: 1;

                        type: NSR policy;
                        name: Decade;
                     comment: ;
                      period: Years;
           number of periods: 10;

                        type: NSR policy;
                        name: Week;
                     comment: ;
                      period: Weeks;
           number of periods: 1;

                        type: NSR policy;
                        name: Day;
                     comment: ;
                      period: Days;
           number of periods: 1;
```

Whoa! As you can imagine, if we just query by type all the time, we're going to get way too much information and we're going to rely on a long scroll buffer in our terminal session.

As it happens, our "Month" policy was the first one printed. However, let's assume that we've got a *lot* of resources. (For instance, if you're dealing with a NetWorker server that has 300 active clients, there'll be *at least* 300 client resources – likely more, since each new type of backup for a client will typically have its own client definition.)

If we want to limit the amount of information shown, we can use the "show" command. In this case, let's say we just want to see the resource "name" – nothing else. So our command would be:

```
nsradmin> show name:
nsradmin> print
                        name: Month;

                        name: Quarter;
```

```
                         name: Year;

                         name: Decade;

                         name: Day;

                         name: Week;
```

Now, we've done something tricky there. See that "print" command? We didn't specify a query. We didn't need to, because of that other part of the syntax description for print, which said: "(set current query)". That's right, when you issue a *print <query>* command, you implicitly tell NetWorker "Show me X *and* make the *default* query X so that I have a shortcut." So in this case, because we'd previously issued the command "print type: NSR policy", NetWorker interpreted our second print command to *also* be "print type: NSR policy". This query-set behaviour will be instrumental in later activities.

By the way, the colon (:) following the attribute in the show command *is optional*; you might wish to skip using it, but I think it adds clarity and consistency to commands, given that colons are required to separate attributes from their values in other commands.

If you had a lot of resources listed, you could then pick out the name of the resource you wanted, and issue a more specific print command to *nsradmin*, making use of that "attrlist" part of the query specification – more than one attribute, semi-colon separated. In this case, our query specification will become:

```
type: NSR policy; name: Month
```

Note that we *don't* terminate with a semi-colon. The semi-colon is a attribute *separator*; if you terminate the query specification with a semi-colon, *nsradmin* won't do anything, because it will be expecting you to put in *another* attribute. So using this updated query, we get:

```
nsradmin> print type: NSR policy; name: Month
                        name: Month;
```

Hold on though, there's more to a policy than just it's name. Remember – we previously used the "show" command to restrict ourselves to just viewing the name. This will continue to operate until such time as we issue another "show" command. In this case, entering "show" by itself will revert to the standard behaviour of showing everything, meaning our command and output will look like the following:

```
nsradmin> show
Will show all attributes
nsradmin> print
                        type: NSR policy;
                        name: Month;
                     comment: ;
                      period: Months;
           number of periods: 1;
```

Again, you'll note that I made use of the query short-cut behaviour – after issuing the "show" command, I didn't issue the full print command again, I just typed "print", and *nsradmin* filled in the rest for me.

One final thing that we want to look at before finishing is the display *options*. These control how much information *nsradmin* gives you when you run a query. You can view the currently enabled options by typing the "option" command. As of NetWorker 7.5.x, these options are:

```
nsradmin> option

Display options:
      Dynamic: Off;
```

```
      Hidden: Off;
      Raw I18N: Off;
      Resource ID: Off;
      Regexp: Off;
```

(It should be noted that the final option is not really a display option, but an input option – we'll get to that later though.)

To turn a particular option on, you use the command:

```
nsradmin> option <feature>
```

To turn it back off, you can either use the command:

```
nsradmin> option <feature>: off
```

Or:

```
nsradmin> unset <feature>
```

For instance, let's turn on "hidden" details, and check the Month policy again:

```
nsradmin> print type: NSR policy; name: Month
                        type: NSR policy;
                        name: Month;
                     comment: ;
                      period: Months;
           number of periods: 1;
nsradmin> option hidden
Hidden display option turned on

Display options:
      Dynamic: Off;
      Hidden: On;
      Raw I18N: Off;
      Resource ID: Off;
      Regexp: Off;
nsradmin> print
                        type: NSR policy;
                        name: Month;
                     comment: ;
                      period: Months;
           number of periods: 1;
                    hostname: tara.pmdg.lab;
               administrator: "user=root,host=tara.pmdg.lab",
                              "user=administrator,host=tara.pmdg.lab",
                              "user=system,host=tara.pmdg.lab";
          ONC program number: 390109;
          ONC version number: 2;
               ONC transport: TCP;
```

If you've used the "Diagnostic" mode in NetWorker Management Console in the past, you'll see over time that the additional attributes that come in when you add turn on the hidden option mostly map to that mode in NMC.

The "dynamic" display option turns on the display of additional attributes that are considered intermittent. The "Raw IL8N" turns off rendering of internationalisation text; the "Resource ID" turns on

the display of each resource's unique ID attribute, and the "Regexp" option, as mentioned before, is more of an input option, allowing the use of (some) regular expressions.

If you happen to be using an older version of NetWorker and some of the examples suggest to show particular attributes that don't subsequently turn up when *you* run the command, your first port of call will be to turn on the *hidden* and *dynamic* display options – previous versions of NetWorker may not have always shown requested attributes if those modes weren't turned on.

When you're done with *nsradmin*, you can exit it by typing in the command "quit":

```
nsradmin> quit
```

That's a basic introduction to NetWorker syntax. We'll move on to more interesting commands next.

6.4 Starting and Stopping Backups

6.4.1 What you'll need

In order to complete this section of the guide, you'll need to have configured a Test setup as per section "5 How to do the examples" (page 11). If you are unsure of how to configure this, please refer to "Appendix A – Test Setup Configuration" (page 57). As a result of this, you should have:

- An ADV_FILE type device with a volume labelled in the "Test" backup pool.
- An ADV_FILE type devices with a volume labelled in the "Test Clone" backup pool.
- A group called "Test".
- A client instance for the backup server that has saveset(s) of around 1-3GB.

6.4.2 Monitoring

On Unix/Linux platforms, you may monitor what we're about to do by running *nsrwatch* in another terminal session. Alternatively, on any platform you can monitor what we're about to do under the "Monitor" tab for NetWorker Management Console.

6.4.3 Starting a Backup

From within *nsradmin*, like within NetWorker Management Console[2], the only type of backup that you can start at the server interface is a *group*. There are three ways that a group can be started:

- Automatically, at it's scheduled time or part of a probe schedule.
- From the command line on the server by running the appropriate *savegrp* command.
- Within NetWorker Management Console or *nsradmin* by adjusting it's autostart property to "Start Now".

If you've only ever manually run a group in NetWorker Management Console, and you never recall changing the autostart property to "Start Now", don't be concerned. When you start a group out of the Monitoring area of NMC, you don't see this option, but that's what NetWorker does in the background for you.

While flexible, a primary failing of running a savegroup manually from the command line is that it cannot be aborted from within either NetWorker Management Console or from a utility such as *nsradmin*. This makes managing the backup somewhat challenging. Starting the group from within NMC or *nsradmin* takes away that issue though.

---

[2] As of the end of 2009.

So, let's look at manually starting our backup using *nsradmin*. From the backup server, run:

```
# nsradmin
nsradmin> print type: NSR group; name: Test
                        type: NSR group;
                        name: Test;
                     comment: ;
                    snapshot: False;
                   autostart: Disabled;
                 autorestart: Disabled;
                  start time: "3:33";
                  last start: ;
                    last end: ;
                    interval: "24:00";
              restart window: "12:00";
           force incremental: Yes;
         savegrp parallelism: 0;
              client retries: 1;
                      clones: No;
                  clone pool: Default Clone;
           success threshold: Warning;
                     options: ;
                       level: ;
                     printer: ;
                    schedule: ;
               schedule time: ;
             expiration time: ;
          inactivity timeout: 30;
   File inactivity threshold: 30;
File inactivity alert threshold: 30;
                   work list: ;
                  completion: ;
                      status: idle;
             Snapshot Policy: Daily;
               Snapshot Pool: Default;
           probe based group: False;
              probe interval: 60;
            probe start time: "0:00";
              probe end time: "23:59";
      probe success criteria: all;
time since successful backup: 0;
time of the last successful backup: ;
nsradmin> update autostart: Start Now
                   autostart: Start Now;
Update? y
updated resource id 115.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(2)
```

So to summarise the above, we:

- Printed, and therefore set the current query to "type: NSR group; name: Test".
- Issued an **update** command against the attribute "autostart", telling NetWorker to change its value to *Start Now*.
- When prompted to confirm by NetWorker, answered *y* to have it go and make the change.

This introduces a new command in *nsradmin*, the **update** command. This instructs *nsradmin* to *alter the resources that match the current query* to use the attribute values we're about to specify. What's important here is that it works on the *current query*. If you were to run *nsradmin* and attempt to run the **update** command without first establishing a query, *nsradmin* will use the *default* query, which maps to all resources. This, to be blunt, is something you would normally not want to do.

So without the output at all, our interaction with *nsradmin* was:

```
nsradmin> print type: NSR group; name: Test
nsradmin> update autostart: Start Now
<answer y for yes>
```

Now, the first thing that you might note is that a group, being a more complex resource than a policy, produces significantly more output on a *print* statement. There's two ways we can effectively reduce this. The first reduction method, which we've already used, is to use the *show* command first. In this case, we might ask *nsradmin* to only show us the name, type and autostart attribute of the resource. In this case, our session would look like the following:

```
nsradmin> show name:; type:; autostart:
nsradmin> print type: NSR group; name: Test
                        type: NSR group;
                        name: Test;
                   autostart: Disabled;
nsradmin> update autostart: Start Now
                   autostart: Start Now;
Update? y
updated resource id 115.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(7)
```

This gives the flexibility of seeing a "bare minimum" to confirm that the correct details are going to be updated.

The second way that you can limit the amount of details to be shown is to use the "dot" command, which sets the current query *but does not print* the results of the query. This is something that you should be careful using – while reducing the amount of output on the screen helps to achieve clarity, going for none can result in a situation where your query "misses" and selects too much but you don't notice before you do an update or delete command. In general I'd usually recommend that you reserve the "dot" command only for situations where:

- You're scripting, and you've already tested the query
- You've become sufficiently trained in *nsradmin* that you are very comfortable that you know what you're doing *and* you can do a bootstrap recovery at the drop of a hat.

The "dot" command sees the "print" command literally replaced with a full-stop (period). Thus, using the dot command, we could start the Test group using the following sequence:

```
nsradmin> . type: NSR group; name: Test
Current query set
nsradmin> update autostart: Start Now
                   autostart: Start Now;
Update? y
updated resource id 115.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(12)
```

The next thing to consider is that we've been *changing* the autostart attribute from its current value (in our case, Disabled) to "Start Now". Normally when you update a value, you'd expect to see that update stick, right? Well, normally that is the case, but the *autostart* attribute of the group resource (as well as a few other attributes across various resources) is a special attribute that supports both regular value changes as well as *action* settings. In this case, the value settings permitted are **Enabled** and **Disabled**. The *action* setting permitted is **Start Now**. When an attribute is updated with an action setting, NetWorker will start the action requested, but leave the attribute value in its previous state.

6.4.4 Stopping a Running Backup

If a backup has either been started from within NMC/*nsradmin*, or started automatically as a scheduled event, you can use *nsradmin* to abort the group. We do this by using another attribute, the "Stop Now" attribute, and a special action setting of "True".

Following is an example of stopping a group:

```
nsradmin> show name:; type:; autostart:; stop now:
nsradmin> print type: NSR group; name: Test
                        type: NSR group;
                        name: Test;
                   autostart: Disabled;
                    stop now: False;
nsradmin> update stop now: True
                    stop now: True;
Update? y
update failed: Groups must be started either automatically or from the GUI and must be
currently running in order to be stopped.
```

Now, see what happened above? The previous section went on for long enough that our group finished, and so there was nothing to stop! (Incidentally, you'll get the above error message if the group has been run from the command line using the *savegrp* command.)

Our challenge here with backing up a single saveset repeatedly is that the incremental changes are going to be very minor. For the time being, let's make sure that every time we run this group, a full backup is attempted. To do this, we'll want to:

- Set the **level** attribute of the group to *full*.
- Set the **force incremental** attribute of the group to *No*.

Our command will therefore be:

```
nsradmin> show type:; name:; force incremental:; level:
nsradmin> print type: NSR group; name: Test
                        type: NSR group;
                        name: Test;
           force incremental: Yes;
                       level: ;
nsradmin> update force incremental: No; level: full
           force incremental: No;
                       level: full;
Update? y
updated resource id 115.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(17)
```

Doing this will ensure that when we start the group, we'll have time to stop it before it finishes naturally. You'll also note we updated more than attribute in a single command.

So start the group again, but then after confirming in your *nsrwatch* or NMC monitoring session that the group is running, issue the stop command again:

```
nsradmin> show
Will show all attributes
nsradmin> show name:; type:; autostart:; stop now:
nsradmin> print type: NSR group; name: Test
                        type: NSR group;
                        name: Test;
                   autostart: Disabled;
                    stop now: False;
nsradmin> update autostart: Start Now
                   autostart: Start Now;
Update? y
updated resource id 115.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(18)

<wait until you can see that the group has started backing up>
```

```
nsradmin> update stop now: True
                    stop now: True;
Update? y
updated resource id 115.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(21)
```

6.4.5 Checking the status of a group

There's very little you can do in NMC or nsrwatch that you can't (in some form or another) achieve in *nsradmin*. Checking the status of a group is one of those things. There are a few attributes that you can look at within a group to determine its running status:

- **status** – Indicates whether the group is running, idle or cloning.
- **completion** – Details of the savesets that have completed, and how they finished.
- **work list** – Savesets that have not run yet (pending).

For example, if we look at our just-stopped group, we get some details about where it was up to when it was aborted, and its current state:

```
nsradmin> show name:; type:; status:; work list:; completion:
nsradmin> print type: NSR group; name: Test
                        type: NSR group;
                        name: Test;
                   work list: tara.pmdg.lab, "full:index", index;
                  completion: tara.pmdg.lab, /usr/share, "failed:full:save", "*
tara.pmdg.lab:/usr/share 66135:save: NSR directive file (/.nsr) parsed
* tara.pmdg.lab:/usr/share (interrupted), exiting
* tara.pmdg.lab:/usr/share aborted
     * <NOTICE> :  termination request was sent to job 19 as requested; Reason given was
'Aborted'";
                      status: idle;
```

There's quite a bit of information in the above output. The first thing to note is the format of the information in both the *work list* and the *completion* attributes. These are presented as multiple-component values. For *work list*, a set of 3 components will form a single value, where:

- The first component is the client the saveset is for.
- The second component indicates level and operation.
- The third component indicates the saveset.

So in this case, in the work list above with a value of:

```
tara.pmdg.lab, “full:index”, index
```

This equates to a single saveset – the full index backup for the client “tara.pmdg.lab” (which is the backup server), and an indication that the operation is an index backup.

For the completion, there's actually 4 components to each value – client, saveset, status and messages:

```
tara.pmdg.lab, /usr/share, "failed:full:save", "* tara.pmdg.lab:/usr/share 66135:save: NSR
directive file (/.nsr) parsed
* tara.pmdg.lab:/usr/share (interrupted), exiting
* tara.pmdg.lab:/usr/share aborted
     * <NOTICE> :  termination request was sent to job 19 as requested; Reason given was
'Aborted'";
```

To see the status change for the group while it's running, start the group again, wait a few seconds, and then print the details, with the same 'show' settings as per the above:

```
nsradmin> print type: NSR group; name: Test
                        type: NSR group;
                        name: Test;
                   work list: tara.pmdg.lab, "full:index", index;
                  completion: tara.pmdg.lab, /usr/share, "failed:full:save", "*
tara.pmdg.lab:/usr/share 66135:save: NSR directive file (/.nsr) parsed
* tara.pmdg.lab:/usr/share (interrupted), exiting
* tara.pmdg.lab:/usr/share aborted
     * <NOTICE> :  termination request was sent to job 19 as requested; Reason given was
'Aborted'";
                      status: idle;
nsradmin> update autostart: Start Now
                   autostart: Start Now;
Update? y
updated resource id 115.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(24)

<wait a few seconds before running the 'print' command below>

nsradmin> print
                        type: NSR group;
                        name: Test;
                   work list: tara.pmdg.lab, "full:save", /usr/share,
                              tara.pmdg.lab, "full:index", index;
                  completion: ;
                      status: running;
```

At this early stage in the group, we can see that the work list remains populated with both savesets that will be written:

```
                   work list: tara.pmdg.lab, "full:save", /usr/share,
                              tara.pmdg.lab, "full:index", index;
```

(Depending on how soon you run your command, your output may of course differ.)

You'll note that for the "/usr/share" saveset, the "level:operation" value is "full:save", rather than "full:index" as we saw previously (and above) for the index saveset. Since the status check was done before the first saveset was completed, the completion status remains empty, while the actual group status shows a value of 'running'.

6.4.6 Cloning and Monitoring

So far we've only seen two potential states for a group – *running*, or *idle*. There is a third state though – one that's provided when a group is cloning. To see what this state is like, we'll need to modify our group "Test" to clone to the "Test Clone" pool. This is readily accomplished by modifying the attributes as follows:

- Change the **clones** attribute to Yes
- Change the **clone pool** attribute to "Test Clone".

To better see what we're doing, we'll also use the "show" setting again to reduce the number of details of the group to a minimum – name, type, status, autostart, clones and clone pool.

The process of will work as follows:

- Set the show status appropriately.
- Print (and set the current query to) the Test group.
- Update the cloning attributes and run the group.
- Wait until the group starts cloning.
- Print the Test group again to view the new status.

If you're changing the attributes that you want to have shown, be sure to issue the "show" command by itself first to clear any previous settings.

Here's what the session will look like:

```
nsradmin> show name:; type:; status:; clones:; clone pool:; autostart:
nsradmin> print type: NSR group; name: Test
                        type: NSR group;
                        name: Test;
                   autostart: Disabled;
                      clones: No;
                  clone pool: Default Clone;
                      status: idle;
nsradmin> update clones: Yes; clone pool: Test Clone; autostart: Start Now
                   autostart: Start Now;
                  clone pool: Test Clone;
                      clones: Yes;
Update? y
updated resource id 115.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(31)

<wait for the backup to finish, and the cloning to start>

nsradmin> print
                        type: NSR group;
                        name: Test;
                   autostart: Disabled;
                      clones: Yes;
                  clone pool: Test Clone;
                      status: cloning;
```

Between group status checking, and device status checking (a topic we'll cover before the end of this chapter), you have the ability to fairly closely see what your NetWorker server is up to, just from within *nsradmin*.

6.5 Append vs Update

So far when we've been altering settings in NetWorker resources, we've been using the *update* command. There's another command, *append*, which works in a different, but equally useful way.

Let's move away from groups for the moment, and consider clients. In particular, one of the most critical attributes for a client is its *save set* setting. While normally this should be set to "All" for filesystem backup client instances, there may be times when it's necessary to have individually named save sets[3].

To check out our current client save set status, let's have a look at its settings, restricting just to type, name and save set. Assuming you've just come from doing the group section before, you'll first need to clear your show settings:

```
nsradmin> show
Will show all attributes
nsradmin> show name:; type:; save set:
nsradmin> print type: NSR client; name: tara
                        type: NSR client;
                        name: tara.pmdg.lab;
                    save set: /usr/share;
```

[3] A time that springs most readily to mind is when completing the exercises in the "nsrd.info micromanual for nsradmin".

(In the above example, be sure to replace the backup server name with the name of the host you're using to work through this micromanual.)

Now that we've got the client details available to us, let's consider the difference between *update* and *append*:

- The *update* command tells *nsradmin* to completely replace the current attribute values with whatever values you specify;
- The *append* command tells *nsradmin* to add, to multi-value attributes, additional values you specify.

(It should be clear from the above that you can't use append if the attribute field doesn't support it – e.g., you can't *append* "Start Now" to the autostart attribute.)

If we wanted to change the client definition so that it had an extra saveset – keeping /usr/share, but adding another one – say for instance, "/etc", using *update* our work would be twice as much effort. Since an *update* command is an instruction to *nsradmin* to replace the value of an attribute, you'd have to specify saveset values of both /usr/share *and* /etc. However, to make life easier for us, we can instead use the *append* command. Our entire sequence is therefore:

```
nsradmin> show
Will show all attributes
nsradmin> show name:; type:; save set:
nsradmin> print type: NSR client; name: tara
                        type: NSR client;
                        name: tara.pmdg.lab;
                    save set: /usr/share;
nsradmin> append save set: /etc
                    save set: /etc;
Append? y
updated resource id 74.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(5)
nsradmin> print
                        type: NSR client;
                        name: tara.pmdg.lab;
                    save set: /usr/share, /etc;
```

There you go – a time saving update thanks to the *append* command.

Since in this case, Windows will be slightly different due to the need to escape backslashes (i.e., use \\ instead of \) and keep paths in quotes, we'll look at how this would work on a Windows client as well:

```
C:\WINDOWS\system32\cmd.exe - nsradmin

C:\Documents and Settings\preston>nsradmin
NetWorker administration program.
Use the "help" command for help.
nsradmin> show name:; type:; save set:
nsradmin> print type: NSR client; name: cyclops
                        type: NSR client;
                        name: cyclops.pmdg.local;
                    save set: "C:\\WINDOWS\\SYSTEM32";
nsradmin> append save set: "C:\\TEMP"
                    save set: "C:\\TEMP";
Append? y
updated resource id 75.0.44.15.0.0.0.0.98.36.57.75.0.0.0.0.192.168.50.5(3)
nsradmin> print
                        type: NSR client;
                        name: cyclops.pmdg.local;
                    save set: "C:\\WINDOWS\\SYSTEM32", "C:\\TEMP";
nsradmin> _
```

Figure 2: Using the append command with Windows paths

As an aside, remember previously I said you couldn't *append* a value to a single-value attribute. Here's an example of what you'd get if you tried:

```
nsradmin> print type: NSR group; name: Test
                     type: NSR group;
                     name: Test;
nsradmin> append autostart: Start Now
                autostart: Start Now;
Append? y
append failed: autostart has too many values
```

6.6 Setting up Regular Backup Components

So far, we've relied on our resources already existing, and we've just been modifying them or getting NetWorker to perform specific actions with them. Now however, we want to look at *creating* new resources.

To do this, we're going to setup the core components that would typically be used in a new NetWorker configuration, notably:

1. Browse/Retention Policies
2. Schedules
3. Groups
4. Clients
5. Pools

Rather than using the NetWorker Management Console for any of these, we'll do the complete setup from within *nsradmin*.

6.6.1 Browse and Retention Policies

As you would know from using NetWorker Management Console, a policy is neither a browse, nor a retention policy, until you go ahead and assign it to the appropriate setting for a client or a pool. We'll setup both "Daily" and "Monthly" policies, with the policy details being:

- Daily – Defining a period of 5 weeks.
- Monthly – Defining a period of 13 months.

When creating a new NetWorker resource and you're not familiar with *nsradmin*, the easiest way to get a handle on it is to look at an existing resource. We already know from previous sections that there's a Month policy, so let's look at that again:

```
nsradmin> print type: NSR policy; name: Month
                        type: NSR policy;
                        name: Month;
                     comment: ;
                      period: Months;
           number of periods: 1;
```

So, this tells us of the attributes that we're going to need to set:

- type
- name
- period
- number of periods

A quick way of seeing some of the period *types* that are available is to restrict our display to just the period, and then view all policies:

```
nsradmin> show period:
nsradmin> print type: NSR policy
                      period: Months;

                      period: Years;

                      period: Weeks;

                      period: Months;

                      period: Days;

                      period: Years;
nsradmin> show
Will show all attributes
```

This won't always work – sometimes the preconfigured resources won't tell you enough information. In those cases it's useful to resort to the command reference guides. For instance, on Unix/Linux, you can view the documentation for any resource type by running "man nsr_*type*" – e.g., in this instance, "man nsr_policy". (These manual pages are included in the NetWorker command reference documentation.)

So, we want to create a Monthly policy that gives a time period of 13 months, and a Daily policy that gives a time period of 5 weeks. Our commands will then be:

```
nsradmin> create type: NSR policy; name: Daily; period: Weeks;
number of periods: 5
                        type: NSR policy;
                        name: Daily;
                      period: Weeks;
           number of periods: 5;
Create? y
created resource id 120.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(1)
nsradmin> create type: NSR policy; name: Monthly; period: Months;
number of periods: 13
                        type: NSR policy;
                        name: Monthly;
```

```
                    period: Months;
         number of periods: 13;
Create? y
created resource id 121.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(1)
```

Note:

- You can see above that the command was spread across two lines. Commands that deal with resources can either be specified as a single, very long line, or as multiple lines, so long as there's a semi-colon at the end of each non-finishing line (i.e., each line finishes with "attribute: value;") so that *nsradmin* knows to expect more input on the command.

If you want to verify that these have been created, issue a print command against them:

```
nsradmin> print type: NSR policy; name: Monthly
                      type: NSR policy;
                      name: Monthly;
                   comment: ;
                    period: Months;
         number of periods: 13;
nsradmin> print type: NSR policy; name: Daily
                      type: NSR policy;
                      name: Daily;
                   comment: ;
                    period: Weeks;
         number of periods: 5;
```

That's our policies setup and now waiting for clients and pools to be associated with them.

6.6.2 Schedules

Our next stop is to configure a Daily and a Monthly schedule for our backups. These will work as follows:

- **Daily schedule** – Fulls on Friday, incrementals the rest of the time, with the last Friday of every month skipped to allow for the Monthly schedule to run.
- **Monthly schedule** – Skips every day of the month, except for the last Friday of the month, where it does a full backup.

If you've only ever worked with schedules from within NMC, schedules are going to look a little peculiar to you in *nsradmin*. Let's look at the Default schedule first:

```
nsradmin> print type: NSR schedule; name: Default
                      type: NSR schedule;
                      name: Default;
                   comment: ;
                    period: Week;
                    action: full incr incr incr incr incr incr;
                  override: ;
```

If you're wondering what these attributes are for:

- **period** – Specifies the schedule period – either *week*, or *month*. This effectively defines how the *action* list is to be interpreted.
- **action** – A list of level backups to be performed on consecutive days. For a *week* schedule, this action list covers Sunday through to Saturday, *in that order*. For a month schedule, the action list covers the 1st through to the 31st, in that order.
- **override** – Any special changes to the backup to suit particular dates or special days.

If you think creating schedules would have to be easier in the GUI, I'll hopefully have you convinced otherwise once I take you through a series of short-cuts!

Let's first consider the Daily schedule that we want to create. We want:

- Fulls on Friday
- Incrementals the rest of the time
- Skip the last Friday every month

The first two requirements translate to attributes such that:

- **period** is defined as "Week"
- **action** is defined as "incr incr incr incr incr full incr"

We can take our first short-cut here, by the way – *nsradmin* doesn't need the full word for each of those levels, so we can shorten our action list to "i i i i i f i" instead. That's a lot less typing to start with. The next short-cut though is our override. We *don't* want this schedule to run on the last Friday of every month. In traditional calendar view creation of schedules in NMC, this would necessitate going through each month and setting a manual override on the final Friday of every month. However, like the non-calendar view of NMC, *nsradmin* supports a "set once" style override here, being:

- **override** defined as "skip last friday every month"

So, our create statement will look like the following:

```
nsradmin> create type: NSR schedule; name: Daily; period: Week;
action: i i i i i f i; override: skip last friday every month
                        type: NSR schedule;
                        name: Daily;
                      period: Week;
                      action: i i i i i f i;
                    override: skip last friday every month;
Create? y
```

Next we have to create our Monthly schedule, and that's going to involve an additional shortcut. You'll recall we want our Monthly schedule to:

- Skip every day of the month except for the last Friday of the month
- Do a full backup on the last Friday of the month

You could, if you wanted to, create an action list of "skip skip skip skip … skip" with 31 entries in it. We already know we can shorten the level names, so you could at least shorten it to "s s s s … s", with 31 instances of the letter "s" instead.

Another way to go about it would be to change the period type to "Week" and then just use an action list with only 7 entries in it: "s s s s s s s". That seems like a pretty good short-cut – but there's an even better short-cut yet. To get to that, we have to look at how the action list works.

As I mentioned before, for a schedule of period type "Month", the action list is typically defined as having 31 entries. What, you might ask, happens if the schedule is defined for 31 days however and the month it is evaluated for only has 30, 29 or 28 days? Well in these instances, any "extra" days in the action list are ignored. There's no 31 February, ever, so the 31st entry in an action list will never be evaluated in February.

Taking the opposite approach to this, action lists also support *fewer* entries than the defined time period, too. If you wanted for instance to create a backup schedule that did a full backup every day, rather than literally defining your action list as "full full full full full full full", you can instead define it just as "full". In this case, when NetWorker evaluates the schedule, if the action list is smaller than the defined period, it will simply keep on looping the action list on top of itself until it builds up an action list that matches the

defined period. So it would instantiate "full" to "full full full full full full full" without any additional effort on your part.

We can use that loop/fill action list behaviour as an "ultimate" short-cut to creating the Monthly schedule, and so our create command will become:

```
nsradmin> create type: NSR schedule; name: Monthly; period: Month;
action: s; override: full last friday every month
                        type: NSR schedule;
                        name: Monthly;
                      period: Month;
                      action: s;
                    override: full last friday every month;
Create? y
created resource id 124.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(1)
```

There you have it – a Monthly schedule that does exactly what we want created in minimal time.

This is one of those rare cases where it's particularly useful to see in NMC that what we did worked correctly:

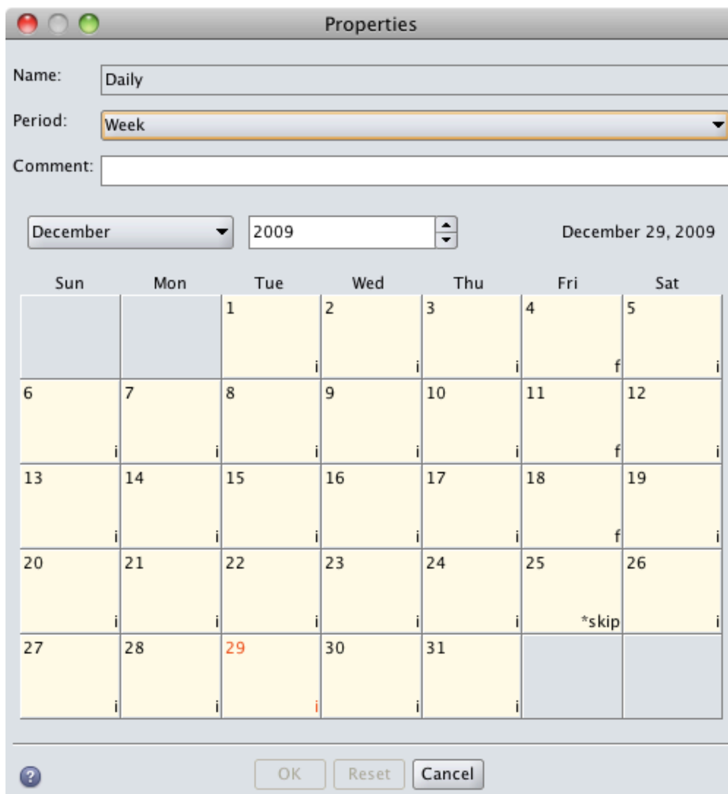

Figure 3: Daily schedule created in nsradmin, as viewed from NMC

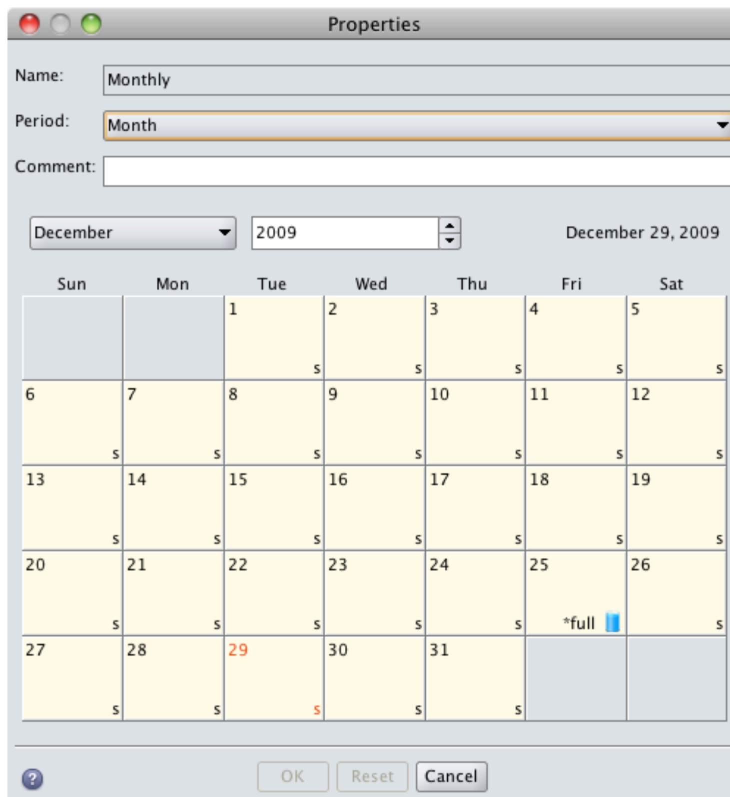

Figure 4: Monthly schedule created in nsradmin, as viewed from NMC

6.6.3 Groups

Now that we've created our schedules, we can move on to create our groups, since for the purposes of our example, we want to force all clients that backup as part of the group to run the same level each day.

As for when we created the policies and the schedules, we'll want to check out an existing group first to see what its settings are. Rather than using our current "Test" group, which will contain a bunch of additional information regarding recent backups, we'll pick a group that shouldn't have had a backup yet – the Default group:

```
nsradmin> print type: NSR group; name: Default
                        type: NSR group;
                        name: Default;
                     comment: ;
                    snapshot: False;
                   autostart: Disabled;
                 autorestart: Disabled;
                  start time: "3:33";
                  last start: ;
                    last end: ;
                    interval: "24:00";
              restart window: "12:00";
           force incremental: Yes;
         savegrp parallelism: 0;
              client retries: 1;
                      clones: No;
                  clone pool: Default Clone;
           success threshold: Warning;
                     options: ;
                       level: ;
```

```
                    printer: ;
                   schedule: ;
              schedule time: ;
            expiration time: ;
         inactivity timeout: 30;
   File inactivity threshold: 30;
File inactivity alert threshold: 30;
                  work list: ;
                 completion: ;
                     status: idle;
            Snapshot Policy: Daily;
              Snapshot Pool: Default;
          probe based group: False;
             probe interval: 60;
           probe start time: "0:00";
             probe end time: "23:59";
      probe success criteria: all;
time since successful backup: 0;
time of the last successful backup: ;
```

We want to create two groups. Our Daily group will start at 21:35, use the “Daily” schedule, and be configured to automatically start. So the attributes we want to set when creating are:

- **type** – NSR group
- **name** – Daily
- **autostart** – Enabled
- **start time** – “21:35”
- **schedule**: Daily

There are two things to note about the start time. The first, obviously, is that our start time is in the evening; that’s because the schedule we’re assigning to the group has full backups run on a Friday, so we’re talking about starting groups after the business day rather than before the business day.

The second, and far more important thing to note about the start time is that it’s specified within double-quotes. The reason for this is that the colon (:) is a special character for *nsradmin*, being used as a separator between an attribute name and its value. Therefore, if a value includes the colon character, the value must be in double quotes.

So our create command for the Daily group will be:

```
nsradmin> create type: NSR group; name: Daily; autostart: Enabled;
start time: "21:35"; schedule: Daily
                        type: NSR group;
                        name: Daily;
                   autostart: Enabled;
                  start time: "21:35";
                    schedule: Daily;
Create? y
created resource id 125.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(1)
```

Our create command for the Monthly group will be reasonably similar, bearing in mind that it should not start at *exactly* the same time as the Daily group, and it will need to use the Monthly schedule:

```
nsradmin> create type: NSR group; name: Monthly; autostart: Enabled;
start time: "21:40"; schedule: Monthly
                        type: NSR group;
                        name: Monthly;
                   autostart: Enabled;
                  start time: "21:40";
                    schedule: Monthly;
```

```
Create? y
created resource id 126.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(1)
```

Normally we'd also configure cloning, but we need to have pools setup for that, so we'll come back to that in a little while.

6.6.4 Clients

Our next step is to create a couple of clients. For the purposes of this sample configuration, we won't worry about having real clients. Instead, setup the *hosts* file on your NetWorker server with entries such as:

```
10.117.118.119       test1   test1.my.lab
10.117.118.120       test2   test2.my.lab
```

The above subnet is a *private* one, so shouldn't interfere with anything else. Check however before setting up these entries that these addresses really don't appear on your network. If they do, choose another set of entries in an empty 10.X.Y subnet.

On Unix/Linux systems, you can setup these entries by editing /etc/hosts; on a Windows system, the file will be typically be found in the "system32\drivers\etc" directory, named "hosts". (On a standard install of Windows 2003, for instance, you should expect to find the local hosts file with a full path of C:\windows\system32\drivers\etc\hosts.)

Next, we need to create these clients within NetWorker. We will follow the newer style of setting up clients in NetWorker, where each client belongs to *both* groups, using the longer browse/retention period, and the appropriate pool having the correct retention period specified as an override.

Again we'll look at an existing client resource to get an idea of the sorts of settings we'll want to specify. In this case, we'll look at the current client in the Test group – there should be only one, the one for our backup server itself:

```
nsradmin> print type: NSR client; group: Test
                        type: NSR client;
                        name: tara.pmdg.lab;
                      server: tara.pmdg.lab;
                   client id: \
923bf5a2-00000004-4b37bc9b-4b37bc9a-00011c00-dfb3d342;
            scheduled backup: Enabled;
                     comment: ;
             Save operations: ;
            archive services: Disabled;
                    schedule: Default;
               browse policy: Month;
            retention policy: Year;
                  statistics: elapsed = 102253, index size (KB) = 40244,
                              amount used (KB) = 40244, entries = 263529;
                   directive: ;
                       group: Test;
                    save set: /usr/share, /etc;
  Backup renamed directories: Disabled;
                    priority: 500;
   File inactivity threshold: 0;
File inactivity alert threshold: 0;
               remote access: ;
                 remote user: ;
                    password: ;
              backup command: ;
     application information: ;
                        ndmp: No;
```

```
             NDMP array name: ;
       De-duplication backup: No;
         De-duplication node: ;
         Probe resource name: ;
              virtual client: No;
               physical host: ;
           Proxy backup type: ;
           Proxy backup host: ;
             executable path: ;
    server network interface: ;
                     aliases: tara, tara.pmdg.lab;
                  index path: ;
          owner notification: ;
                 parallelism: 12;
               archive users: ;
               storage nodes: nsrserverhost;
       recover storage nodes: ;
         clone storage nodes: ;
                  hard links: Disabled;
             short filenames: Disabled;
                         BMR: Disabled;
                 BMR options: ;
                 backup type: ;
              client OS type: Linux;
                        CPUs: 1;
           NetWorker version: 7.5.1.Build.413;
              enabler in use: Yes;
       licensed applications: ;
               licensed PSPs: ;
```

There's a lot of attributes there (and that's with dynamic/hidden attributes turned *off*), and we don't have to set all of those today. We'll just limit ourselves to:

- type
- name
- browse policy
- retention policy
- group
- save set
- parallelism
- aliases

As you may have guessed by now – NetWorker will fill in attributes that you leave out when the resource is created. This is quite handy, and works the same way that resource creation does within NetWorker Management Console.

Looking at our first client, test1, our create command will be:

```
nsradmin> create type: NSR client; name: test1;
browse policy: Monthly; retention policy: Monthly;
group: Daily, Monthly; save set: All; parallelism: 1;
aliases: test1, test1.my.lab
                        type: NSR client;
                        name: test1;
               browse policy: Monthly;
            retention policy: Monthly;
                       group: Daily, Monthly;
                    save set: All;
                     aliases: test1, test1.my.lab;
                 parallelism: 1;
Create? y
```

```
created resource id 127.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(1)
```

Our second client can be created with a similar command:

```
nsradmin> create type: NSR client; name: test2;
browse policy: Monthly; retention policy: Monthly;
group: Daily, Monthly; save set: All; parallelism: 1;
aliases: test2, test2.my.lab
                        type: NSR client;
                        name: test2;
               browse policy: Monthly;
            retention policy: Monthly;
                       group: Daily, Monthly;
                    save set: All;
                     aliases: test2, test2.my.lab;
                 parallelism: 1;
Create? y
created resource id 128.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(1)
```

While not all of these attributes are required, it's worthwhile when creating clients being in the habit of *at least* specifying the above. It forces you, for instance, to conduct performance tuning on the client backups by starting with a parallelism of 1, and reminds you to think about the different aliases a host may have.

6.6.5 Pools

Our next-to-last components to be setup are the pools. We'll define *four* pools – Daily, Daily Clone, Monthly and Monthly Clone. Don't worry; these pools are for demonstration purposes only, so we won't need a bunch of additional disk backup units or media.

Let's start, as we always do, by looking at an existing pool. In this case, we'll look at the Default pool:

```
nsradmin> print type: NSR pool; name: Default
                        type: NSR pool;
                        name: Default;
                     comment: ;
                     enabled: Yes;
                   pool type: Backup;
              label template: Default;
            retention policy: ;
                      groups: ;
                     clients: ;
                   save sets: ;
                      levels: ;
                     devices: ;
         store index entries: Yes;
           auto media verify: No;
      Recycle to other pools: No;
    Recycle from other pools: No;
      volume type preference: ;
             max parallelism: 0;
                 mount class: default;
                   WORM pool: No;
              create DLTWORM: No;
              barcode prefix: ;
```

As is always the case, there's a bunch of attributes in a pool that for regular setups we don't need to consider. For our setup though, we'll need to provide settings for:

- type
- name

- enabled
- pool type
- groups
- retention policy
- store index entries
- auto media verify
- Recycle to other pools
- Recycle from other pools

The last two entries, normally turned off for pools, are used to replace the need for a Scratch pool within NetWorker. Rather than putting media in one "special" pool to subsequently pull out when media is needed in another pool, NetWorker allows us to specify that pools can take (recyclable) media from other pools, and donate recyclable media to other pools.

The third last option, *auto media verify* should be something that you turn on for most, if not all pools. For a small performance hit, it actually does verification reads on parts of savesets, making it a powerful tool in ensuring your backups are recoverable. (Or rather, at least confirming that the media is readable.)

The "store index entries" is normally "on" for a pool, and that's fine for our backup pools, but when we make our backup clone pools, it will need to be off.

So, let's first look at our Daily pool, and how it is created:

```
nsradmin> create type: NSR pool; name: Daily; enabled: Yes; pool type: Backup;
groups: Daily; auto media verify: Yes; recycle to other pools: Yes;
recycle from other pools: Yes; retention policy: Daily
                        type: NSR pool;
                        name: Daily;
                     enabled: Yes;
                   pool type: Backup;
            retention policy: Daily;
                      groups: Daily;
           auto media verify: Yes;
      recycle to other pools: Yes;
    recycle from other pools: Yes;
Create? y
created resource id 129.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(1)
```

Note – if you get an alert about needing a label template, or to re-create the resource to automatically create a label template – it's time to upgrade to a more recent version of NetWorker!

Our Daily Clone pool will be quite similar – but we *don't* specify any groups for a Clone pool, and we tell NetWorker not to store index entries for it:

```
nsradmin> create type: NSR pool; name: Daily Clone; enabled: Yes;
pool type: Backup Clone; store index entries: No;
auto media verify: Yes; recycle to other pools: Yes;
recycle from other pools: Yes; retention policy: Daily
                        type: NSR pool;
                        name: Daily Clone;
                     enabled: Yes;
                   pool type: Backup Clone;
            retention policy: Daily;
         store index entries: No;
           auto media verify: Yes;
      recycle to other pools: Yes;
    recycle from other pools: Yes;
Create? y
created resource id 130.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(1)
```

Our Monthly, and Monthly Clone pools will be quite similar, with the appropriate basic modifications, namely:

1. For the Monthly Pool, the group changes from “Daily” to “Monthly”.
2. For both the Monthly and Monthly Clone pools, the retention policy changes from “Daily” to “Monthly”.

Our create commands will therefore resemble:

```
nsradmin> create type: NSR pool; name: Monthly; enabled: Yes;
pool type: Backup; groups: Monthly; auto media verify: Yes;
recycle to other pools: Yes; recycle from other pools: Yes;
retention policy: Monthly
                        type: NSR pool;
                        name: Monthly;
                     enabled: Yes;
                   pool type: Backup;
            retention policy: Monthly;
                      groups: Monthly;
           auto media verify: Yes;
      recycle to other pools: Yes;
    recycle from other pools: Yes;
Create? y
created resource id 131.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(1)
nsradmin> create type: NSR pool; name: Monthly Clone; enabled: Yes;
pool type: Backup Clone; store index entries: No; auto media verify: Yes;
recycle to other pools: Yes; recycle from other pools: Yes;
retention policy: Monthly
                        type: NSR pool;
                        name: Monthly Clone;
                     enabled: Yes;
                   pool type: Backup Clone;
            retention policy: Monthly;
         store index entries: No;
           auto media verify: Yes;
      recycle to other pools: Yes;
    recycle from other pools: Yes;
Create? y
created resource id 132.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(1)
```

We’re almost there! The last thing we have to do is modify our groups to automatically clone, which we’ll do by going back to our groups, below.

6.6.6 Revisiting our Groups

Now that our pools have been established, we can modify the Daily group to clone to the Daily Clone pool, and the Monthly group to clone to the Monthly Clone pool. We’ve previously done this when we configured the Test group to clone, so this process should be relatively straightforward:

```
nsradmin> show name:; clones:; clone pool:
nsradmin> print type: NSR group; name: Daily
                        name: Daily;
                      clones: No;
                  clone pool: Default Clone;
nsradmin> update clones: Yes; clone pool: Daily Clone
                  clone pool: Daily Clone;
                      clones: Yes;
Update? y
updated resource id 125.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(2)
nsradmin> print type: NSR group; name: Monthly
                        name: Monthly;
```

```
                    clones: No;
                clone pool: Default Clone;
nsradmin> update clones: Yes; clone pool: Monthly Clone
                clone pool: Monthly Clone;
                    clones: Yes;
Update? y
updated resource id 126.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(2)
```

At this point, we've completed a standard setup process covering policies, schedules, groups, clients and pools.

6.7 Monitoring Devices

Another basic activity that you should know how to do with *nsradmin* is monitoring devices while activities are currently running. This is relatively easy, and is handy to know how to do.

Let's look at a device. If you're on Unix/Linux, the command (and output) should be along the following lines:

```
nsradmin> print type: NSR device; name: /backup1
                        type: NSR device;
                        name: /backup1;
                     comment: ;
                 description: ;
                message_I18N: "writing, done ";
                     message: "writing, done ";
                 volume name: Backup.001;
                media family: disk;
                  media type: adv_file;
                     enabled: Yes;
                   read only: No;
             target sessions: 4;
                max sessions: 512;
              parent jukebox: ;
           cleaning required: No;
           cleaning interval: ;
          date last cleaned: ;
       auto media management: No;
                        ndmp: No;
      dedicated storage node: No;
                 remote user: ;
                    password: ;
                 hardware id: ;
                         CDI: Not used;
          TapeAlert Critical: ;
           TapeAlert Warning: ;
       TapeAlert Information: ;
               WORM capable: No;
            DLTWORM capable: No;
      WORM cartridge present: No;
        device serial number: ;
```

On Windows, change the name from "/backup1" to "X:\\NSR\\01", where X was the drive letter where the disk backup units were created. For example, this might resemble the following:

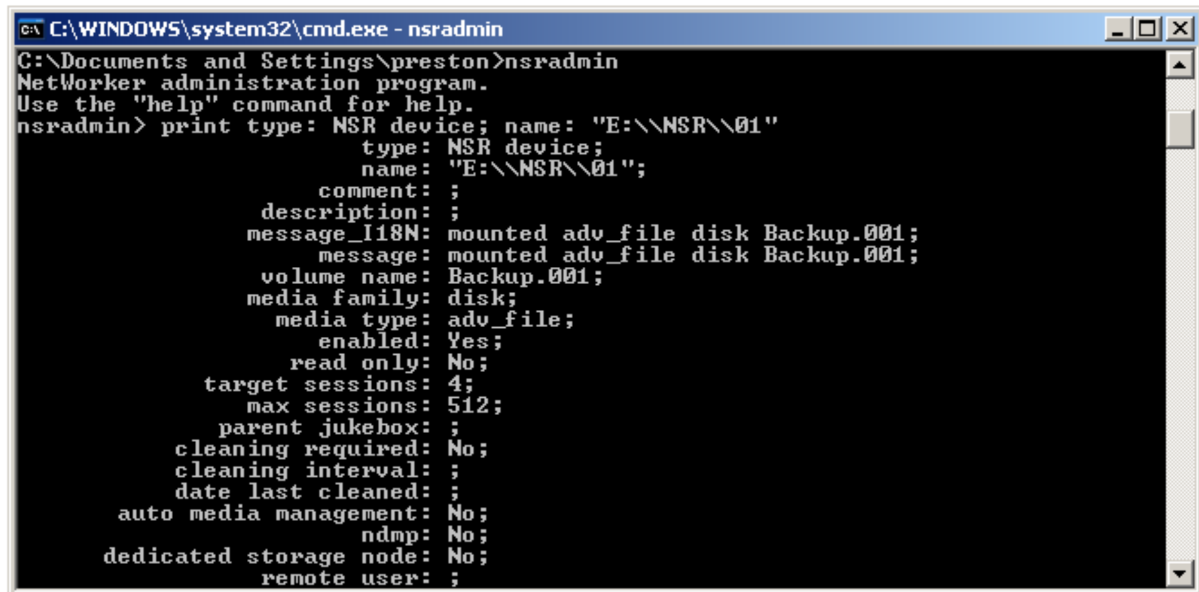

Figure 5: Viewing a device resource in Windows

The fields in particular that we want to look at when monitoring devices are:

- message
- message_I18N

What we'll now do is restart out *Test* group from before, and monitor what the devices do, both on backup, then clone.

To do this, we'll run through the following steps:

1. Start the Test group.
2. Update our show command to just show name, message, and message_I18N.
3. Print (and set the query for) the current devices.
4. Periodically re-print the status during the backup.

This will resemble the following:

```
nsradmin> . type: NSR group; name: Test
Current query set
nsradmin> update autostart: Start Now
                   autostart: Start Now;
Update? y
updated resource id 115.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(38)
nsradmin> show name:; message:; message_I18N
nsradmin> print type: NSR device
                        name: /backup1;
                message_I18N: "writing at 16 MB/s, 100 MB, 2 sessions";
                     message: "writing at 16 MB/s, 100 MB, 2 sessions";

                        name: /backup1/_AF_readonly;
                message_I18N: "reading, done ";
                     message: "reading, done ";

                        name: /backup2/_AF_readonly;
                message_I18N: space recovered from volume Clone.001.RO;
                     message: space recovered from volume Clone.001.RO;

                        name: /backup2;
```

```
                message_I18N: "writing, done ";
                     message: "writing, done ";
```

Once cloning starts, we should see output of the form:

```
nsradmin> print
                        name: /backup1/_AF_readonly;
                message_I18N: "reading, data ";
                     message: "reading, data ";

                        name: /backup1;
                message_I18N: "writing, idle ";
                     message: "writing, idle ";

                        name: /backup2/_AF_readonly;
                message_I18N: space recovered from volume Clone.001.RO;
                     message: space recovered from volume Clone.001.RO;

                        name: /backup2;
                message_I18N: "writing at 49 MB/s, 98 MB";
                     message: "writing at 49 MB/s, 98 MB";
```

(Note – if you are working in an English-only environment, you can choose to leave off the message_I18N attribute from your show command.)

As you can see by this – even if you have a NetWorker server running on Windows and can't get to NMC, you can at least check to see what the devices are doing.

6.8 Deleting Resources

Our last basic exercise sees us deleting our Daily/Monthly setup. This not only demonstrates the *delete* command, but also leaves you in a position to reattempt any of the exercises in the previous section if necessary.

There's two ways that the delete command can work:

- Run by itself, *delete*, it will offer to delete the resources that match the currently set query.
- Run with a query, *delete query* set the current query and offer to delete the resources that match that just-set query.

If you're wondering which you should use, I'll tell you now that I **never** run delete without specifying a query as well. Regardless of whether it is supported or not, I very strongly recommend against doing so. All examples provided here will work from providing the query with the delete command.

As you might imagine, NetWorker (usually) doesn't let you delete resources that have dependencies. For instance, you can't delete a group if it is still referenced by a pool, etc. So sometimes, in order to delete, we have to backtrack part of the configuration.

The first resources that we can delete though are the clients – these aren't named anywhere else, so it's safe to get rid of them with the commands:

```
nsradmin> show name:; save set:; group:
nsradmin> delete type: NSR client; group: Daily
                        name: test1;
                       group: Daily, Monthly;
                    save set: All;
Delete? y
deleted resource id 127.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(1)
```

```
                        name: test2;
                       group: Daily, Monthly;
                    save set: All;
Delete? y
deleted resource id 128.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(1)
```

You'll note there that we didn't issue a delete command for each client, but rather, on an attribute we knew to be unique for all of the clients we needed to delete – in this case, belonging to the Daily group.

Moving on, it would be handy if we deleted our groups – Daily and Monthly. However, these currently belong to the Daily and Monthly pools – if we try to unassign those groups from the pools, we'll get in trouble from NetWorker since pools must have unique attributes – e.g.,:

```
nsradmin> show name:; enabled:; groups:
nsradmin> print type: NSR pool; name: Daily
                        name: Daily;
                     enabled: Yes;
                      groups: Daily;
nsradmin> update groups:
                      groups: ;
Update? y
update failed: There must be at least one selection criterion
(groups, clients, save sets or levels) set
when creating or updating a non-clone pool.
```

You'll note there that I used the special way of *clearing* an attribute:

```
update groups:
```

There's an attribute there, but no value. That's how you take the current value away from an attribute without putting in a new one. (Now, in this case, because of pool configuration requirements, we weren't allowed to do that, but it's worth knowing how to do this.)

So, we'll first need to delete our Daily and Monthly pools:

```
nsradmin> delete type: NSR pool; name: Monthly
                        name: Monthly;
                     enabled: Yes;
                      groups: Monthly;
Delete? y
deleted resource id 131.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(1)
nsradmin> delete type: NSR pool; name: Daily
                        name: Daily;
                     enabled: Yes;
                      groups: Daily;
Delete? y
deleted resource id 133.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(1)
```

In older versions of NetWorker, you might have had to then go and clear the Daily Clone and Monthly Clone pools from the Daily and Monthly groups, but newer versions of NetWorker will unfortunately allow you to delete the Clone pool while still having it referenced!

```
nsradmin> delete type: NSR pool; name: Daily Clone
                        name: Daily Clone;
                     enabled: Yes;
                      groups: ;
Delete? y
deleted resource id 130.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(1)
nsradmin> delete type: NSR pool; name: Monthly Clone
```

```
                        name: Monthly Clone;
                     enabled: Yes;
                      groups: ;
Delete? y
deleted resource id 132.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(1)
```

Now we can move on to deleting the groups, which means we're heading into the home stretch:

```
nsradmin> show
Will show all attributes
nsradmin> show name:; clones:; clone pool:
nsradmin> delete type: NSR group; name: Daily
                        name: Daily;
                      clones: Yes;
                  clone pool: Daily Clone;
Delete? y
deleted resource id 125.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(2)
nsradmin> delete type: NSR group; name: Monthly
                        name: Monthly;
                      clones: Yes;
                  clone pool: Monthly Clone;
Delete? y
deleted resource id 126.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(2)
```

If you were wondering what would happen if NetWorker goes to run one of those groups, and then clone but the pool doesn't exist – the clone will fail. It's a situation best to be avoided.

With the groups, pools, and clients out of the way, we can now push through and delete our schedules:

```
nsradmin> delete type: NSR schedule; name: Daily
                        name: Daily;
Delete? y
deleted resource id 123.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(1)
nsradmin> delete type: NSR schedule; name: Monthly
                        name: Monthly;
Delete? y
deleted resource id 124.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(1)
```

You'll note there that the "show" command previously issued is still in effect; however, of the three attributes configured to be shown – name, clones and clone pool, only "name" is valid for a schedule, so only "name" is shown. We'll turn the show restrictions back off to delete our policies:

```
nsradmin> show
Will show all attributes
nsradmin> delete type: NSR policy; name: Daily
                        type: NSR policy;
                        name: Daily;
                     comment: ;
                      period: Weeks;
           number of periods: 5;
Delete? y
deleted resource id 120.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(1)
nsradmin> delete type: NSR policy; name: Monthly
                        type: NSR policy;
                        name: Monthly;
                     comment: ;
                      period: Months;
           number of periods: 13;
Delete? y
deleted resource id 121.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(1)
```

There you have it – we've now backtracked out of all the components we setup for our Daily/Monthly configuration.

6.9 Closing Comments

When it comes to *nsradmin*, practice makes perfect. If anything in this section hasn't made sense, or you want to repeat it, be sure to follow the steps in the previous section – 6.8, "Deleting Resources" (starting page 40), in order to reset the NetWorker configuration to try again.

7 Advanced nsradmin

7.1 Introduction

Our previous chapter (6, "Operational Basics", starting page 12) gave a reasonably thorough overview to how one works with *nsradmin* as part of an interactive session. This chapter will primarily deal with ways that one can interact with *nsradmin* in a non-interactive mode.

The nature of advanced *nsradmin* usage is that what you do in it will depend *entirely* on your local circumstances. Therefore, this chapter will be shorter than the previous chapter. That doesn't reflect that there's not many advanced things you can do with *nsradmin*, but instead more that those advanced things will need *you* to decide what to do.

7.2 Bulk Activities

Our first advanced use for *nsradmin* is in *bulk* activities. Let's say that an additional 20 servers are going to be added to your environment. They'll all be standard systems without databases, which means that either from using NMC, or *nsradmin* in interactive mode, there's going to be a lot of tedious work in setting them up.

However, we can do it as a non-interactive *nsradmin* session fairly quickly, using a combination of copy/paste and quick editing in a text file.

To start with, we're obviously going to need host entries again – NetWorker won't let us create clients without a means of resolving them, and the easiest way to get resolution running for a lab situation is to populate the hosts file.

Previously we discussed setting up hosts entries of the form:

```
10.117.118.119       test1   test1.my.lab
10.117.118.120       test2   test2.my.lab
```

Now, let's setup another 18, so that they read:

```
10.117.118.119       test1   test1.my.lab
10.117.118.120       test2   test2.my.lab
10.117.118.121       test3   test3.my.lab
10.117.118.122       test4   test4.my.lab
10.117.118.123       test5   test5.my.lab
10.117.118.124       test6   test6.my.lab
10.117.118.125       test7   test7.my.lab
10.117.118.126       test8   test8.my.lab
10.117.118.127       test9   test9.my.lab
10.117.118.128       test10  test10.my.lab
10.117.118.129       test11  test11.my.lab
10.117.118.130       test12  test12.my.lab
10.117.118.131       test13  test13.my.lab
10.117.118.132       test14  test14.my.lab
10.117.118.133       test15  test15.my.lab
10.117.118.134       test16  test16.my.lab
10.117.118.135       test17  test17.my.lab
10.117.118.136       test18  test18.my.lab
10.117.118.137       test19  test19.my.lab
10.117.118.138       test20  test20.my.lab
```

For the purposes of our example only, we'll use some more "Default" settings in NetWorker. In a real-world situation, we'd already have groups, schedules, pools, etc., setup. So instead of going through and setting all these up, we'll just specify the name, aliases and parallelism, and rely on NetWorker to establish the browse/retention policy as well as the group for us.

Next, create a text file that contains the following lines:

```
create type: NSR client; name: test1; aliases: test1, test1.my.lab; parallelism: 1
create type: NSR client; name: test2; aliases: test2, test2.my.lab; parallelism: 1
create type: NSR client; name: test3; aliases: test3, test3.my.lab; parallelism: 1
create type: NSR client; name: test4; aliases: test4, test4.my.lab; parallelism: 1
create type: NSR client; name: test5; aliases: test5, test5.my.lab; parallelism: 1
create type: NSR client; name: test6; aliases: test6, test6.my.lab; parallelism: 1
create type: NSR client; name: test7; aliases: test7, test7.my.lab; parallelism: 1
create type: NSR client; name: test8; aliases: test8, test8.my.lab; parallelism: 1
create type: NSR client; name: test9; aliases: test9, test9.my.lab; parallelism: 1
create type: NSR client; name: test10; aliases: test10, test10.my.lab; parallelism: 1
create type: NSR client; name: test11; aliases: test11, test11.my.lab; parallelism: 1
create type: NSR client; name: test12; aliases: test12, test12.my.lab; parallelism: 1
create type: NSR client; name: test13; aliases: test13, test13.my.lab; parallelism: 1
create type: NSR client; name: test14; aliases: test14, test14.my.lab; parallelism: 1
create type: NSR client; name: test15; aliases: test15, test15.my.lab; parallelism: 1
create type: NSR client; name: test16; aliases: test16, test16.my.lab; parallelism: 1
create type: NSR client; name: test17; aliases: test17, test17.my.lab; parallelism: 1
create type: NSR client; name: test18; aliases: test18, test18.my.lab; parallelism: 1
create type: NSR client; name: test19; aliases: test19, test19.my.lab; parallelism: 1
create type: NSR client; name: test20; aliases: test20, test20.my.lab; parallelism: 1

```

Save the file as "bulk-create.nsri". (Include a blank line at the bottom of the command file, so that regardless of which operating system you're on, there won't be any issues to do with end-of-file points.)

Now, in the previous section, every time we've gone to create resources in NetWorker, we've been prompted to confirm, "yes" to go ahead and create each resource. This isn't the case when we run in non-interactive mode. To invoke *nsradmin* in non-interactive mode, simply run:

```
# nsradmin -i file
```

Where "file" is the path to the file and its name. Note that while not mandatory, it is recommended to supply this as an absolute, rather than relative path, as at various times there have been bugs in *nsradmin* that have caused it to fail when being invoked with a command file or database from a relative path.

In this case, assuming we've saved the file as */tmp/bulk-create.nsri*, and we're not in that directory, we'd simply run:

```
[root@tara ~]# nsradmin -i /tmp/bulk-create.nsri
created resource id 134.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(1)
created resource id 135.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(1)
created resource id 136.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(1)
created resource id 137.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(1)
created resource id 138.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(1)
created resource id 139.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(1)
created resource id 140.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(1)
created resource id 141.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(1)
created resource id 142.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(1)
created resource id 143.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(1)
created resource id 144.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(1)
created resource id 145.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(1)
created resource id 146.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(1)
created resource id 147.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(1)
```

```
created resource id 148.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(1)
created resource id 149.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(1)
created resource id 150.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(1)
created resource id 151.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(1)
created resource id 152.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(1)
created resource id 153.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(1)
```

There you have it. With very little preparation time, you've added 20 clients to a NetWorker configuration. Not even NetWorker Fast Start will get you that level of speed saving!

Backing out of this configuration change is very easy using bulk operations as well. Let's create a text file now that has the following content:

```
delete type: NSR client; name: test1
delete type: NSR client; name: test2
delete type: NSR client; name: test3
delete type: NSR client; name: test4
delete type: NSR client; name: test5
delete type: NSR client; name: test6
delete type: NSR client; name: test7
delete type: NSR client; name: test8
delete type: NSR client; name: test9
delete type: NSR client; name: test10
delete type: NSR client; name: test11
delete type: NSR client; name: test12
delete type: NSR client; name: test13
delete type: NSR client; name: test14
delete type: NSR client; name: test15
delete type: NSR client; name: test16
delete type: NSR client; name: test17
delete type: NSR client; name: test18
delete type: NSR client; name: test19
delete type: NSR client; name: test20
```

Assuming this has been saved as */tmp/bulk-delete.nsri*, you can run:

```
[root@tara ~]# nsradmin -i /tmp/bulk-delete.nsri
deleted resource id 134.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(1)
deleted resource id 135.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(1)
deleted resource id 136.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(1)
deleted resource id 137.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(1)
deleted resource id 138.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(1)
deleted resource id 139.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(1)
deleted resource id 140.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(1)
deleted resource id 141.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(1)
deleted resource id 142.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(1)
deleted resource id 143.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(1)
deleted resource id 144.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(1)
deleted resource id 145.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(1)
deleted resource id 146.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(1)
deleted resource id 147.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(1)
deleted resource id 148.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(1)
deleted resource id 149.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(1)
deleted resource id 150.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(1)
deleted resource id 151.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(1)
deleted resource id 152.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(1)
deleted resource id 153.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(1)
```

As you can see, bulk updates are relatively straightforward.

7.3 Scripting with nsradmin

7.3.1 Intended Goal

Let's say you run an environment where you want the support staff responsible for setting up new servers to be able to have simple commands they can run to interactively fill in the details required for new clients within NetWorker.

This means being able to script with your NetWorker server.

Note: For the examples below scripts will be kept as basic as possible, performing no input validation, etc. For more comprehensive scripting of *nsradmin*, you are advised to (a) use a "good" scripting language, such as Perl or Python (rather than say, DOS shell scripting), and (b) perform input validation to ensure that the values you feed through to *nsradmin* are acceptable.

7.3.2 Introductory scripting

We'll start with something a little less ambitious, as a test case. Consider a scenario where (for some reason), you want a script that will create a new day-based policy for you after you supply a policy name and a number of days.

On Windows, you might call this script "create-policy.bat" and it would have the following content:

```
@echo off
echo Creating a new policy
set /p name="Enter policy name: "
set /p days="Enter number of days: "

> command.nsri echo create type: NSR policy; name: %name%;
>> command.nsri echo period: Days; number of periods: %days%
>> command.nsri echo print type: NSR policy; name: %name%

nsradmin -i command.nsri
del command.nsri
```

Running this might result in a session such as the following:

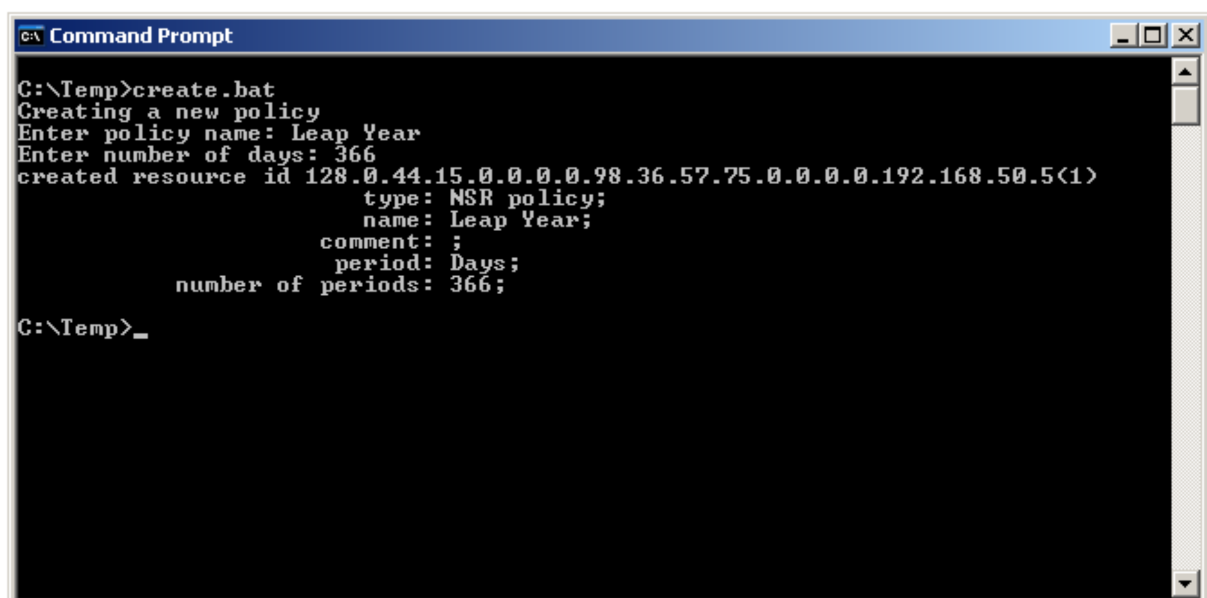

Figure 6: Using the create-policy.bat script on Windows

On Unix systems, using Perl, we could do a similar function with the following script, named "create-policy.pl" and made executable:

```
#!/usr/bin/perl -w

use strict;

print "Enter policy name: ";
my $policyName = <>; chomp $policyName;

print "Enter number of days: ";
my $numDays = <>; chomp $numDays;

if (open(CMD,">command-$$.nsri")) {
	print CMD "create type: NSR policy; name: $policyName;\n";
	print CMD "period: Days; number of periods: $numDays\n";
	print CMD "print type: NSR policy; name: $policyName\n";
	close(CMD);
	system("nsradmin -i command-$$.nsri");
	unlink("command-$$.nsri");
} else {
	die "Unable to create command-$$.nsri\n";
}
```

The output from such a script might resemble the following:

```
[root@tara ~]# ./create-policy.pl
Enter policy name: Leap Year
Enter number of days: 366
created resource id 154.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(1)
                        type: NSR policy;
                        name: Leap Year;
                     comment: ;
                      period: Days;
           number of periods: 366;
```

7.3.3 Preliminary Setup

In order for a client creation script to work, we have to work on the basis of there being a bunch of resources already present – pools, groups and schedules are a good start. To make things easier, we'll present all the commands below for recreating the Daily/Monthly setup discussed in Chapter 6, "Operational Basics", (starting page 12), plus a couple of extra resources.

Create a new text file called "*create-resources.nsri*", with the following content:

```
create type: NSR policy; name: Daily; period: Weeks; number of periods: 5
create type: NSR policy; name: Monthly; period: Months; number of periods: 13

create type: NSR schedule; name: Daily; period: Week;
action: i i i i i f i; override: skip last friday every month
create type: NSR schedule; name: Monthly; period: Month;
action: s; override: full last friday every month

create type: NSR group; name: Daily; autostart: Enabled;
start time: "21:35"; schedule: Daily
create type: NSR group; name: Daily MSSQL; autostart: Enabled;
start time: "21:55"; schedule: Daily
create type: NSR group; name: Monthly; autostart: Enabled;
start time: "21:40"; schedule: Monthly
create type: NSR group; name: Monthly MSSQL; autostart: Enabled;
start time: "22:00"; schedule: Monthly
```

```
create type: NSR pool; name: Daily; enabled: Yes; pool type: Backup;
groups: Daily, Daily MSSQL; auto media verify: Yes;
recycle to other pools: Yes; recycle from other pools: Yes;
retention policy: Daily
create type: NSR pool; name: Monthly; enabled: Yes; pool type: Backup;
groups: Monthly, Monthly MSSQL; auto media verify: Yes;
recycle to other pools: Yes; recycle from other pools: Yes;
retention policy: Monthly
create type: NSR pool; name: Daily Clone; enabled: Yes;
pool type: Backup Clone; auto media verify: yes;
recycle to other pools: Yes; recycle from other pools: Yes;
retention policy: Daily; store index entries: No
create type: NSR pool; name: Monthly Clone; enabled: Yes;
pool type: Backup Clone; auto media verify: yes;
recycle to other pools: Yes; recycle from other pools: Yes;
retention policy: Monthly; store index entries: No

. type: NSR group; name: Daily
update clones: Yes; clone pool: Daily Clone
. type: NSR group; name: Daily MSSQL
update clones: Yes; clone pool: Daily Clone

. type: NSR group; name: Monthly
update clones: Yes; clone pool: Monthly Clone
. type: NSR group; name: Monthly MSSQL
update clones: yes; clone pool: Monthly Clone
```

Note: You can download pre-created files from the website that contain the above *nsradmin* commands. A single zip file containing both a Unix format file and a Windows format file can be retrieved from:

http://nsrd.info/micromanuals/resources/nsradmin/733_create.zip

When run, this will produce output along the lines of:

```
[root@tara ~]# nsradmin -i create-resources.nsri
created resource id 165.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(1)
created resource id 166.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(1)
created resource id 167.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(1)
created resource id 168.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(1)
created resource id 169.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(1)
created resource id 170.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(1)
created resource id 171.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(1)
created resource id 172.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(1)
created resource id 173.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(1)
created resource id 174.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(1)
created resource id 175.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(1)
created resource id 176.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(1)
Current query set
updated resource id 169.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(2)
Current query set
updated resource id 170.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(2)
Current query set
updated resource id 171.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(2)
Current query set
updated resource id 172.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(2)
```

7.3.4 A Client Creation Script

Now that we've got a basic script done, let's look at a script for adding new clients. This time, let's look at the script in Unix/Linux first. It might resemble the following:

```
#!/usr/bin/perl -w

use strict;

my $hostname = "tara";
print "Enter new client name: ";
my $newClient = <>; chomp $newClient;

print "Should new client have MSSQL module enabled? (y/n) ";
my $module = <>; chomp $module;

if (open(NEWCL,">new-client-$$.nsri")) {
	print NEWCL "create type: NSR client; name: $newClient;\n";
	print NEWCL "group: Daily, Monthly; browse policy: Monthly;\n";
	print NEWCL "retention policy: Monthly; parallelism: 1\n";

	if ($module eq "y") {
		print NEWCL "create type: NSR client; name: $newClient;\n";
		print NEWCL "group: Daily MSSQL, Monthly MSSQL;\n";
		print NEWCL "browse policy: Monthly;\n";
		print NEWCL "retention policy: Monthly;\n";
		print NEWCL "backup command: nsrsqlsv.exe -s $hostname;\n";
		print NEWCL "save set: \"MSSQL:\"\n";
	}
	close(NEWCL);

	system("nsradmin -i new-client-$$.nsri");
	unlink("new-client-$$.nsri");
} else {
	die "Could not create new-client-$$.nsri\n";
}
```

Be sure to change the NetWorker server hostname in the line 'my $hostname = "***tara***"' to the name of your lab NetWorker server.

Sample run sessions from the above script might look like the following:

```
[root@tara ~]# ./create-client.pl
Enter new client name: test11
Should new client have MSSQL module enabled? (y/n) y
created resource id 181.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(1)
created resource id 182.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(1)

[root@tara ~]# ./create-client.pl
Enter new client name: test13
Should new client have MSSQL module enabled? (y/n) n
created resource id 183.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(1)
```

In the first session, you'll note that two resources were created – one for the filesystem backup, the other for the SQL server backup. In the second session, only the filesystem backup instance was created.

Next, let's look at the script required for client creation on Windows.

```
@echo off
set server=cyclops
```

```
echo Creating a new client
set /p name="Enter new client name: "
set /p module="Should new client have MSSQL module enabled? (y/n) "

> command.nsri echo create type: NSR client; name: %name%;
>> command.nsri echo group: Daily, Monthly; browse policy: Monthly;
>> command.nsri echo retention policy: Monthly; parallelism: 1

if %module%==y (
>> command.nsri echo create type: NSR client; name: %name%;
>> command.nsri echo group: Daily MSSQL, Monthly MSSQL;
>> command.nsri echo browse policy: Monthly; retention policy: Monthly;
>> command.nsri echo backup command: nsrsqlsv.exe -s %server%;
>> command.nsri echo save set: "MSSQL:"
)

nsradmin -i command.nsri
del command.nsri
```

Be sure to change the NetWorker server hostname in the line 'set server=***cyclops***' to the name of your lab NetWorker server.

With the script created and saved as "create-client.bat", sample run session results are as follows:

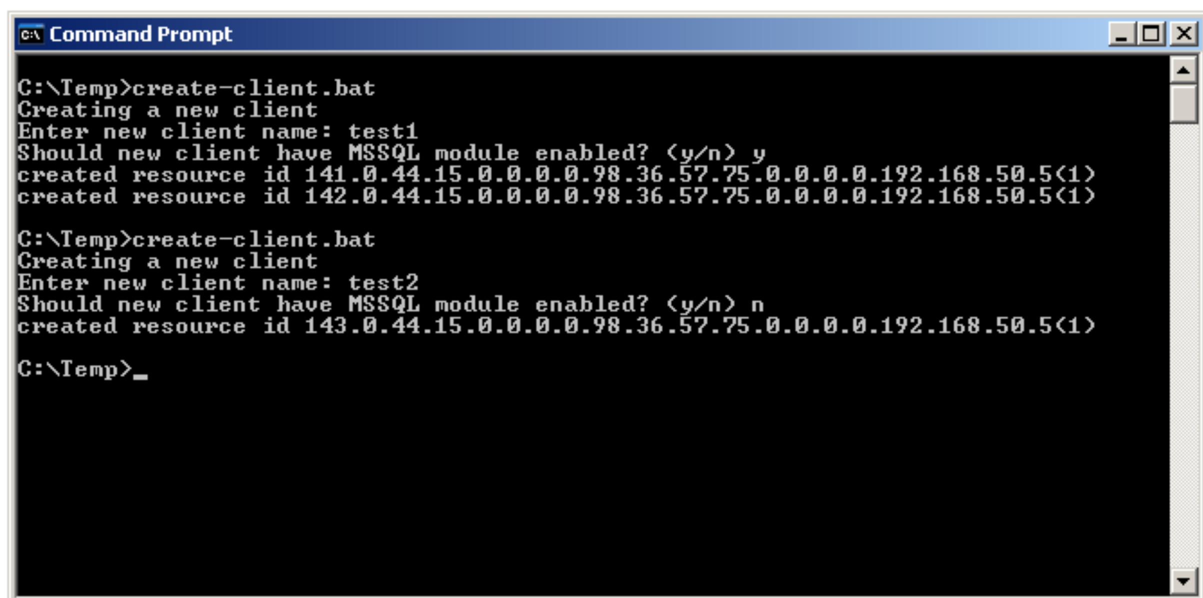

Figure 7: Output from the "create-client.bat" script on Windows

7.4 Connecting to the Client Services

So far, we've concentrated on connecting to the NetWorker *server* daemons. However, *nsradmin* supports connecting to other NetWorker services as well – most notably the client services.

If you'll recall the original usage output for *nsradmin*'s usage details, you'll remember it looks like this:

```
[root@tara ~]# nsradmin -?
usage: nsradmin [-c] [-i file] [-s server] [-p {prognum | progname} ]
                [-v version] [query]...
usage: nsradmin [-c] [-i file] [-d resdir] [-t typefile] ... [query]...
usage: nsradmin [-c] [-i file] [-f resfile] [-t typefile] ... [query]...
```

In order to connect to the *client* service, we'll run *nsradmin* using both the "-s" option, and the "-p" option. The program number in this case is 390113 (if wondering why, run "**rpcinfo –p *yourServerName***" and look for 390113) – or, a program name of nsrexec. Note that when connecting to a client NetWorker service, the "-s" option refers not to the NetWorker server, but the server/host running the *service* you want to connect to.

For the moment restricting ourselves just to the NetWorker server's client service, run *nsradmin* against the server and do a plain "print" command:

```
[root@tara ~]# nsradmin -s tara -p nsrexec
NetWorker administration program.
Use the "help" command for help, "visual" for full-screen mode.
nsradmin> print
                        type: NSRLA;
                        name: tara.pmdg.lab;
 NW instance info operations: ;
       NW instance info file: ;
          installed products: ;
                     version: EMC NetWorker 7.5.1.Build.413 12/11/09;
                     servers: ;
                auth methods: "0.0.0.0/0,nsrauth/oldauth";
               administrator: root, "user=root,host=tara.pmdg.lab";
                 kernel arch: x86_64;
                    CPU type: x86_64;
                machine type: desktop;
                          OS: Linux 2.6.18-128.2.1.el5;
           NetWorker version: 7.5.1.Build.413;
              client OS type: Linux;
                        CPUs: 1;
                     MB used: 12334;
                  IP address: 192.168.50.7, "fe80::21c:42ff:fed3:b3df%eth0";

                        type: NSR remote agent;
                        name: Filesystem;
                 backup type: Filesystem;
             product version: ;
remote agent protocol version: 1;
                    features: Configuration, List Directory;
     remote agent executable: nsrfsra;
            backup type icon: ;

                        type: NSR log;
               administrator: root, "user=root,host=tara.pmdg.lab";
                       owner: NetWorker;
             maximum size MB: 2;
            maximum versions: 10;
        runtime rendered log: ;
                        name: daemon.raw;
                    log path: /nsr/logs/daemon.raw;

                        type: NSR peer information;
               administrator: root, "user=root,host=tara.pmdg.lab";
                        name: nimrod.pmdg.lab;
               peer hostname: nimrod.pmdg.lab;
          Change certificate: ;
    certificate file to load: ;
```

This gives us several pieces of information. (If you do this against an active NetWorker server, by the way, expect to see one "NSR peer information" resource for each NetWorker client that has contacted/communicated with the NetWorker server at least once.)

One of the more useful components reported is the "NSRLA" type. This provides us a chunk of useful information about the client we're communicating with, including, but not limited to:

- The authorisation methods supported;
- The version of NetWorker installed;
- The recognised NetWorker administrators (this list will not be provided if the accessing user doesn't have authority to see that list);
- The architecture and operating system type;
- The amount of space used across its filesystems ("MB used");
- The IP addresses.

This in itself is useful information that can be used to check the status of clients. However, if we look at say, the "NSR log" component:

```
nsradmin> print type: NSR log
                        type: NSR log;
               administrator: root, "user=root,host=tara.pmdg.lab";
                       owner: NetWorker;
             maximum size MB: 2;
            maximum versions: 10;
        runtime rendered log: ;
                        name: daemon.raw;
                    log path: /nsr/logs/daemon.raw;
```

(If you have installed and started NetWorker Management Console, you'll have additional entries here.)

One of the more interesting options here is the "runtime rendered log", which is an attribute which allows you to tell NetWorker where to put a standard "daemon.log" file, in addition to the 7.4.x style daemon.raw file.

Thus, you could tell NetWorker to generate the log by using the command:

```
nsradmin> update runtime rendered log: /nsr/logs/daemon.log
        runtime rendered log: /nsr/logs/daemon.log;
Update? y
updated resource id 12.0.206.8.0.0.0.0.11.164.58.75.0.0.0.0.192.168.50.7(2)
```

Note that for a Windows system, the path to the log should be enclosed in double quotes, and backslashes should be escaped, as per normal for Windows paths in *nsradmin*.

For further examples of adjustments to the *NSR log* resource, go to:

http://nsrd.info/blog/2009/07/28/basics-–-realtime-rendered-logs-and-other-log-options/

Note that changes to the client services typically don't take effect until you stop and restart NetWorker.

Another common area where you can use *nsradmin* on the client services is to correct those pesky "NSR peer information" errors. An article covering this on the NetWorker blog can be found at:

http://nsrd.info/blog/2009/02/23/basics-fixing-nsr-peer-information-errors/

While you're more likely to run *nsradmin* against the server services, knowing that you *can* run it against client services (and how to) is an important tool in the arsenal of a NetWorker Power User.

7.5 Using regular expressions in nsradmin

Perhaps more so than any other topic we've covered so far, it's worth reiterating the importance of always ensuring your *nsradmin* query works properly before running an action against it (e.g., update, append or delete). The reason for this is that if you're someone who uses regular expressions frequently, you may be surprised by how and when they can fail within *nsradmin*.

Teaching how regular expressions work is certainly beyond the scope of this micromanual, as entire books are dedicated to the intricacies of this topic. However, to bastardise the description, let's say that regular expressions (in NetWorker parlance at least) are about being able to use the primary wildcard character, asterisk (*) in queries. (Beyond this, your experience with regular expressions and *nsradmin* will be somewhat … poor.)

Let's start with a basic example – we'd like to print out all the clients that start with "test" in their names. Based on the *Unix* create-client.pl output from the previous section, this might run as follows:

```
nsradmin> option regexp
Regexp display option turned on

Display options:
      Dynamic: Off;
      Hidden: Off;
      Raw I18N: Off;
      Resource ID: Off;
      Regexp: On;
nsradmin> show name:; backup command:; save set:; group:
nsradmin> print type: NSR client; name: test*
                        name: test11;
                       group: Daily, Monthly;
                    save set: All;
              backup command: ;

                        name: test11;
                       group: Daily MSSQL, Monthly MSSQL;
                    save set: "MSSQL:";
              backup command: nsrsqlsv.exe -s tara;

                        name: test13;
                       group: Daily, Monthly;
                    save set: All;
              backup command: ;
```

You'll note that the first thing done was to actually turn on regexp mode. If we hadn't, our session would have looked quite different:

```
# nsradmin
NetWorker administration program.
Use the "help" command for help, "visual" for full-screen mode.
nsradmin> show name:; backup command:; save set:; group:
nsradmin> print type: NSR client; name: test*
No resources found for query:
                        name: test*;
                        type: NSR client;
```

In this case, without regexp turned on, NetWorker expected to literally find a client named "test*", rather than any client whose name started with "test".

Unfortunately, EMC's implementation of regexp support within *nsradmin* is extremely limited at best. Let's look at a command where we want to retrieve all clients whose name *ends* with 11. Based on the previous regexp enabled search, we know this should succeed. However:

```
nsradmin> option regexp
Regexp display option turned on

Display options:
        Dynamic: Off;
        Hidden: Off;
        Raw I18N: Off;
        Resource ID: Off;
        Regexp: On;
nsradmin> print type: NSR client; name: *11
No resources found for query:
                        name: *11;
                        type: NSR client;
```

It is unfortunate that EMC has implemented such a limited version of regexp support in *nsradmin* – better enabled, this would even further extend the scripting and power-user options available within the utility.

With this in mind, that pretty much represents the extent to which regular expressions can currently be used in *nsradmin*.

7.6 Offline Mode

For the majority of this micromanual, we've concentrated on *online* mode access in *nsradmin*, where we connect to a running NetWorker server and work with its running configuration.

What we haven't covered is running *nsradmin* in *offline* mode. This is where, instead of invoking it with just a plain:

```
# nsradmin
```

Or, against a server:

```
C:\> nsradmin –s serverName
```

We run it against a directory containing a resource configuration database.

NOTE: Under no circumstances should you ever run nsradmin in offline mode against a running NetWorker server's configuration database. Doing so could cause irreparable damage necessitating a bootstrap recovery.

To look at offline mode, we'll shutdown our NetWorker services. On Unix/Linux, this can be done with the command:

```
# /etc/init.d/networker stop
```

On Windows, you can do it from the services control snap-in, or you can instead just run, at the command prompt:

```
C:\> net stop nsrexecd /y
```

Once this has been done, we'll check out what it's like to run *nsradmin* against the NetWorker configuration database in offline mode. On Unix/Linux, this might resemble the following:

```
# nsradmin -d /nsr/res/nsrdb
NetWorker administration program.
Use the "help" command for help, "visual" for full-screen mode.
```

```
nsradmin> print type: NSR policy; name: Month
                        type: NSR policy;
                     comment: ;
                        name: Month;
           number of periods: 1;
                      period: Months;
```

You can see there that we've invoked NetWorker differently. We've told it to run against a resource configuration *directory*, located in "/nsr/res/nsrdb".

Moving across to Windows, we can do the same thing, with the result looking like the following:

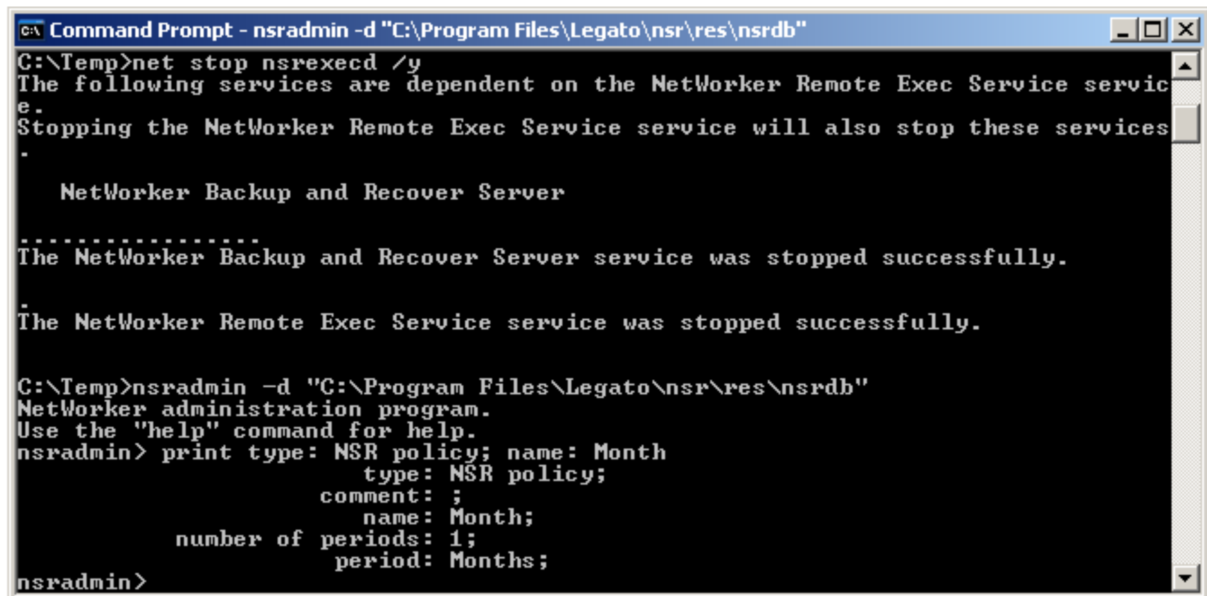

Figure 8: Running nsradmin in offline mode

That's all for offline mode, and for one very critical reason – many of the input validation/safety checks performed by the NetWorker *server* when you work with *nsradmin* don't get done when working in offline mode. Therefore, even as a power-user, you should avoid running *nsradmin* in offline mode against a resource configuration database unless your support provider has advised you to.

8 Appendix A – Test Setup Configuration

As outlined in section "5 How to do the examples" (page 11), the examples in this micromanual assume a testing configuration established on an otherwise unused NetWorker server. This Appendix will take you through the steps required to establish this configuration.

8.1 From NetWorker Management Console

From within the NetWorker management console, create:

- 2 x Disk backup devices using an appropriate path.
  - For Unix/Linux, you might use: /backup1 and /backup2 for Unix/Linux;
  - For Windows, you might use E:\NSR\01 and E:\NSR\02 for Windows.
- These paths must already exist, and should have at least 5GB free.
- A group called Test.
- A pool called Test, with the Test group in it.
- A pool called Test Clone, as a clone pool.
- Mount/label an ADV_FILE volume on the first disk backup path in the Test pool.
- Mount/label an ADV_FILE volume on the second disk backup path in the Test Clone pool.
- Modify the NetWorker server client resource to have a single directory as the save set, picking a directory that is somewhere between 1 and 3 GB. Also modify the client to belong to the Test group.

Note:

- Step-by-step instructions for creating these resources are not supplied for a very specific purpose. This manual is targeted at NetWorker Power Users. If you need instruction on setting up these resources within the NetWorker Management Console, some additional experience in NetWorker would be strongly recommended before attempting this manual.

8.2 From Unix/Linux Command Line

8.2.1 Resource Configuration Setup

Assuming you have **a freshly installed NetWorker server** that has just been started for the first time:

1. Create a directory called "/backup1".
2. Create a directory called "/backup2"[4].
3. Create a text file called "bootstrap-tutorials.nsri" with the content shown in the box below, replacing any instance of the word "***yourServerNameHere***" with your current NetWorker server's name.
4. Run the command (as root): *nsradmin -i bootstrap-tutorials.nsri*

```
create type: NSR device; name: /backup1; media type: adv_file
create type: NSR device; name: /backup2; media type: adv_file
create type: NSR group; name: Test
. type: NSR client; name: yourServerNameHere; save set: All
update group: Test; save set: /usr/share
create type: NSR pool; name: Test; groups: Test; pool type: Backup
```

[4] Directories of "/backup1" and "/backup2" assume (a) at least 5GB free in the root filesystem *and* (b) that you have administrative privileges on the host that you are working from. If this is not the case, consult your system administrator for access to a suitable location.

```
create type: NSR pool; name: Test Clone; store index entries: no; pool type: Backup Clone
```

Note:

- It is recommended to leave a blank line on the bottom of any *nsradmin* script file. This ensures maximum compatibility across platforms.

The output from running the command should appear similar to the following:

```
[root@tara ~]# nsradmin -i bootstrap-tutorials.nsri
created resource id 113.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(1)
created resource id 114.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(1)
created resource id 115.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(1)
Current query set
updated resource id 74.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(2)
created resource id 116.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(1)
created resource id 117.0.152.62.0.0.0.0.154.188.55.75.0.0.0.0.192.168.50.7(1)
```

Note that numbers in the resource IDs are likely to appear differently on your system.

Don't worry if those commands don't yet make sense to you – we'll work on all those and more throughout the micromanual.

8.2.2 Volume Setup

We have created two disk backup units – /backup1 and /backup2. Now we need to label them. We will create a volume called "Backup.001" in the "Test" pool on disk backup unit /backup1, and a volume called "Clone.001" in the "Test Clone" pool on disk backup unit /backup2:

```
[root@tara ~]# nsrmm -b Test -m -l -f /backup1 Backup.001
[root@tara ~]# nsrmm -b "Test Clone" -m -l -f /backup2 Clone.001
[root@tara ~]# nsrmm
adv_file disk Clone.001 mounted on /backup2, write enabled
adv_file disk Clone.001.RO mounted on /backup2/_AF_readonly, write protected
adv_file disk Backup.001.RO mounted on /backup1/_AF_readonly, write protected
adv_file disk Backup.001 mounted on /backup1, write enabled
```

8.3 From Windows Command Line

8.3.1 Resource Configuration Setup

Assuming you have a **freshly installed NetWorker server** that has just been started for the first time:

1. Create a directory called "X:\NSR" (where X is the path to an appropriate drive)
2. Create a directory called "X:\NSR\01" (where X is the path to an appropriate drive)
3. Create a directory called "X:\NSR\02" (where X is the path to an appropriate drive)[5]
4. Create a text file called "*bootstrap-tutorials.nsri*", with the content shown in the box below, replacing any instance of the word "***yourServerNameHere***" with your current NetWorker server's name.

[5] The directories specified should have at least 5GB free on the nominated drive, *and* you have should administrative privileges on the host that you are working from. If this is not the case, consult your system administrator for access to a suitable location.

5. Run the command (as an account in the Administrator group): *nsradmin –i bootstrap-turorials.nsri*

```
create type: NSR device; name: "E:\\NSR\\01"; media type: adv_file
create type: NSR device; name: "E:\\NSR\\02"; media type: adv_file
create type: NSR group; name: Test
. type: NSR client; name: yourServerNameHere; save set: All
update group: Test; save set: "C:\\WINDOWS\\SYSTEM32"
create type: NSR pool; name: Test; groups: Test; pool type: Backup
create type: NSR pool; name: Test Clone; store index entries: no; pool type: Backup Clone
```

Note:

- It is recommended to leave a blank line on the bottom of any *nsradmin* script file. This ensures maximum compatibility across platforms.
- Out of the box on Windows, NetWorker does not support creating disk backup units directly off the root directory (\) of a drive. Please ensure to create the appropriate subdirectory structure to ensure your commands work.
- Adjust paths for disk backup devices and the directory to be backed up appropriately, but remember to always keep Windows paths in double quotes, and whenever a single backslash (\) would be used in Windows, use two (\\) for *nsradmin*.

The output from running this command should appear similar to the following:

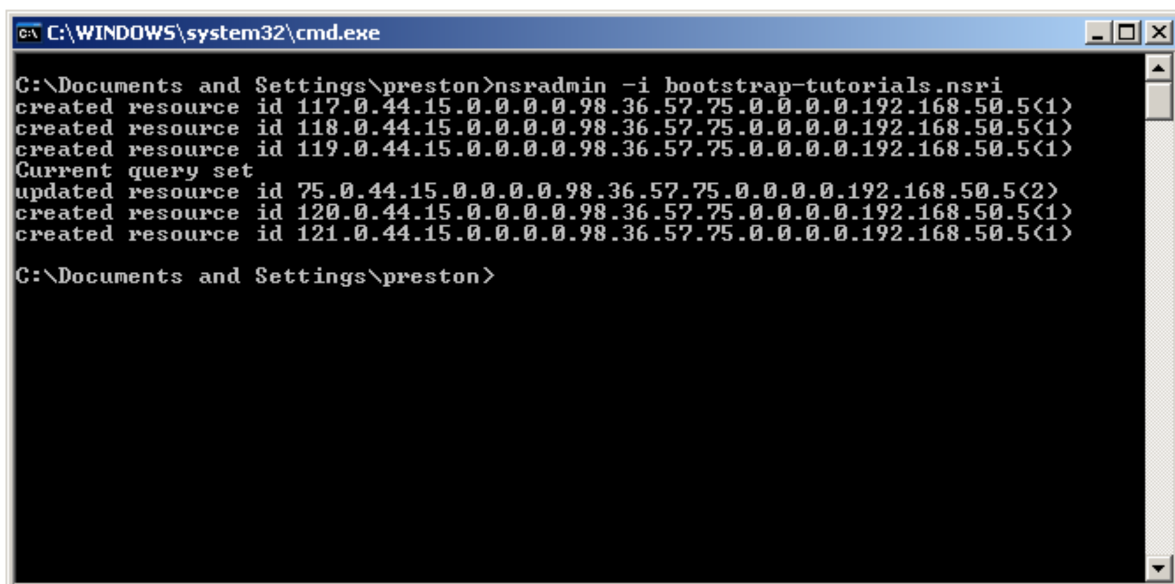

Figure 9: Bootstrapping the NetWorker configuration required for the micromanual on Windows

8.3.2 Volume Setup

We have created two disk backup units. In the example given, these were created at “E:\NSR\01” and “E:\NSR\02”. Now, we need to label them. We will create a volume called Backup.001 in the “Test” pool on disk backup unit “E:\NSR\01”, and a volume called “Clone.001” in the “Test Clone” pool on disk backup unit “E:\NSR\02”. Please adjust your paths accordingly if your devices were created on alternate paths:

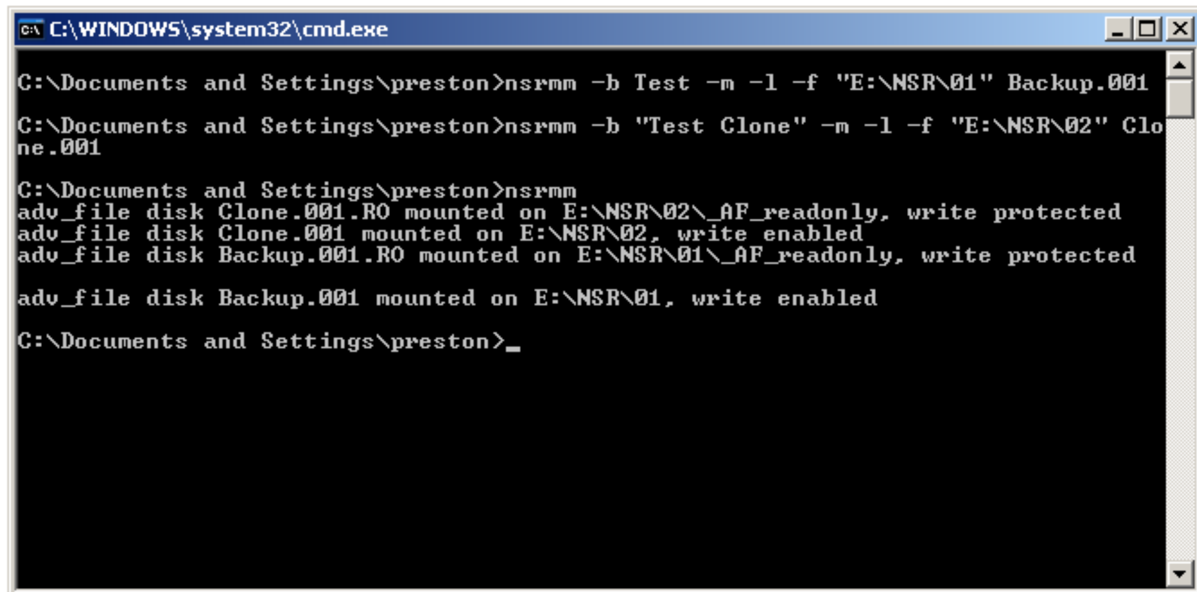

Figure 10: Labelling media in the ADV_FILE devices on Windows

You'll see in the above output, by the way, that it was not necessary to "escape" the backslashes (i.e., use \\ instead of \) in the *nsrmm* commands.